A Parameter-Optimizing Model-Based Approach to the Analysis of Low-SNR Image Sequences for Biological Virus Detection

Dissertation

zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Dominic Siedhoff

Dortmund

2016

Tag der mündlichen Prüfung:	15. September 2016
Dekan / Dekanin:	Prof. Dr.-Ing. Gernot A. Fink
Gutachter / Gutachterinnen:	Prof. Dr. Heinrich Müller
	Prof. Dr.-Ing. Dorit Merhof

Acknowledgments

I would like to thank everybody who supported me during the research that led to this thesis. In particular, I would like to thank my supervisor, Prof. Dr. Heinrich Müller, for his advice and guidance, fruitful discussions, sharing his experience and letting me develop my own. Furthermore, I would like to thank Prof. Dr.-Ing. Dorit Merhof from RWTH Aachen University for kindly agreeing to review my thesis and for improving it with helpful remarks.

In addition to that, I want to thank everybody in the team of project B2 of the collaborative research center SFB 876. I am proud I could be a part of you and grateful for the opportunity to work with you. Besides my direct team mates, I want to thank everybody within SFB 876 with whom I had the opportunity to collaborate over the years. I want to thank all my co-authors, my colleagues across different chairs and departments at TU Dortmund University, as well as my family and friends.

I will not list the names of all of you here. Instead, I simply enjoy the thought that many of you felt addressed more than once while reading this. Thank you all for your friendship and support, for discussions and ideas, and for constant inspiration!

A Parameter-Optimizing Model-Based Approach to the Analysis of Low-SNR Image Sequences for Biological Virus Detection

Abstract:
This thesis presents the multi-objective parameter optimization of a novel image analysis process. The focus of application is automatic detection of nano-objects, for example biological viruses, in real-time. Nano-objects are detected by analyzing time series of images recorded with the *PAMONO* biosensor, after parameters have been optimized on synthetic data created by a signal model for *PAMONO*.

PAMONO, which is short for Plasmon-Assisted Microscopy of Nano-Sized Objects, is a biosensor yielding indirect proofs for objects on the nanometer-scale by measuring the Surface Plasmon Resonance (SPR) effects they cause on the micrometer scale. It is an optical microscopy technique enabling the detection of biological viruses and other nano-objects within a portable device. The *PAMONO* biosensor produces time series of 2-D images on the order of 4000 half-megapixel images per experiment. A particular challenge for automatic analysis of this data emerges from its low Signal-to-Noise Ratio (SNR). Manual analysis takes approximately two days per experiment and analyzing person.

With the automatic analysis process developed in this thesis, occurrences of nano-objects in *PAMONO* data can be counted and displayed in real-time while measurements are being taken. Analysis is divided into a GPU-based detector aiming at high sensitivity, complemented with a machine learning-based classifier aiming at high precision. The analysis process is embedded into a multi-objective optimization approach that automatically adapts algorithm choice and parameters to changes in physical sensor parameters. Such changes occur, for example, during sensor prototype development.

In order to automatically evaluate the objectives undergoing optimization, a signal model for the *PAMONO* sensor is proposed, which serves to synthesize ground truth-annotated data. The parameters of the analysis process are optimized on this synthetic data, and the classifier is learned from it. Hence, the signal model must accurately mimic the data recorded by the sensor, which is achieved by incorporating real sensor data into synthesis.

Both, optimized parameters and the learned classifier, achieve high quality results on the real sensor data to be analyzed: Nano-objects with diameters down to 100 nm are detected reliably in *PAMONO* data. Note that the median SNR over all nano-objects to be detected was below two in the examined experiments with 100 nm objects.

While the presented analysis process can be used for real-time virus detection in *PAMONO* data, the optimization approach can serve in accelerating the advancement of the sensor prototype towards a final setup of its physical parameters: In this scenario, frequent changes in physical sensor parameters make the automatic adaptation of algorithmic process parameters a desirable goal. No expertise concerning the underlying algorithms is required in these use cases, enabling ready applicability in a lab scenario.

Keywords:
Parameter Optimization, Automated Microscopy, Low SNR, Data Analysis, Image Processing, Time Series Analysis, Nano-object Detection, Biological Virus Detection, Biosensor

Contents

Chapter 1

Introduction

Contents

Automation of quantitative microscopy, e.g. counting certain objects of interest in microscopic images, received increasing attention in the last decades. The reason is that the scale of experiments and hence the number of objects to be detected became larger and larger, thus rendering manual evaluation a severe bottleneck. Numerous approaches to alleviate this bottleneck have been proposed, exploiting the increasing image processing power of modern computers to (semi-)automate the time-consuming object detection tasks that otherwise would have to be performed manually. Typically, the development of modality-specific algorithms for automatic analysis takes place after the development of the microscopy technique itself has already been completed. Hence these algorithms focus on established imaging techniques that have been fully evolved. Variations in the physical-world parameters of the microscopy device are small, and the kind of data to be analyzed can be regarded as virtually 'the same' every time.

However, also the *developmental* phase of a novel microscopy technique can largely benefit from early availability of an automatic data analysis process: Developing a new type of microscopy device can be regarded as an experiment with many physical parameters that can be varied. The impact of modifying these physical parameters must be evaluated, in order to identify the most suitable setup of the device for a given purpose. This requires a large number of measurements to be conducted. Being able to automatically analyze the outcomes of these measurements accelerates prototype development because the time otherwise taken for manual examination of possibly large amounts of data is saved. This calls for highly adaptive analysis methods, capable of automatically adapting to changes in the physical parameters of the prototype device. Given such methods, high quality analysis results can be achieved, which is mandatory for attributing any deterioration in results quality to a poor setup of physical device parameters, as opposed to unsuitable processing. Availability of an automatic analysis process is not only beneficial for accelerating prototype development during the experimental phase of a microscopy device but is also of high practical value in everyday lab practice, once the development has been completed.

1.1 Motivation and Relevance

The *PAMONO* biosensor [ZKG+10], which is the focus of application in this thesis, is a novel microscopy technique in development. This prototype development can largely benefit from adaptive methods as described above. Such methods will be developed within this thesis, exemplified with respect to the *PAMONO* biosensor. First of all, to illustrate the relevance of *PAMONO*, a short summary of its applications and capabilities will be given now.

The *PAMONO* biosensor addresses the increasing need for rapid detection of nano-objects, in particular of biological viruses [EMY+08; MRE09]. Methods for rapid detection of newly emerging viruses accelerate clinical diagnoses, aid in reducing the costs for health care, and most importantly combat epidemic spread of viruses [MRE09]. *PAMONO* is a real-time-capable method for indirect, optical detection of nano-objects, e.g. intact biological viruses, in liquid samples. Its capability for virus detection is based on virus-antibody-interactions, thus it can furthermore be used in testing the capability of newly developed antibodies to bind their targets, which is applicable in pharmaceutical research.

PAMONO fulfills most of the desiderata for next generation biosensors as identified in [EMY+08] and [MRE09]: It is sensitive to low concentrations of the target nano-objects [STM+15] and easy-to-operate. By using only commercial off-the-shelf optical components it is inexpensive. It can be realized as a portable device, enabling Point of Care (POC) usage, e.g. in remote locations or at the home of patients. Portability furthermore enables deployment of a network of cooperating biosensor units, which allows monitoring virus prevalence over large geographic scales [LKD+14]. Finally, *PAMONO* virus detection is fast: The attachment of viruses to the sensor surface can be visualized and automatically detected in real-time, provided that suitable data processing is applied. The algorithmic methods to be developed throughout this thesis meet up with this real-time capability, i.e. they enable analysis and visualization of *PAMONO* sensor data in real-time, while measurements are being taken.

Broader Context

When regarded in a broader context, asking for an automatic method for *PAMONO* data analysis means asking for a new, modality-specific method for automated microscopy. With regard to other, already established microscopy modalities, such specific methods exist, e.g. [HBR+08; HBR+12; PKC09; YBC+10; ALN+12; WHS+12; TRS+02; Oli02; ZFS+07; SLN+09; JZK+07; MSB+13]. A frequent characteristic of the tasks solved by these methods is the requirement for detecting low-intensity objects of interest in noisy data. This characteristic is shared by *PAMONO* data analysis and is also encountered on very different scales, e.g. in astroparticle physics for astronomical object detection [DE13; RSV+13; VFB+14; DT14; Ruh13].

Automatic methods often require careful tuning of algorithmic parameters to achieve the highest analysis quality. Furthermore, in case the method builds on supervised machine learning techniques, training data is assumed as input. Hence, the effort of manual image analysis is shifted to parameter tuning and possibly the creation of training data. Particularly the latter is subjective and involves tedious, time-consuming work [HBR+12], just like manual image analysis does. However, avoiding subjectivity and the entailed lack of repeatability of results is one of the desired goals for which automatic analysis methods are used in the first place [HBR+12; Oli02]. This contradiction alleviates as long as the same modality

is considered and measurement conditions are kept constant: Time-consuming parameter tuning and creation of training data have to be done only once, and their results can benefit all further analyses. Subjectivity in training data creation is less of an issue because the trained model remains the *same* over all analyses. Hence it incurs the same subjectivity in a *repeatable* fashion, yielding repeatability of results.

If measurement conditions change, however, ensuring high quality analysis results may require parameters to be re-tuned and new training data to be created, in order to adapt the analysis process to the new conditions. Changes in measurement conditions are frequently encountered during prototype development of new sensor technologies like *PAMONO*. A desire for methods that can adapt to different experimental scenarios has also been reported in cell detection [ALN+12]. One class of highly adaptive methods is constituted by machine learning: Knowledge is abstracted from the training data itself, thus creating predictive models that are specifically adapted to this training data. In case of supervised machine learning this entails the need for creating new training data for each prototypical sensor setup, which can pose a bottleneck. Hence a need for methods facilitating and accelerating the creation of training data can be identified in reaching adaptivity via supervised machine learning. Another way of constructing highly adaptive methods is providing a large number of algorithmic parameters controlling how the data is processed, such that changes in the physical parameters of the sensor setup can be accommodated by changes in algorithmic parameters. However, exhaustive manual tuning of algorithmic parameters for every new sensor configuration examined during prototype development slows down the advancement of the sensor technique. Manual parameter tuning is tedious if the parameter space is small and infeasible if it is large. Furthermore, it is considered "more of an art than a science" [BBB+11] as it is not a systematic approach. Nevertheless, recent improvements of results in image classification benchmarks were often due to finding better parameters for existing approaches, rather than better new approaches [BBB+11]. For these reasons, automating the search for such better parameters is a desirable goal: Reaching this goal means that automatic algorithms are not only used for avoiding the bottleneck of manual data analysis but also for avoiding the entailed bottleneck of configuring the algorithms driving the automatic analysis. This is of particular value if algorithms have to be configured numerous times in order to adapt to changing measurement conditions, as in sensor prototype development.

Having an automatic and automatically adapting analysis process available during sensor prototype development can vastly accelerate this development: The impact of changes made to the sensor setup can be thoroughly studied without the need to manually analyze the data or to manually adapt algorithmic parameters of the analysis process. Different sensor setups can be compared, and those most suitable for different purposes can be identified. Such comparisons largely benefit from the increased repeatability delivered by automation. Additionally, the analysis process can later be used to analyze the data produced by the final sensor setup: Once sensor development is completed, a well-tested analysis process is readily available.

PAMONO Data Analysis

As touched upon above, *PAMONO* is a prototypical biosensor in development, and the previous paragraph fully applies to it. Hence, this thesis devises an adaptive microscopy data analysis process, using *PAMONO* as its application scenario. The analysis process consists of

a nano-object detector and a learned model for classifying detector output. It is embedded in an approach that automatically adapts the arising algorithmic parameters to changing measurement conditions and that uses machine learning to compute the classifying model that separates detector output into correct and spurious responses. In the detector, selections are made among competing algorithms with equivalent input and output behavior, and optimized parameters for the selected algorithms are determined in order to adapt the detector to the data and thus to the respective sensor setup. Both is done in a fully automatic fashion, after a data-driven signal model was seeded, which involves only minor manual effort. The signal model empirically simulates image formation on the *PAMONO* sensor and is used to create large amounts of synthetic ground truth data with respect to which the parameters are optimized and which serves as training data in learning the classifying model.

Manual analysis of the data produced by a single *PAMONO* experiment involves the examination of a sequence on the order of 4000 images and takes approximately two days per analyzing person. A bottleneck this severe impedes large-scale experiments, the attainment of good statistics and enhancing the sensor prototype. In contrast, given algorithmic parameters and a classifying model, an average application of the automatic analysis process takes time on the order of seconds: The real-time capability delivered by the *PAMONO* sensor is retained by this analysis process. The encompassing approach determining optimized parameters and the classifying model can be run overnight, with no interaction required. Running such an optimization is only necessary if measurement conditions were changed in a way that rendered the previous parameters and classifying model unsuitable. After that, an arbitrary number of experiments can be analyzed in real-time, until measurement conditions are changed again. The presented approach neither assumes knowledge about the underlying algorithms and their parameters, nor about machine learning, making it readily applicable by lab personnel in everyday lab practice.

One particular challenge pertaining to *PAMONO* data analysis is that the sequences of images provided by the sensor may exhibit Signal-to-Noise Ratios (SNRs) below two. Previous approaches to signal detection in comparable data already fail at higher SNRs: All algorithms surveyed in [CWG01] fail for SNRs approaching four, while for the best methods surveyed in [SLN+09], an SNR approaching two is the ultimate limit. In order to successfully analyze *PAMONO* data, it is mandatory to push the envelope further by handling SNRs below two. The analysis process developed in this thesis demonstrates that by optimized combination of denoising and other image processing methods, nano-objects in *PAMONO* data exhibiting a median SNR below two can be detected. This enables finding nano-objects with diameters down to 100 nm. For comparison, a median SNR below two means that at least half of the occurring nano-objects are likely to be missed by the best of the algorithms surveyed in [CWG01] and [SLN+09].

In a Nutshell

As a quick summary, the central topics covered in this thesis are the following:

- An automatic analysis process is devised for the data produced by the *PAMONO* sensor, which enables biological virus detection.
- Median SNRs below two are handled by this process which allows the detection of nano-objects with diameters down to 100 nm.

- The analysis process is embedded into an approach which automatically adapts the underlying algorithmic parameters and the classifying model to changes in measurement conditions. One use case of this approach lies in accelerating the advancement of the *PAMONO* prototype towards a final sensor setup.
- No expertise concerning the underlying algorithms or machine learning is assumed in order to operate the developed approach and to conduct analyses with it. This enables ready applicability in a lab scenario.

1.2 Contributions of this Work

Investigating the central topics listed at the end of the previous section led to a number of contributions made by this thesis. Large parts of these contributions have previously been published in the context of peer-reviewed national and international conferences and journals. The corresponding publications are [SWL+11; SLW13; LST+13a; LST+13b; SLW+14; SFL+14; STM+15]. The subsequent text lists and summarizes the individual scientific contributions made by this thesis. Complementary information focusing on the publications in which these contributions were made is given in Appendix C.

Signal Model for the PAMONO Biosensor. An empirical signal model of the *PAMONO* sensor is devised which describes the formation of *PAMONO* imagery. It serves as part of a data-driven method for creating synthetic *PAMONO* images with ground truth information about the contained nano-objects. A small number of manually segmented examples of the nano-objects to be detected is required to seed the signal model. Then it is applied to synthesize a large number of *PAMONO* images annotated with ground truth, with respect to which automatic parameter optimization and supervised learning of a classifying model are carried out. The signal model is validated in terms of its suitability for these two tasks: It is demonstrated that parameters optimized for this synthetic data can be transferred to real sensor data with only minor decrease in analysis quality. Furthermore, it is demonstrated that a classifying model learned from this synthetic data also yields high quality classification of real data.

Analysis Process for PAMONO Data. A two-part analysis process for the data provided by the *PAMONO* sensor is devised. The first part is a highly sensitive nano-object **detector** which identifies spatiotemporal regions in the data that are candidates for being affected by nano-object adhesions. This detection encompasses existing and newly developed methods for

- *PAMONO*-specific image processing (newly developed methods),
- denoising (existing methods),
- time series classification (newly developed methods) and
- segmentation (existing methods).

High sensitivity of this detection is demonstrated. The second part aims at precision: A **classifier** process for separating the nano-object candidate regions provided by the detector into actual nano-objects and spurious detector responses is presented. The central contributions made in this context are as follows:

- A set of features is identified which is used in classifying detector results. It contains features of shape, spatial and spatiotemporal intensities.
- A modular supervised classification process for the resulting data is constructed that builds on existing machine-learning algorithms.
- The modules and algorithms that perform best on the given data are identified.

The classifying model eliminates spurious detector responses, thus complementing the goal of high sensitivity pursued by the detector with corresponding precision. The success of this division into a detector collaborating with a classifier is demonstrated with respect to the task of counting the nano-objects appearing in *PAMONO* data.

Automatic Adaptation to Varying Measurement Conditions. The parameters configuring the detector and the employed classifying model are automatically adapted to changes in measurement conditions and *PAMONO* sensor setup. This is achieved by the *SynOpSis* approach, which integrates synthesis based on the *PAMONO* signal model into an optimization of detector parameters and classifying model, which are then used in the final analysis of the real data provided by the sensor. Configuring the proposed *PAMONO* data analysis process by means of *SynOpSis* yields the first method capable of analyzing *PAMONO* data with particle sizes down to 100 nm. This demonstrates empirically that automatic analysis of *PAMONO* data is feasible. *SynOpSis* combines multi-objective parameter optimization with a parametric image processing pipeline. This pipeline is composed of the nano-object detector and the classifier which embeds machine learning-based computation of a classifying model into this optimization. Automatic evaluation of pipeline objectives is enabled by the synthetic ground truth, with respect to which both, parameters and classifying model, are determined. No familiarity with the algorithms and parameters underlying the detector and classifier is required for scientists to benefit from the adaptivity these methods provide. Furthermore, no experience in optimization or machine learning is assumed. Only synthetic ground truth is needed, which can be obtained using the signal model discussed above, requiring a small number of manually segmented examples of the type of nano-objects to be detected. Hence solely *domain knowledge* concerning the field of application of the *PAMONO* sensor is demanded for using *SynOpSis*. This is a huge advantage for practical application in a lab environment.

Validation. The process proposed for *PAMONO* data analysis and the *SynOpSis* approach for automatic adaptation of its algorithmic parameters and classifying model are validated with respect to synthetic and real sensor data. Their capabilities and limitations are investigated in face of *PAMONO* experiments with varying measurement conditions and increasing difficulty. Pareto-optimal parameter sets are examined to determine their commonalities in order to find out what makes a good parameter set and to assess competing algorithms. Predictability of objective values from parameter sets is investigated. Run times are measured and the real-time-capability of detection and classification is verified: Analysis results can be computed and visualized for the sensor operators while measurements are being taken. Overall, the achieved results are an empirical demonstration of the ways in which image processing and data analysis can benefit from automatic parameter optimization and machine learning, based on synthetic ground truth.

1.3 Organization of the Thesis

The organization of this thesis is as follows. Chapter 2 introduces the *PAMONO* biosensor as the application scenario in which the data analysis to be developed is conducted. It elaborates further on the capabilities and fields of application of *PAMONO* and discusses its underlying physics. Tightly connected to these physics are the properties of the data provided by the sensor, and thus the data analysis task to be solved, which is discussed at the end of that chapter.

Chapter 3 develops the *SynOpSis* approach on an abstract level: It firstly identifies the type of abstract detection and classification task that *PAMONO* data analysis belongs to and then presents *SynOpSis* as a method for solving such tasks. As *SynOpSis* can basically be regarded as an extended image processing pipeline undergoing automatic parameter optimization, literature related to these topics is reviewed. Then the modules for synthesis, detection and classification are developed abstractly, before they are revisited concretely for the *PAMONO* scenario in Chapters 4 to 6. The remainder of Chapter 3 describes the multi-objective optimization of detector and classifier and the employed techniques for desirability-based model selection and for performance estimation.

Chapter 4 is the first of three subsequent chapters that concretise the abstract modules of *SynOpSis* from Chapter 3, custom-tailoring them specifically for the *PAMONO* sensor. In Chapter 4, the creation of synthetic ground truth data is covered, which involves presenting a signal model for the *PAMONO* sensor, along with an empirical method for using this signal model in generating synthetic *PAMONO* imagery.

Chapter 5 concretises the detector for *PAMONO* data analysis: *PAMONO*-specific image- and time series processing techniques are presented along with general methods for denoising and segmentation in the spatiotemporal data provided by the sensor. The parameters of each method employed in the detector are listed and discussed because they are to be optimized by *SynOpSis*.

Chapter 6 presents the modular classifier process used in *PAMONO* data analysis. First of all, the features employed by the classifier are depicted, which are extracted from the output of the detector. Then the modules used in the classification process are described, encompassing class balancing, feature scale normalization, feature selection and finally the examined learning algorithms.

Chapter 7 contains the overall evaluation of applying *SynOpSis* for *PAMONO* data analysis. It introduces the employed experimental data and the physical parameters of the *PAMONO* sensor that were varied in recording that data. Furthermore, it concretises the remaining degrees of freedom of *SynOpSis* that were left open in Chapter 3, in order to separate the description of the method from the description of its *PAMONO*-specific experimental setup. Results are firstly reported for a single *PAMONO* experiment in order to illustrate the outcome of one application of *SynOpSis*. Subsequently, results aggregated over all experiments are regarded and different variants of applying *SynOpSis* are evaluated. Final analysis results are reported per experiment, and the quality of performance estimates, method specificity and real-time-capability are investigated. Furthermore, the Pareto fronts of optimized parameter sets are examined, thus determining what makes a *good* parameter set. Predictability of parameter set quality is assessed by computing response surface models for objectives and other measures of quality over parameter space. Finally, the cross-experiment generalization performance of both, detector parameters and classifying models is evaluated.

Chapter 8 serves as a conclusion to the evaluation in Chapter 7 as well as to the overall thesis. Results are summarized and discussed within a broader scope. Directions for future research, based on the findings of this thesis, are presented.

Each chapter of this thesis builds upon the previous one, hence a linear reading order is recommended. Readers not interested in the *PAMONO* sensor possibly prefer Chapter 3 as it depicts the *SynOpSis* approach on an abstract level and abstractly describes the type of task that can be solved by it. In this case Chapters 4 to 6 may serve as optional exemplifications. For readers interested in *PAMONO*, it is recommended to read all chapters in linear order and to optionally peek ahead from Chapter 3 to Section 7.3, which complements the abstract depiction of *SynOpSis* with a *PAMONO*-specific configuration.

1.4 Acknowledgment

Part of this work has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, Project B2.
URL: `http://sfb876.tu-dortmund.de/`

Chapter 2

Biological Virus Detection with the PAMONO Sensor

Contents

Analysis of the data recorded by the *PAMONO* sensor serves as the central field of application examined in this thesis. *PAMONO* data analysis is the task to be solved, in order to demonstrate the capabilities of the presented approach. In this chapter, the *PAMONO* sensor is presented from three perspectives: Section 2.1 takes an application point of view and presents some of the capabilities and fields of use of the *PAMONO* sensor, with a focus on its primary application, i.e. the detection of biological viruses. Section 2.2 provides a physical point of view and explains how these capabilities are achieved by exploiting the Surface Plasmon Resonance (SPR) effect. Finally, Section 2.3 takes a data-oriented point of view, by describing the data output by the sensor and the desired results to be output by the analysis approach proposed within this thesis.

2.1 PAMONO Capabilities and Applications

Plasmon-Assisted Microscopy of Nano-Sized Objects (*PAMONO*) [ZKG+10] is a method enabling the indirect detection of objects on the nanometer (nm) scale, using equipment from optical microscopy. Conventional optical microscopy is not suitable for observing nano-objects directly, due to the following relationship investigated by Mie [Mie08]: The intensity of the light scattered by an object with a radius smaller than the wavelength of the employed light decreases in the sixth power of the radius of the object. Therefore, the intensity of light scattered by nano-objects is by orders of magnitude smaller than for objects on the micrometer (µm) scale, if visible light is used (wavelength ≈ 380 nm to 740 nm). This impedes direct detection of nano-objects by means of conventional optical microscopy. Furthermore, even if the nano-objects would reflect/emit a sufficient amount of light, the diffraction limit discovered by Abbe [Abb73] still limits optical microscopy to a maximum lateral resolution of ≈ 250 nm, if visible light is used.

The *PAMONO* sensor enables optical detection of nano-objects by exploiting the Surface Plasmon Resonance (SPR) effect: An individual nano-object can be indirectly detected by observing the SPR effect it causes on the micrometer scale. This effect occurs when the nano-object attaches to the sensor surface and hence its occurrence can be taken as an indirect

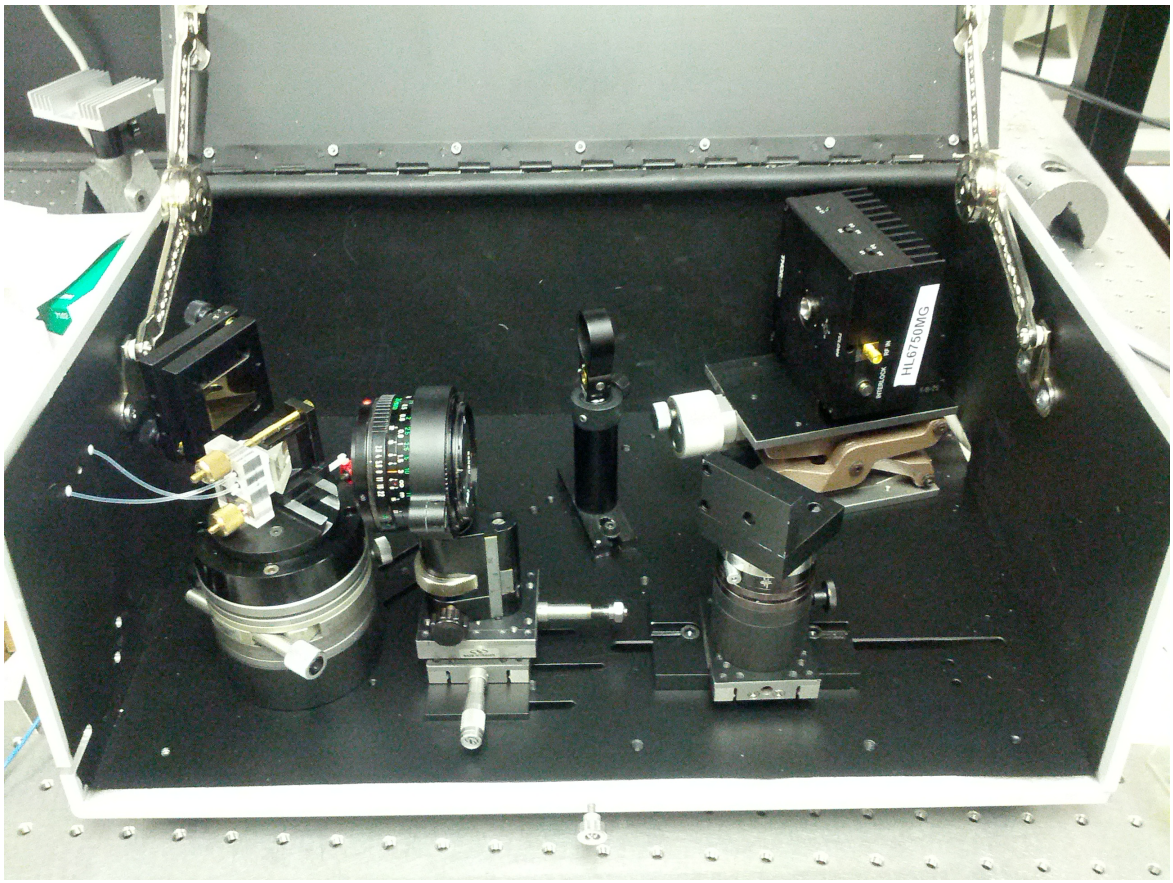


Figure 2.1: Portable PAMONO Sensor. A prototypical *PAMONO* sensor [ZKG+10; WGT+10] is shown, built into a portable case that is approximately 45 cm wide. Processing the sensor data can be handled by a portable laptop computer in real-time [LST+13a; LST+13b]. Details on the components of the sensor are given in Section 2.2 and Figure 2.2. Photograph courtesy of Pascal Libuschewski.

proof for the attaching nano-object. In contrast to the sixth power relation between intensity and object size as in Mie scattering, the observed intensity of the effect in *PAMONO* decreases approximately linearly with object size [ZKG+10]. Therefore, the effect is bright enough to be detected. Furthermore, its spatial extension is on the micrometer scale, hence enabling its detection with conventional optical microscopy techniques. These properties make *PAMONO* a new method for bridging the gap between the micrometer- and the nanometer scale in optical microscopy.

Unlike super-resolution techniques such as Stimulated Emission Depletion (STED) microscopy [HW94] or Stochastic Optical Reconstruction Microscopy (STORM) [RBZ06] it does not rely on fluorescence or any markers. The light observed in *PAMONO* is not emitted by the objects under observation but is due to SPR effects in the surface around them. This makes *PAMONO* more similar to conventional optical microscopy, resulting in comparably low device cost because common off-the-shelf components can be used. In contrast to conventional SPR techniques [BSB+04; CKB+05], not only the concentration of nano-objects in a sample can be estimated but individual nano-objects can be detected because each attachment manifests as a discrete event on the sensor surface.

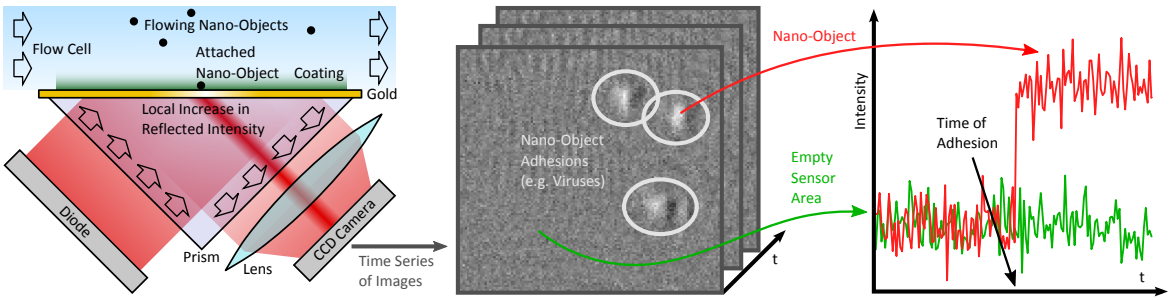


Figure 2.2: PAMONO Sensor and Data. Schematic depiction of the *PAMONO* sensor (left), recording a time series of intensity images (center). Nano-objects attaching to the sensor surface (left) manifest as faint blobs in the spatial domain (center) and as step functions in the temporal domain (right). The displayed step function in the temporal domain was chosen for illustration purposes. Signal-to-Noise Ratios (SNRs) observed in real data make it considerably more difficult to distinguish sensor noise from time series related to nano-object adhesions, cf. Figures 7.1 and 7.2 for more representative example time series. Figure adapted from [WGT+10].

PAMONO can detect any type of nano-object that can be brought to attach to the sensor surface because the attachment is the constituent ingredient for triggering the SPR effect. The primary use case of the sensor is detecting biological viruses. In this scenario, antibodies that are specific to a certain type of virus of interest are used to mobilize the sensor surface, and the viruses are detected as they attach to the antibodies. In contrast to non-optical, nano-resolution-capable microscopy like Electron Microscopy (EM), *PAMONO* does not require a vacuum, hence intact viruses can be indirectly observed in their natural surroundings [WGT+10]. The observation can occur in real-time, enabling e.g. monitoring of the attachment processes or very fast diagnoses. Automatic analysis of the sensor data can be carried out in real-time as well [LST+13a; LST+13b], which will be covered in this thesis. The sensor and the required processing units can be realized in a portable device, making *PAMONO* a technique for virus detection beyond the laboratory environment, cf. Figure 2.1. Furthermore, its principle can be reversed by using a defined sample of viruses and investigating whether or not a newly developed antibody can make the sample viruses attach to the sensor, hence yielding applications in pharmacology. Selectivity of the method can be controlled by making different areas of the sensor surface attach different types of nano-objects: For example different spots of the sensor surface can be coated with different antibodies, enabling the detection of multiple types of viruses on a single sensor. Virus types can then be distinguished by their location. *PAMONO* is, however, not limited to biological virus detection. It can detect any type of nano-object for which there is a method of attaching it to the sensor surface. This makes it applicable e.g. in detecting fine dust and particulate matter in industry or car exhaust.

In summary, *PAMONO* is a versatile technique for indirectly detecting nano-objects by means of inexpensive optical microscopy. The sensor, as well as the processing unit enabling automatic data analysis, can be realized as portable devices, allowing for applications beyond the confined environment of a laboratory.

2.2 The Physics Behind the PAMONO Sensor

The physical principles and details behind the *PAMONO* sensor will be provided in this section. The order of explanation follows the left part of Figure 2.2, which shows a schematic depiction of the sensor setup. This setup is a modified Kretschmann configuration [Kre71] for SPR microscopy. A superluminescent diode emits light through a glass prism upon a thin gold layer on a glass plate that is fixed to the prism. This gold layer constitutes the sensor surface. The light reflected due to total internal reflection (details below) is imaged through a lens upon a 12-bit grayscale industrial Charge-Coupled Device (CCD) camera, which records a time series of images of this reflected light. The top of the gold layer is coated with a substance to which the nano-objects of interest can attach, e.g. antibodies that are selective for a certain virus under observation. The nano-objects are pumped through a flow cell over the gold layer, and by diffusion an assessable amount of them gets close enough to the coating to become permanently attached to the surface. When this happens, the Surface Plasmon Resonance properties in a micrometer scale region around the attachment site change, increasing the intensity of light reflected onto the CCD. Measurement of these small increases of intensity provides the foundation of how SPR-based biosensors work.

The physical explanation of the SPR effect causing these increases is as follows [Pat05]: The light emitted by the diode enters the prism and hits the surface with the gold layer at an angle that is beyond the critical angle for total internal reflection at the given interface between a material with higher refractive index (glass prism) and a material with lower refractive index (liquid in the flow cell). Besides being reflected at the interface, the photons of the light beam can alternatively excite oscillation of the electrons in the gold layer. Such an oscillating electron is called a surface plasmon. Hence a photon can either be reflected towards the CCD, or it can be transformed into a surface plasmon. Every photon becoming a surface plasmon means that less light is reflected onto the CCD. The ratio of photons becoming surface plasmons depends on the incidence angle of the light beam with respect to the gold layer, and the angle maximizing this ratio is called the Surface Plasmon Resonance angle. As a consequence, the SPR angle is the minimizer of the intensity reflected towards the CCD, beyond the angle of total internal reflection. Now this SPR angle sensitively depends on the refractive index of the involved material: A nano-object attaching to the sensor surface (e.g. a virus attaching to an antibody) causes a local change in refractive index at the interface, which entails a change in the local SPR angle. If the light hits the interface at the previous SPR angle, i.e. the minimizer of reflected intensity, any local change in the SPR angle increases the amount of light reflected from the affected region. This enables indirect detection of nano-objects attaching to the sensor surface by finding the regions of increased reflected intensity.

In *PAMONO* the incidence angle of the light emitted by the diode upon the gold layer is chosen as the minimizer of the SPR reflectivity curve, cf. Figure 1 in [ZBM+07], thus following the idea presented above. Hence, any local change in SPR properties causes an increase in the amount of light that is reflected from the interface to the CCD camera. The spatial extension of this increase is a micrometer scale blob (despite being the effect of a nanometer scale cause), therefore its optical detection is possible. On the other hand, the magnitude of the increase in intensity is only a fraction of the intensity of the background signal recorded by the CCD. If

this background signal is computationally removed[1] and the contrast of the result is stretched, the blobs indicating nano-objects look like those displayed in the center of Figure 2.2. The magnitude of the effect in *PAMONO* imaging is nevertheless a significant improvement over direct optical imaging: While the intensity reflected by nano-objects decreases in the sixth power with object size, due to Mie scattering [Mie08], the effect observed in *PAMONO* is related approximately linearly to object size [ZKG+10].

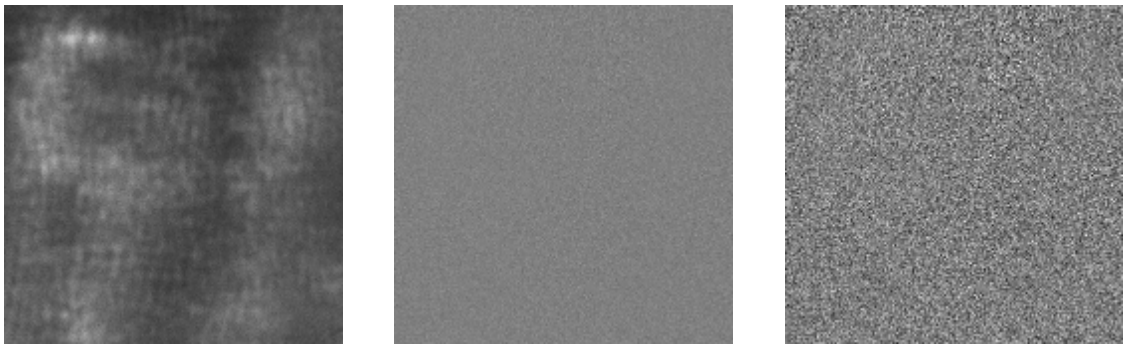
Regarded on a per-pixel level, a nano-object attaching to the sensor manifests as a step function in the recorded per-pixel time series of intensities, cf. Figure 2.2, right. A sufficient number of spatially coherent pixels recording a step function at the same point of time serves as the indirect proof of the attaching nano-object. In areas not affected by nano-objects, only noise on an approximately constant background is recorded. This is what enables optical, though indirect, detection of nano-objects in a microscopy device. The principle is summarized in the acronym: Plasmon-Assisted Microscopy of Nano-Sized Objects (*PAMONO*). Note that the time series on the right of Figure 2.2 were chosen for the purpose of illustration. They do not provide a representative impression of real data: The Signal-to-Noise Ratios (SNRs) observed in real data typically are lower, such that distinguishing time series related to nano-object adhesions from those related to empty sensor areas by visual inspection is considerably more difficult. Representative examples are given in the evaluation chapter, cf. Figures 7.1 and 7.2.

Proportionality between virus concentration in the sample and the number of attachments observed via *PAMONO* has been established by Shpacovitch et al. [STM+15]. Specificity of the sensor responses to the viruses to be found has been demonstrated in the same work. Further details concerning the physical principles behind the *PAMONO* sensor can be found in [ZKG+10]. A mathematical model of the data provided by the sensor will be presented in Chapter 4, which will also guide the image processing and classification techniques discussed in Chapters 5 and 6.

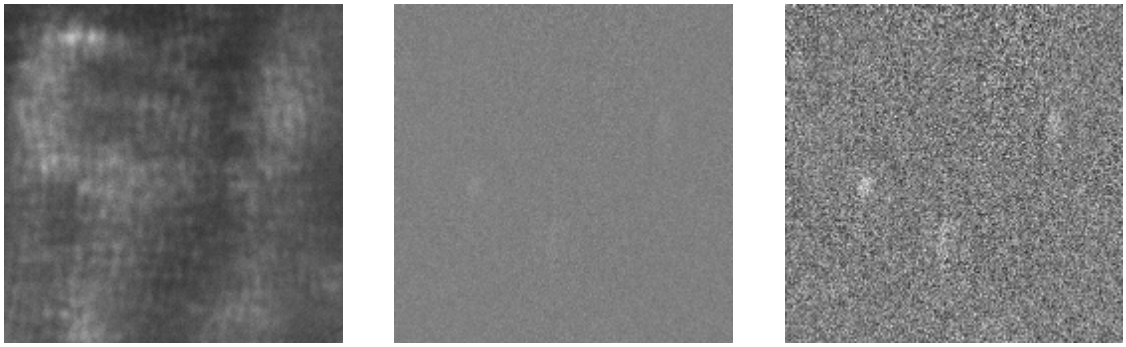
2.3 PAMONO Data and Analysis Task

Conducting a measurement with the *PAMONO* sensor delivers a time series of images, each recording the intensity of light originating from the sensor surface in the direction of the camera. This time series of images can be considered as a spatiotemporal volume of intensities. Figure 2.3 shows four of these images to which different processing has been applied, which serves to illustrate the nature of the data. Each column corresponds to one type of processing and each row corresponds to one of the four exemplary images. In total, the measurement consists of 4000 images, hence the last row displays the final image. The left column shows the raw data as it is recorded by the sensor. Comparing raw image 100 to raw image 4000 reveals no difference that is apparent to visual inspection. What can be observed are interference patterns on the sensor surface that remain approximately constant over all images and that dominate the desired signal in the data. Hence, the center column shows differences between the image indicated by the respective row and the first image in the measurement. In these difference images, the approximately constant interference patterns are eliminated and the desired signal is revealed on the same intensity scale as in the raw images of the left column.

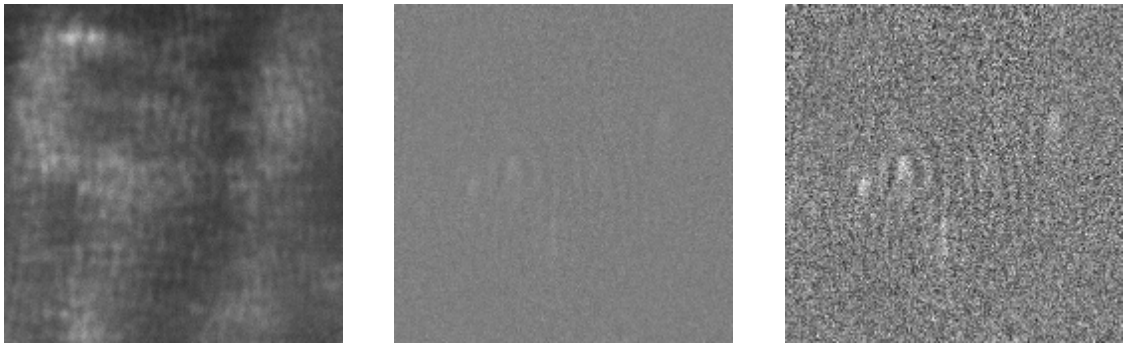
[1] As a lookahead: A model of *PAMONO* signal formation, including the background signal, is provided in Chapter 4, and a method for removing the background signal is described in Section 5.2. However, these are not vital for understanding the current chapter.



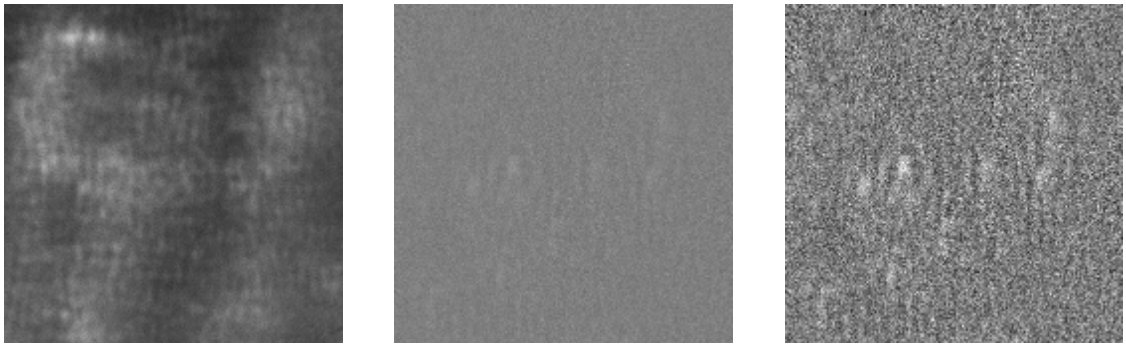
(a) Image 100



(b) Image 1000



(c) Image 2000



(d) Image 4000

Figure 2.3: Time Series of PAMONO Images. Exemplary images recorded by the sensor are shown as raw data (left column), and as difference images between the respective shown raw image and the first raw image of the series without (center) and with amplification (right column).

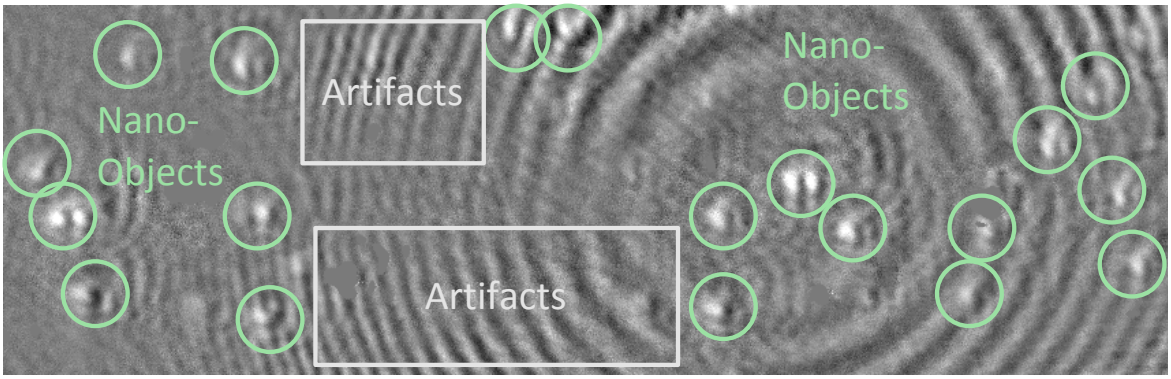


Figure 2.4: Data and Desired Output. For visualization purposes, a highly processed sensor image is shown, giving an example of the output desired from the analysis process: The goal is to find and count all nano-object adhesions (marked by circles), without mistakenly counting the structures that are due to sensing artifacts (marked by rectangles) as nano-objects.

In image 4000, faint blob-like structures are (not easily) perceivable which correspond to nano-object adhesions. The right column shows the same images as the center column, but on a different intensity scale: The contrast in all images has been linearly increased by a factor of five, revealing more and fainter blobs while amplifying sensor noise. The noise is due to the CCD (chip readout noise [FM06]) and photon statistics (shot noise [BB00]) and is aggravated by the derivative-like nature of taking a difference image.

The goal of quantitative analysis of this data is to find the attachments of nano-objects to the sensor surface by finding their characteristic spatiotemporal signatures, which are the sets of spatially coherent and temporally coincident step functions described in the previous section. Figure 2.3 also serves to give examples of these signatures and their magnitudes in comparison to noise and to the background signal. Since the volume of data is large (e.g. 4000 images with 750 px × 230 px in one measurement) and finding nano-object signatures manually is slow, subjective, tedious and error-prone [HBR+12; Oli02], automation of this analysis is desirable. The sought after output of an analysis process is illustrated in Figure 2.4. For better visualization and hence clarity of presentation, the image has been processed with more sophisticated methods than the processing described in the context of Figure 2.3. Details of these methods are presented in Chapter 5. The circles in Figure 2.4 enclose the nano-object attachments, as observed via *PAMONO*. Finding *all* of and *only* these nano-objects in a time series of images are the two subtasks constituting the primary goal in the desired automatic quantitative analysis process. Finding *all* nano-objects is impeded by the low SNR in the data[2], while finding *only* the actual nano-objects is impeded by the sensor artifacts marked by rectangles in Figure 2.4. These artifacts have a temporal signature and intensity that is similar to actual nano-objects. An analysis aiming at finding *all* nano-objects is very sensitive, thus increasing the chance of erroneously reporting artifacts as nano-objects. Solving both subtasks with sufficient quality enables automatic counting of the nano-objects in the data. Hence conclusions about the concentration of nano-objects in the sample can be drawn [STM+15]. Furthermore, by finding individual attachment sites, it is e.g. possible to monitor differences in the attachment behaviors over several nano-objects and surface mobilizations.

[2]Figure 2.4 shows an experiment with a rather high SNR for better visualization, while Figure 2.3 gives a better impression of commonly encountered SNRs.

Further goals of the analysis process are related to the resource-constrained scenario of a portable sensor device: The first requirement is that the analysis process must run on a portable computer, ideally an embedded system. This results in constraints on the allowable energy consumption of the employed algorithms: The lower the energy consumption, the more measurements can be done without recharging the battery of the portable device. In addition to that, it is desirable that the analysis process can be executed in real-time because then the data can be analyzed and displayed already during measurement. The goals of low-energy real-time processing on an embedded system are treated in [NLE+15], while the aspect of real-time capability is also touched upon in Chapters 5 to 7 of this thesis.

The last goal is associated with the aspect of facilitating further improvement and development of the *PAMONO* technique: Due to its prototypical state, the sensor setup described in Section 2.2 involves many physical parameters undergoing variation between experiments, e.g. the quality of the gold layer, type of light source, wavelength and intensity of its emitted light, type of camera, distance between sensor and camera, type and magnification factor of the lens, observed image section, and different buffer solutions in the flow cell. All these physical parameters can have an impact on the appearance of the images recorded by the sensor. In order to maximize the quality of the analysis results, a flexible approach is required that adapts the involved algorithmic parameters to best suit the given physical parameters in the sensor setup.

Chapter 3

The SynOpSis Approach

Contents

In this chapter, the foundation is built for presenting the methods developed to solve the *PAMONO* data analysis task described in Section 2.3. This foundation provides a modular frame into which the *PAMONO*-specific components are later incorporated. Section 3.1 gives an *abstract* description of the *PAMONO* data analysis task, hence presenting its

conception adopted in this thesis. The perspective is problem-oriented, instead of being application-oriented as in the previous section. This abstract view on the problems involved in the task leads to Section 3.2 proposing a methodological approach to solving problems that fit the abstract task description. This approach is denoted the *SynOpSis* approach (Synthesis/Optimization/Analysis) because it creates synthetic datasets and uses them in optimizing algorithms which are subsequently applied in real data analysis. *SynOpSis* is the outer frame into which the application-specific modules for *PAMONO* data analysis are inserted. The order of presentation in this chapter, as well as in the rest of the thesis, is top-down. Hence, Section 3.2 gives an overview of the *SynOpSis* approach on a high level of abstraction, while its constituent components are depicted in more detail in the remaining sections of Chapter 3. As these sections are organized along the workflow of *SynOpSis*, Section 3.2 also serves as an overview of this chapter.

3.1 Abstract Task Description

After Section 2.3 gave an account of the concrete task of *PAMONO* data analysis, this section restates that task on an abstract level, to prepare for Section 3.2 introducing the approach proposed to solve tasks fitting this abstract description. Section 2.3 identified finding *all* and *only* the nano-objects in a time series of images as the central task in *PAMONO* data analysis. Talking about this in an abstract fashion requires more abstract terminology:

Terminology 3.1. *The term **pattern** is in the following defined as a region that is a candidate for containing an object of interest in the data. Candidate regions and hence patterns are characterized by being salient in an application-specific sense of the term 'saliency'. The process of identifying such regions is called **pattern detection**[1]. Two classes of patterns can be distinguished, and this distinction is defined by their causations: Firstly, a pattern can be caused by an object of interest. Such a pattern is called a **target pattern**. Secondly, a pattern can be caused by a spurious signal that is not associated with an object of interest. Such a pattern is called a **non-target pattern**.*

Exemplary causes of target patterns are the nano-objects in *PAMONO*. Exemplary causes of non-target patterns are the artifact and background signals in *PAMONO*. The concrete task of finding *all* and *only* the nano-objects in a time series of *PAMONO* images can then be stated abstractly as finding all target patterns in a given dataset and, if given, filtering out the non-target patterns incurred as "by-catch". Hence the abstract analysis task can be divided into two subtasks: The first is a detection task: *All* target patterns in a given input dataset have to be detected. Aiming at not missing any target patterns requires a highly sensitive detector, which in turn might yield, besides the target patterns, a high number of non-target patterns. The second subtask is a classification task: Its goal is to separate the set of detected patterns into target and non-target patterns. Hence the overall task is detecting and classifying a finite number of patterns with (ideally) well-defined and distinct appearances for the target and non-target class. This is the conception of the abstract task to be solved as adopted in the thesis.

Figure 3.1 illustrates the abstract task and its relation to *PAMONO* data analysis. The input consist of images in which all objects of interest are to be found. A detector finds a set

[1] The term 'pattern detection' is usually defined more broadly, but for consistent terminology, this thesis employs a more narrow definition.

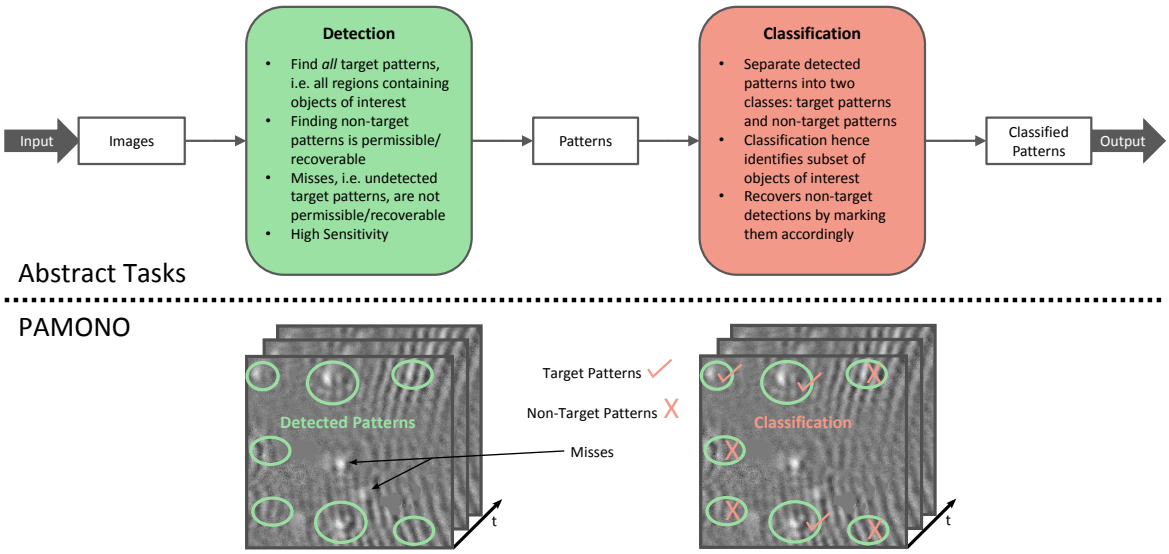


Figure 3.1: Abstract Task Description and *PAMONO* Correlates. The upper part of the figure displays the abstract conception of the *PAMONO* data analysis task, as adopted in this thesis. The overall task is regarded as divided into two subtasks: The first subtask is detection, aimed at finding the set of *all* patterns in the data, where a pattern is defined as a region that is a candidate for containing an object of interest. The second subtask is classification, aimed at restricting this set to *only* the patterns caused by actual objects of interest. The lower part of the figure illustrates these tasks by showing their correlates in *PAMONO* data analysis. Here the patterns of interest are caused by actual nano-object adhesions, while those not of interest are spurious detector responses due to sensing artifacts as seen e.g. in Figure 2.4.

of patterns under the policy that non-target detector responses are permissible, while not responding to a target pattern, and hence missing an object of interest in the data, is not permissible. The reason for this policy is that non-target patterns can be sorted out in the subsequent classification, while missing target patterns can not be recovered. The output of classification is the set of classified patterns.

Section 3.2 now introduces one possible approach to solve tasks like the one described in this section. With this approach it is possible to conduct quantitative analysis of *PAMONO* data, hence it can be used to solve the concrete task described in Section 2.3.

3.2 SynOpSis Overview

As stated in Section 2.3, the volume of data to be processed within a *PAMONO* measurement is very large, and manual analysis is slow, subjective, tedious and error-prone [HBR+12; Oli02]. Hence full automation of quantitative analysis is desired. The algorithms to be used for solving the detection and classification subtasks identified in the previous section operate in a fully automatic manner, once suitable algorithmic parameters are known. This makes automatically analyzing a large volume of data equivalent to automatically determining suitable algorithmic parameters from a possibly large parameter space. Suitability of points in that parameter space is tightly connected to the physical parameters defining the experimental conditions in *PAMONO* data acquisition: The latter are subject to change due to *PAMONO* prototype development, and careful adaptation of algorithmic parameters to changes in physical parameters is necessary for high quality analysis results. While performing this

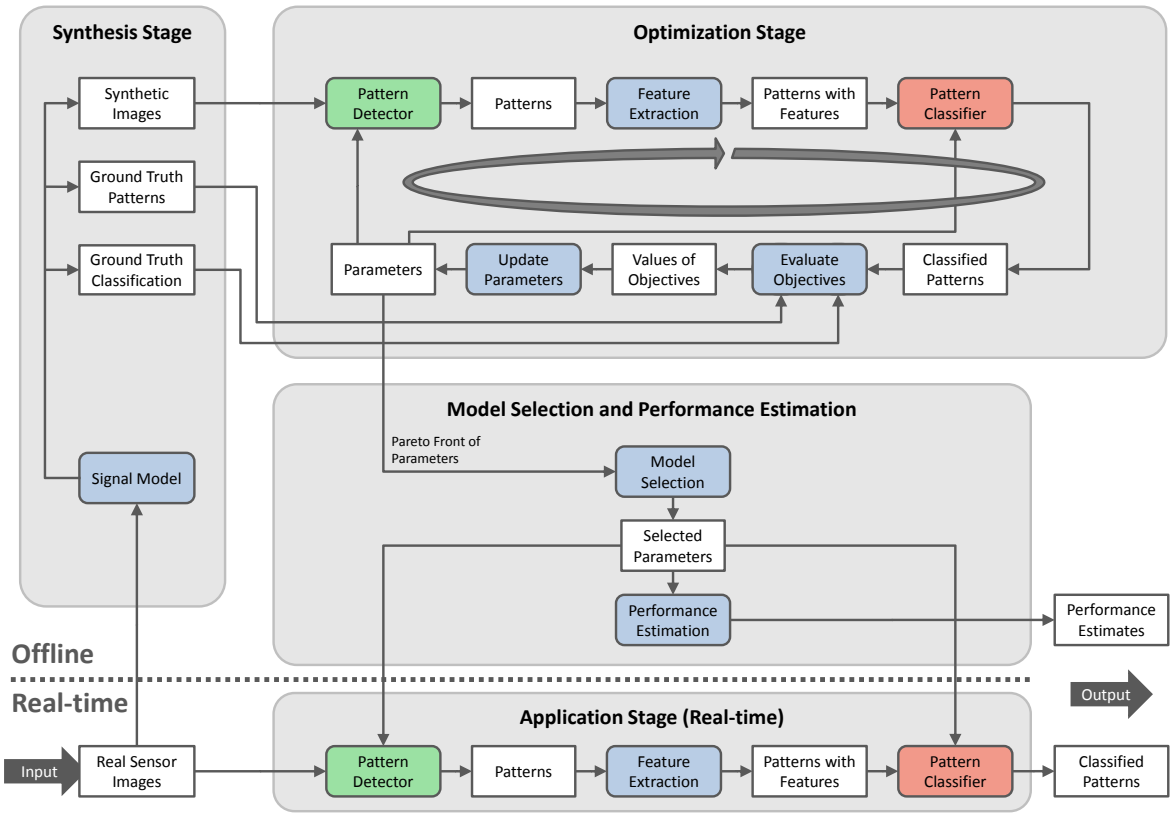


Figure 3.2: Overview of *SynOpSis* Approach. Real sensor data is used as the input of a *Synthesis* stage generating ground truth-annotated training data. The training data is used in an *Optimization* stage to automatically find Pareto-optimal parameter sets for a pattern detector and a pattern classifier. Model selection with desirability functions is applied to determine the most desirable parameters for both, by running them on synthetic validation data. The performance of detector parameters and classifying model on unseen real data is estimated by applying them to unseen synthetic test data. Finally, both are passed to the real-time capable application stage where they are applied to the real input data, producing the final detection/classification results. Section 3.2 describes this figure in more detail.

adaptation of parameters manually is already more convenient than full manual data analysis, it provides no information about the estimated quality of the attained results, and if the parameter space grows larger, its manual exploration is severely aggravated.

In order to attain high quality analysis results and quality estimates in a fully automatic fashion, an approach of data synthesis and optimization is proposed, which is hence denoted as *SynOpSis* (Synthesis/Optimization/Analysis): A *Synthesis* stage generates synthetic images mimicking the signal properties of the real input images to be analyzed. The synthetic images are annotated with ground truth about the target patterns[2] they contain. This ground truth is known because the target patterns are created synthetically at defined locations, using a signal model. Ground truth about the non-target patterns is known implicitly because any

[2]In addition to that, in case multiple classes of target patterns are to be distinguished, the target patterns can be labeled with their individual classes. *SynOpSis* supports multi-class target patterns, however, the discussion given throughout this thesis is limited to the case of a single target pattern class that has to be distinguished from the non-target class. Though, where applicable, additional information on multi-class support is provided in footnotes.

detected pattern that does not correspond to a ground truth target pattern is a non-target pattern. Knowing the locations and classes of the patterns detected in the synthetic images enables the definition and automatic evaluation of objective functions measuring detection and classification quality. Optimizing these objective functions with respect to the algorithmic parameters involved in the data analysis process is a method to automatically obtain suitable points in a large parameter space.

The *SynOpSis* approach will be presented abstractly in the remainder of this chapter. Furthermore, the subsequent chapters of this thesis will complement this abstract depiction by specializing the presented components towards *PAMONO*, thus separating the concept behind *SynOpSis* from the concrete algorithms to be employed in processing *PAMONO* data.

Figure 3.2 gives an overview of *SynOpSis* as a whole and serves as a guide through the rest of this thesis. The brief description given now follows the flow of data in the figure from input to output and shortly introduces the components of *SynOpSis*. In the figure, rectangles represent data tokens and rounded rectangles represent processing steps. The input of *SynOpSis* consists of the real images to be analyzed. Before this analysis takes place (bottommost part of the figure), the *Synthesis* stage is run (leftmost part of the figure): A signal model (cf. Section 3.4) is applied to produce ground truth-annotated synthetic training data. This data consists of synthetically generated images with target patterns, both mimicking the signal properties of the real input images. By being synthetically constructed, location and class of the target patterns are known. Hence an ideal detection and classification result is available with respect to which the quality of computed detections and classifications can be evaluated. After the *Synthesis* stage, the *Optimization* stage is run (top right part of the figure). In this stage, the pattern detector (cf. Section 3.5) is run on the synthetic training images using initial detector parameters. It detects a set of patterns. These patterns are then annotated with features to be used in pattern classification. The pattern classifier (cf. Section 3.6) receives as inputs the feature-annotated patterns and initial classifier parameters. From these inputs it computes a model that classifies the patterns. The detected patterns and their classes as assigned automatically by the pattern classifier are then compared to the synthetic ground truth, and the quality of the achieved detection and classification is assessed by evaluating suitable objective functions based on this comparison. Besides measuring results quality, these objective functions can also be regarded as measures of the quality of the algorithmic parameters employed in the detector and classifier. Given these measures, the parameters are updated in a promising direction of the search space of detector and classifier parameters, and the loop body is run again (cf. Section 3.7 for more details on the *Optimization* stage). After the loop terminates, the set of points in parameter space that are non-dominated[3] in objective space is examined to determine the single most desirable non-dominated point. This is done by computing a desirability index (cf. Section 3.8) of the objective functions within a statistical model selection (cf. Section 3.9). Furthermore, a performance estimate is computed by running detector and classifier with the selected parameters on unseen synthetic test data. This estimate consists of evaluations of performance measures that are relevant for the analysis task. It estimates the performance of the given parameters on unseen data and is one of the two outputs of the *SynOpSis* approach. Finally, as optimized detector and classifier parameters are known, both are applied to the real data to be analyzed (cf. Section 3.10): The real sensor images are input to the

[3]Throughout this thesis, the concept of Pareto-optimality [Deb01] is used to define the point set that performs best with respect to objective space. This concept is discussed in more detail in Section 3.7.4.

pattern detector, using the optimized detector parameters, and yielding patterns in the real data. Features are extracted from these patterns, which are then employed in classification. The pattern classifier uses a classifying model trained beforehand from synthetic ground truth. The learning algorithm used to compute the classifying model receives the parameters determined during the *Optimization* stage. The computed classification aims at labeling target and non-target detector responses as such. The classified patterns are output as the analysis result. For the pattern detector and classifier proposed in the context of *PAMONO* (Chapters 5 to 6), the analysis can be performed in real-time, once detector and classifier parameters (involving the classifying model) have been determined [LST+13a; LST+13b]. That means the *Application* stage in Figure 3.2 is real-time capable for *PAMONO* data analysis. As long as the sensor setup and the type of nano-objects in *PAMONO* do not change, the offline *Optimization* stage need not be executed again. Hence series of consecutive *PAMONO* measurements with unchanged sensor setup and nano-object type can be performed in real-time and rapid succession.

As can be seen from comparing Figures 2.4 and 3.1, data analysis in *PAMONO* as described in Section 2.3 falls into the category of the abstract task description from Section 3.1. Hence the task has a structure that makes it solvable with *SynOpSis*. The next section gives an overview of related work in the field of automatic tuning of algorithmic parameters, as well as in the field of image processing pipelines. All sections after that follow the flow of the data in Figure 3.2 and provide more details on each processing step. Finally, Chapters 4 to 6 provide a signal model, pattern detector and pattern classifier that are custom-tailored to the *PAMONO* application scenario.

3.3 Related Work

In presenting the literature related to the topics of this work, the same top-down approach as in the overall thesis is followed. Hence this section starts by firstly regarding the superordinate topic of automatic tuning of the parameters of given algorithms. Abstractly speaking, this parameter tuning can be regarded as an outer loop around the image processing pipeline in *SynOpSis* (cf. *Optimization* stage in Figure 3.2). Secondly, literature related to the components of this image processing pipeline, especially to the pattern detection and classification components, is discussed, and the relations and differences to *SynOpSis* and to *PAMONO* data analysis are identified.

Automatic Tuning of Algorithmic Parameters

Existing literature on the automatic tuning of algorithmic parameters employs different terminologies with regard to the same concepts, often depending on the context of application. Hence this first paragraph introduces the semantics behind the terminology as it is used throughout the following discussion with regard to *all* presented works, independent of the terminology employed in the individual paper. By algorithmic **parameters** the variables configuring a given algorithm are denoted. These parameters are often referred to as hyperparameters in the context of machine learning algorithms. They have to be clearly distinguished from the data to which an algorithm is applied. A set of parameters encompassing values for all free parameters of an algorithm is called a **parameter set**, sometimes referred to as a configuration of the algorithm in the literature. A parameter set yields a certain

instance of a given algorithm, enabling it to be applied to its input data. Each parameter can assume a certain set of values (discrete or continuous), and each such set of values spans one dimension in the **parameter space** of the algorithm. Any valid parameter set is a point in that parameter space. Finding an optimized parameter set for an algorithm is called **tuning**. A measure of algorithm performance that is optimized during tuning is called an **objective** of that optimization. Running an algorithm for a certain input dataset and parameter set, followed by measuring an objective is called an **evaluation** of that objective. For a fixed input and variable parameter sets, the process of evaluation establishes a mapping between the parameter space and the **objective space**, which can have one dimension (single-objective optimization) or more (multi-objective optimization). In a one-dimensional objective space, this mapping is referred to as the **response surface**. Since most of the presented approaches are designed for single-objective optimization, general discussion uses the singular word 'objective', even in contexts where multiple objectives are conceivable.

The methods for tuning algorithmic parameters to be presented now have been divided into three categories:

1. **Model-free methods** traverse the parameters space solely by actual evaluations of the objective to find optimized parameter sets [HHL+09; BSP+02; BYB+10].
2. **Meta-modeling methods** create a model of the response surface and use it as a proxy function in traversing parameter space. Predictions by the model can be computed quickly and can guide the search [BLP10; KKF+11; HHL11; BBB+11; BMT+12].
3. **Evolutionary methods** maintain parameter sets and their attained objective values in a population that undergoes simulated evolution to breed new promising parameter sets [AST09; MMB+14a; MMB+14b].

Model-free methods for parameter tuning attain knowledge about the response surface solely by actually running the algorithm to be tuned. They do not rely on a model predicting the objective values for new parameter sets.

ParamILS [HHL+09] is a local search strategy which assumes categorical[4] parameters. It randomly searches the finite set of possible values of a single parameter until the objective to be optimized improves, and then greedily continues this procedure with the next parameter until convergence to a local optimum. This local optimum is then perturbed in parameter space to enable escaping from locality and the procedure is repeated. ParamILS can be used in optimizing single objectives over categorical parameters and has been successfully applied, among others, in configuring the CPLEX solver with 63 parameters.

F-Race [BSP+02] is representative of the class of racing approaches [MM97]. It assumes that a finite set of candidate parameter sets has already been sampled from the parameter space (or that the parameter space itself is finite). It selects the best-performing parameter sets from the sample by successively eliminating less promising candidates. The latter are identified by a Friedman test [Con99], once statistical evidence has been collected for their significantly worse performance in comparison to the other parameter sets. Evidence is collected over multiple input datasets, i.e. F-Race furthermore assumes the possibility to sample an arbitrary number of dataset instances from a distribution that is representative of the actual datasets to which the algorithm will be applied. Due to the fact that F-Race

[4] Categorical variables can assume values from a finite set that need not be ordered. An example of such a set is $\{\text{red}, \text{green}, \text{blue}\}$.

assumes a finite number of candidate parameter sets, in order to apply it within the *PAMONO* scenario, where the parameter space consists of 28 to 31 binary, integer and floating point parameters, the problem is shifted to finding a set of promising candidates for racing.

Iterated F-Race [BYB+10] is an extended version of F-Race that can be used to address this issue: It executes F-Race in an iterative fashion, where the racing candidates in each iteration are sampled from the full parameter space, and sampling is biased towards regions containing parameters that performed well in previous iterations. Implementing a search strategy that is guided by previous results takes iterated F-Race conceptually closer to model-based approaches because known results are used to make forecasts about the behavior of the objective function over the parameter space. However, the usual notion of model-based approaches involves the explicit construction of a so-called meta-model, which will be explained in the following.

Meta-modeling methods, in contrast to model-free methods, do not traverse the parameter space by actually running the algorithm for evaluating the objective to be optimized, but by using a surrogate function that is designed to approximate the objective. This surrogate function is called the meta-model (or response surface model [JSW98]). Its purpose is to capture the relation between points in parameter space and the associated objective value with respect to which the parameters are optimized. In contrast to running the actual algorithm and obtaining an exact evaluation of the objective, the meta-model is used to predict the objective value, which is faster than an exact evaluation. Hence the meta-model can guide the search through parameter space by enabling many objective predictions in a fast manner, which is used to identify a promising candidate parameter set. Once the most promising candidate parameter set has been found, the actual algorithm is evaluated with this parameter set, and the resulting exact objective value is used to update the meta-model. Each update improves its fit to the actual response surface, the entirety of which remains unknown. However, the mapping between parameter space and objective values is approximated by this response surface model. The update procedure makes meta-modeling inherently sequential: In each iteration, the newly evaluated point in parameter space helps improving the meta-model by being incorporated as an update. The meta-model-guided optimization procedure is iterated until a selected stop criterion is fulfilled.

The Sequential Parameter Optimization Toolbox (SPOT) [BLP10] is a meta-modeling approach using linear regression to model the response surface. Kriging and tree-based regression [HTF09] are also supported to this end. The authors of SPOT see its main field of application in tuning the parameters of metaheuristics, e.g. Evolutionary Algorithms (EAs).

Konen et al. [KKF+11] build on SPOT and use it in tuning the parameters of a generic data mining process template. They compare the results attained by SPOT to those by three other optimization methods: Latin Hypercube Design (LHD) [MBC79] is considered as a baseline comparison. Since the parameters of the data mining template are numeric, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [Kel99] algorithm[5] is applied as a representative of local non-evolutionary numerical optimization. The more recent Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Han06] is used as the third competitor, representing non-local evolutionary numerical optimization. The benchmarks examined with the data mining template are the 2007 and 2010 editions of the Data Mining Cup [Pru15] and appAcid

[5] Among other optimization algorithms, BFGS is briefly summarized in Section 3.7.1.

[WGS+10]. Results are that SPOT ranks best in tuning the template, followed by the best LHD results, while average LHD performance is far worse. CMA-ES ranks third, and the local BFGS strategy is the worst competitor. As a summary, Konen et al. demonstrate the success of optimization in automatically tuning the parameters of a generic template for data mining. This relates to the classification subtask in *SynOpSis*, which is realized similarly.

Hutter, Hoos, and Leyton-Brown [HHL11] propose a meta-modeling approach that uses a Random Forest regression [Bre01] to model the response surface. Random Forest regression yields three benefits: Firstly, categorical parameters are supported in addition to numerical parameters. Secondly it can integrate multiple dataset instances into one meta-model by incorporating instance-related features to the forest. Thirdly, Random Forest regression provides not only an estimate of the objective for a given parameter set (mean value over the regression tree outputs), but also the uncertainty of this estimate (derived from the variance over the regression tree outputs). This information is exploited in the search for promising candidate parameter sets: Candidates are found by maximizing Expected Improvement (EI) [JSW98] over the incumbent (i.e. the current best) parameter set. EI is high for regions with good objective values and for regions with high uncertainty. It hence trades off searching in known good versus unknown regions (exploitation versus exploration), while regions known to be bad are not examined.

Bergstra et al. [BBB+11] apply meta-modeling in optimizing image classifiers based on Deep Belief Networks (DBNs) [LEC+07; HOT06]. The approaches proposed in the paper can handle conditional parameters, i.e. parameters which are only relevant if some other parameters take certain values. For example, if a boolean parameter toggles execution of a sub-algorithm, the other parameters of this sub-algorithm are only relevant if its execution is enabled. One prediction, based on 200 samples in the meta-model, is reported to take 10 s, respectively 150 s, depending on the type of underlying response surface model. For *PAMONO*, one actual function evaluation takes on the order of 20 s to 50 s for typical input sizes, exploiting the parallel processing capabilities of the Graphics Processing Unit (GPU) [LST+13a]. Hence, predicted function values taking 10 s to 150 s provide no gain. Furthermore, due to sparsity of good points in parameter space, considerably more than 200 samples are required to obtain a good response surface model. This further increases the time needed for predictions.

For further reading, Bischl et al. [BMT+12] provide a survey of meta-modeling methods applied within evolutionary optimization. Besides that, their work presents a method for assessing the accuracy of a meta-model. Furthermore, meta-model selection and tuning of the parameters of a meta-model are discussed. This wraps parameter optimization in an additional meta layer with its own parameters. While this kind of meta layer is not considered in *SynOpSis*, model selection and parameter tuning are considered, even though this happens one layer below. The section named “Common Pitfalls, Recommendations, and Statistical Properties” is a recommended reading for every layer of statistical modeling and machine learning in designing a data analysis process.

Evolutionary methods for parameter tuning combine some of the characteristics of model-free and meta-modeling methods. Nevertheless, they are *not* a hybridization of those approaches. Evolutionary methods share the following property with meta-modeling approaches: Knowledge from previous evaluations of the objective enters into the search for new promising candidate parameter sets. However, this is realized in a model-free

fashion, i.e. no explicit model of the response surface is constructed. Instead, evolutionary methods simulate the mechanisms driving natural evolution: Parameter sets assume the role of *individuals* that compete against each other in terms of objective values. The winners in this competitive *selection* process mate and exchange parameters. This is where knowledge from previous evaluations of the objective enters: Parameter sets that are known to perform well are recombined to form (hopefully) better parameter sets via *crossover*. Furthermore, new individuals can be generated by randomized *mutation* of parameters, i.e. a parameter is assigned a new value, randomly drawn from a defined distribution. More information on evolutionary methods will be given in Section 3.7. In-depth details can be found in the literature [Deb01; Luk13].

Ansótegui, Sellmann, and Tierney [AST09] address tuning of solvers for the Propositional Satisfiability Problem (SAT) by means of a Gender-Based Genetic Algorithm (GGA)[6]. It divides the population of all individuals into two genders, of which only one gender is subjected to selection pressure: Only the top x percent of individuals of that so-called competitive gender are allowed to mate. In the so-called noncompetitive gender, all individuals are allowed to mate. This is done with a randomly assigned partner from the top x percent in the competitive gender. Mating is only carried out between different genders, and the offspring is randomly assigned one gender. The paper empirically shows that making the genetic algorithm gender-based improves solution quality and reduces the number of necessary objective evaluations in minimizing three analytical toy functions, as well as in tuning several SAT solvers.

In their two-part paper, Mukhopadhyay et al. [MMB+14a; MMB+14b] specifically address data mining algorithms as a field of application for evolutionary methods. Data mining algorithms often pursue conflicting goals that can be formalized as opposing objectives for optimization, which is why the paper focuses exclusively on Multi-Objective Evolutionary Algorithms (MOEAs). While the scope of application of MOEAs in that paper exceeds[7] using them for *tuning existing* algorithms, successful approaches in tuning the parameters of classification algorithms are surveyed. These include applications with severe class imbalance[8], which benefit particularly from multi-objective optimization because additional objectives penalizing misclassification of minority class examples can be considered.

Image Processing Pipelines

While the previous paragraphs primarily addressed work related to the *Optimization* stage in Figure 3.2, which can be regarded as an outer loop governing offline computation in *SynOpSis*, the following paragraphs will cover work related to the inner components of that loop, i.e. to the algorithms to be tuned. As according to the previous sections, the main algorithmic components requiring tuning are a pattern detector, finding candidate locations for objects of

[6] Genetic algorithms are the most common kind of evolutionary methods. Variables to be optimized are represented as genes in a simulated process of evolution [Deb01].

[7] The literature surveyed by Mukhopadhyay et al. covers using MOEAs for feature selection and classification (part 1 of the paper [MMB+14a]) as well as for clustering, association rule mining and further data mining problems (part 2 of the paper [MMB+14b]).

[8] A classification problem is considered imbalanced, if one of the classes is severely outnumbered by another class. This not only causes problems for most learning algorithms but also deteriorates the usefulness of frequently employed performance metrics such as Accuracy. For further information on these issues, cf. Section 6.3, as well as [HG09] and the references therein.

interest in the input images, and a pattern classifier, sorting out spurious detector responses. One or both of these components frequently occur as parts of image processing pipelines. Many image processing pipelines fulfilling tasks similar to *PAMONO* data analysis are found in the following fields:

1. **Cell detection** is the task of finding image pixels that belong to biological cells. This task has been tackled for a variety of microscopic modalities and data dimensions [HBR+08; HBR+12; PKC09; YBC+10; ALN+12; WHS+12].
2. **Particle detection on the micrometer- and nanometer scales** is a more general task, where the objects of interest are not limited to biological cells. However, many approaches in this field aim at detection of constituent parts of such cells, i.e. even smaller entities are searched. Due to the resolution limit [Abb73] as discussed in Section 2.1, this often boils down to finding blob-like patterns or other approximations of the Point Spread Function (PSF) [NW10] of the optical system [TRS+02; Oli02; ZFS+07; SLN+09; JZK+07].
3. **Other image processing applications** related to *SynOpSis* tackle detection of objects on larger scales, e.g. tumors [MSB+13] or galaxies and stars [JT81; MSB95].

Hence the following discussion is divided into these three parts. The uniting characteristic of the data is that objects of interest often cover only a few pixels and exhibit a very low Signal-to-Noise Ratio (SNR).

Cell detection makes up a large part of the related work concerning image processing pipelines. It relates to the field of cell biology, and like *PAMONO* data analysis, its applications demand for *automatic* processing of microscopy images because manual evaluation of acquired data becomes an increasingly severe bottleneck, impeding large-scale experiments. Different modalities of cell microscopy and differing goals of analysis spawned a very diverse landscape of methods for solving the individual analysis problems [WHS+12]. A selection of these methods will be presented now, with a focus on how detection and, if given, classification are realized by each one.

Han et al. [HBR+08] divide the task of distinguishing multiple classes of cells in histological slices into detection via watershed segmentation [VS91] and classification via machine learning: Several weak Haar-based classifiers [VJ01] are cascaded via boosting [HTF09] to build one strong classifier. The individual classifiers are supervised, and a large amount of training data (about 500 to 1300 examples per class) must be manually segmented in order to obtain a good boosted classifier. The method shares with *SynOpSis* the use of separate images for training, validation and testing, to be described in the context of model selection and performance estimation, cf. Section 3.9. In contrast to *SynOpSis*, the images used for training are not synthetic and must hence be manually annotated with ground truth information.

Later [HBR+12], Han et al. moved from the cascaded Haar ensemble to Support Vector Machine (SVM) classification [MMR+01] using Laplace edge features [RW95]. The approach is reported to generalize well over cell type, cell scale and histological staining technique. Detection is realized by the classifier: A sliding window is used to cut out candidate regions from the input images and the SVM classifier predicts, whether the region corresponds to a cell or not. The SVM is optimized by conducting a grid search over its misclassification cost C and the γ parameter in the employed Radial Basis Function (RBF) kernel. The paper mourns the lack of a systematic approach to configure algorithmic parameters beyond those of

the SVM, hence connecting it to *SynOpSis* which is proposed as such a systematic approach to fill that gap.

Pan, Kanade, and Chen [PKC09] present a method that shares with *SynOpSis* the division into detection and classification. Detection is realized in three steps: Firstly, local fluctuation energy of the image is computed as summed Laplacian filter responses at different scales and orientations, and candidate regions are obtained where this energy exceeds a certain threshold determined from training data. Secondly, the original input image is masked with the regions from the first step, and among the remaining pixels, local minima[9] are determined as candidate cell points. Thirdly, point locations are refined in a procedure similar to mean-shift [CM02], to make them coincident with perceived cell centers. After detection, the points are classified on the basis of, amongst others, Histogram of Oriented Gradients (HOG) features [DT05]. An SVM classifier [MMR+01] with RBF kernel is used to distinguish points located within cells from those located on image background. The overall approach is claimed to generalize well over different datasets (e.g. different cell types or modalities of microscopy) without the need for tuning algorithmic parameters. However, it requires a complete manual segmentation as training data for each type of dataset.

Yin et al. [YBC+10] classify cell images acquired via phase contrast microscopy and differential interference contrast microscopy on the pixel level. Each pixel is classified by a bag of local Bayesian classifiers, and the final decision is computed by a mixture-of-experts model, that aggregates classifier votes with weighting functions depending on the inputs to be classified, hence allowing to define which classifier is the more dominant expert in which part of the input space [JJN+91]. Making classifier weights in ensemble aggregation depend on the input data to be classified contrasts with boosting approaches [FS97; JZK+07; HBR+08] where the weight of a classifier depends on its performance on the training data. The local expert classifiers in the approach by Yin et al. are computed as follows: For each feature (e.g. intensity, gradient on multiple scales), a clustering of local feature histograms from random positions in the image is computed. Then one Bayesian expert classifier is formed from each found cluster center. Priors and likelihoods can be inferred from the training data, and the posteriors can be formed using Bayes' rule. After training, classification takes 50 s for a 1.4 Mpx image, ruling out this approach for real-time application in the *PAMONO* scenario.

Arteta et al. [ALN+12] detect cell region candidates via Maximally Stable Extremal Regions (MSER) [MCU+04]. Detected regions are classified by a structured SVM [THJ+04] rewarding a one-to-one correspondence of detected regions to the dot annotation supplied by the user as training data. The SVM works on 92-dimensional feature vectors containing histograms of intensity and intensity difference between the border of a region and its surroundings, a shape descriptor of the region and its area. While this method is demonstrated to generalize over three modalities of microscopy, it takes 30 seconds for processing a 400 px × 400 px image on an Intel® Core™ i7 CPU and is hence not applicable if real-time data analysis is desired.

Wienert et al. [WHS+12] aim at minimizing the amount of a priori information necessary for an analysis, for the sake of reducing bias towards cells with "regular" morphology. The main benefit is that cells with more irregular morphological features, e.g. malignant cells in tumors, can be detected more reliably. The method employs closed-contour-tracing and subsequent classification of contour-related features. In order to classify irregularly

[9] The cells considered in [PKC09] are darker than the background.

shaped cells correctly, the utilized features are invariant to the morphological variations in question. Nevertheless, the final processing step, separating cell nuclei from background detector responses, relies on features caused by the Haematoxylin staining of the cell nuclei, thus restricting generality of the method. Processing time for a $400\,\text{px} \times 400\,\text{px}$ image on a "standard PC" is reported to be 0.39 s, which is ≈ 77 times faster than [ALN+12] but still too slow for processing *PAMONO* data in real-time.

Particle detection on the micrometer- and nanometer scales is another field that provides work related to *SynOpSis*. Most papers in this field are closely related to cell detection because *components* of cells in differing modalities of microscopy are the objects of interest. The difference, however, to detection of whole cells, is that in many cases, features like textures and contours can not be successfully applied, due to the very low size of the particles of interest. This small size in combination with the resolution limit [Abb73] discussed in Section 2.1, turns most of these detection tasks into finding blob-like patterns or other approximations of the Point Spread Function (PSF) [NW10] of the optical system. Hence, in the following discussion, the particles to be detected will be referred to simply as **blobs**, i.e. by their appearance in the images. A further common characteristic in this context is the very low Signal-to-Noise Ratio (SNR) in the input [JZK+07].

Thomann et al. [TRS+02] tackle a problem demonstrating many of these aspects: The task is fluorescent tag detection in 3-D super-resolution microscopy. The blobs to be found are very small (e.g. $7\,\text{px} \times 7\,\text{px}$) compared to the image resolution and their appearance is dominated by the PSF of the optical system, approximated by a Gaussian, while the SNR is low. Detection of candidate blobs is carried out by thresholding a feature map integrating intensity and curvature information, cf. also Section 6.2. Super-resolution localization is obtained by fitting a Gaussian mixture model to the detected blobs, reflecting the prior knowledge that the signal consists of superimposed variants of the Gaussian-like PSF of the optical system.

Olivo-Marin [Oli02] also works on fluorescence images and follows the goal of finding small, bright blobs. His method uses the à trous wavelet transform [Mal99] of an input image, to represent it in an undecimated multiscale space. In that space, denoising and blob detection are conducted. For detection, the detail coefficient planes are multiplied, and local maxima are searched in the product. This search exploits the result that local maxima that are due to additive Gaussian white noise do not propagate across detail coefficient planes, while local maxima that are due to actual discontinuities in the image (e.g. edges or blobs) do [MZ92; JS97; JB99]. However, this method was reported to perform worst, by a large margin, on low SNRs images, in the 2009 blob detection survey by Smal et al. [SLN+09], which will be discussed later in this paragraph.

Zhang et al. [ZFS+07] propose a method that can also be used to detect blobs in fluorescence microscopy, but their focus lies on a denoising method to facilitate this detection. Their denoising strategy explicitly addresses the Mixed-Poisson-Gaussian (MPG) nature of fluorescence microscopy, which results from a mixture of the Poisson noise (shot noise) due to photon count statistics [BB00] and the Gaussian noise due to sensor readout [FM06]. The MPG nature of an input image is alleviated by transforming it to a near Gaussian process using a multiscale Variance Stabilizing Transform (VST), which extends the Generalized Anscombe Transform (GAT) [SMB98] by a post-processing with undecimated isotropic wavelets [SMB98]. The "Gaussianized" coefficients are then denoised by increasing their

sparsity using an iterative optimization scheme. Smal et al. [SLN+09] report competitive results for the detector part of this method which is based on zeroing both, approximation coefficients and the coefficients deemed insignificant by the denoising step. Computation times are not reported, but the iterative optimization scheme, involving steepest descent, as well as forward and backward wavelet transform in each iteration, hints at unsuitability for real-time applications like *PAMONO*.

The already mentioned work by Smal et al. [SLN+09] gives a survey on low SNR blob detection involving, among others, the three previously discussed unsupervised methods [TRS+02; Oli02; ZFS+07]. Beyond those, the survey also examines approaches involving supervised machine learning and reveals that these are superior to unsupervised methods if the SNR is very low. This advantage, however, decreases with increasing SNR. The examined machine learning methods are the work by Jiang et al. [JZK+07] and a variant of that, using Linear Discriminant Analysis (LDA) [HTF09] as the classifier.

In the original version, Jiang et al. [JZK+07] use Adaptive Boosting (AdaBoost) [FS97] as the classifier. This is done in the context of detecting clathrin-coated pits within cells. Just like in the previously discussed cell detection algorithm by Han et al. [HBR+08], the employed features are based on the seminal work by Viola and Jones, originally developed for face detection [VJ01]. Jiang et al. argue that intensity alone can not capture the information necessary to successfully classify the very small blobs to be detected, and they see the advantage of Haar features in their ability to simultaneously capture information about intensity, shape and scale of the underlying objects. The good performance of this method, especially in comparison to unsupervised methods in the presence of low SNR, as reported in [SLN+09], makes supervised machine learning an interesting approach for *PAMONO* data analysis that will be investigated in Chapter 6.

Other image processing applications that are related to *SynOpSis* deal with objects of interest that reside on larger scales. A selection of methods applied in these contexts will be given now.

The first application is tumor detection in Automated Whole Breast Ultrasound (ABUS) images. To this end, Moon et al. [MSB+13] propose a multiscale blob detection based on analyzing the Hessian of the ABUS images. Hence, similarly to [TRS+02], local curvature of intensity is used as a measure of "blobness". Before Hessian approximation, a speckle noise reduction is carried out to address artifacts inherent to the ABUS method. Like *SynOpSis*, this approach firstly aims at not missing any pattern that is a candidate for an object of interest (tumor), followed by classifying the candidates into true and false positive detector responses (TPs and FPs). In candidate detection, all tumors are detected (Recall = 1) at the cost of many[10] FPs. Then a threshold on a logistic regression [HTF09] estimate of tumor likelihood is applied to classify the candidates, trading off Recall for fewer FPs[11] in a ten-fold cross-validation. The employed features in this classification were derived from the Hessian-based blobness measure used in detection, complemented with features of local intensity distribution and morphology. Computation time is reported to be 13 min per image and thus too slow for real-time analysis. Calculation of the blobness feature can be accelerated by exploiting the parallel processing capabilities of GPUs.

[10] An average of 1044.89 detector responses versus a maximum of 3 tumors per dataset is reported.

[11] The number of FPs varies, depending on the achieved value of Recall, but it is small enough for manual post-classification in the context of Computer-Aided Detection (CADe) by medical experts.

While finding tumors already is a task residing on a larger scale than identifying cells or their constituents, the superordinate problem of automatically detecting objects of interest with a certain appearance in digital images not only arises in medical applications, but also in very different scientific disciplines operating on yet considerably larger scales, e.g. astronomy. Despite the large difference in scale between the previously discussed fields of application and astronomy, the problem to be solved still fits into the abstract task description from Section 3.1: Objects with defined appearances, that often cover only a few image pixels, are to be detected and classified, while their intensity is rather small in comparison to background noise (low SNR). Most of these aspects can immediately be seen when looking at the title of the work by Jarvis and Tyson [JT81]: Faint Object Classification and Analysis System (FOCAS). The input of FOCAS are digitized astronomical plates, and the task to be solved is computing histograms that count the number of stellar objects for ranges of intensity magnitudes. This task involves a classification subtask because stars have to be distinguished from galaxies of the same intensity magnitude. Furthermore, these two classes of interest have to be separated from spurious noise detections due to dust, lint and adverse properties of plate emulsion. Separating these three classes becomes more difficult for decreasing magnitudes of the galaxies and stars. FOCAS, like most previously discussed methods for cell and particle detection, shares with *SynOpSis* the division of the task into detection and classification. Detection is carried out by thresholding a filtered version of the original input image. Classification employs as features the moments of intensity and shape [Hu62] of the detected object candidates, along with template matching-based and other application-specific features. An interactive training procedure of a preliminary classifier is carried out by visually inspecting the scatter plot matrix of the seven-dimensional feature space, and manually picking separating curves for each combination of two distinct features, which are then assembled to form a separating hypersurface in the original feature space. Then a clustering is carried out in feature space on the unclassified detected points, and the preliminary classifier serves to determine which cluster centers correspond to which class. Hyperellipsoids are then fitted around the clusters, the union of which defines the final decision hypersurface. This process has to be done separately for each astronomical plate. While being outdated and rather empirical, this method very well illustrates the need for automation and systematization of image processing and object classification processes, as covered by *SynOpSis*.

Murtagh, Starck, and Bijaoui [MSB95] present a more general framework for astronomical image processing that uses undecimated wavelets to solve a variety of tasks in a multiresolution setting. Among these tasks are image denoising, restoration, compression and object detection, all of which use the results of an à trous wavelet transform [Mal99] of the input image. The denoising part can be regarded as a methodological ancestor of [ZFS+07] which was already presented in the context of the micrometer- and nanometer scales: A VST is used to "Gaussianize" the MPG process provided by the image sensor, and subsequently wavelet coefficients that are deemed noise-related are zeroed in an iterative denoising scheme. The detection process, on the other hand, can be regarded as a methodological ancestor of [Oli02] because it examines the multiresolution representation of an input image in a scale-by-scale manner: Each scale deeming a pixel significant votes for the pixel containing an object of interest. Scale-weighted summation of the per-pixel votes, followed by thresholding, yields a binary mask of candidate object pixels. This mask is then post-processed by applying morphological opening [GW07] two times to remove small ridges. By being closely related to

two works already presented in the context of detecting objects on the micro- and nanometer scales, the astronomical image processing framework by Murtagh, Starck, and Bijaoui illustrates the success of methods in denoising and detection across application boundaries and, more importantly, across a large difference in scale of the objects of interest.

SynOpSis and PAMONO in the Context of This Related Work

Putting *SynOpSis* and *PAMONO* into the context of the presented related work can be divided into the same two aspects as the presentation in this section. *SynOpSis* as a method belongs into the context of **Automatic Tuning of Algorithmic Parameters**, while *PAMONO* as the application case belongs to the context of **Image Processing Pipelines**, containing the algorithms, the parameters of which are tuned by *SynOpSis*.

This thesis applies parameter optimization to an image processing pipeline for *PAMONO* data analysis, used as an example. In the related work presented in this section, parameter optimization was demonstrated to be successful in application fields like optimizing meta-heuristics, solvers for NP-complete problems like SAT, and for data mining. Image processing pipelines frequently exhibit numerous parameters that heavily influence the quality of the attained processing results. A lack of automatic and systematic tuning procedures for these parameters was identified, the resolution of which was considered a desirable goal [HBR+12]. The contributions of this thesis can be divided into two central aspects: Firstly, the thesis combines automatic parameter optimization with a parametric image processing pipeline and demonstrates the success of applying the former to configure the latter. Secondly, it devises that image processing pipeline, which is the first analysis process to successfully analyze *PAMONO* sensor data with object sizes down to 100 nm. By doing so, the thesis alleviates two major bottlenecks in the practical application of the *PAMONO* technique: The first bottleneck is manual analysis of several thousands of images per measurement, impeding large-scale studies and introducing subjectivity. This bottleneck is targeted by the automatic image processing pipeline, which gives rise to the second bottleneck: The pipeline has many algorithmic parameters enabling its adaptation to the variable physical parameters of the sensor prototype. Manually finding suitable values for the algorithmic parameters is still subjective, and due to the large parameter space it can become even more tedious than manual data analysis. *SynOpSis* targets this second bottleneck by automatic parameter optimization. With that optimization it systematically integrates data synthesis, pattern detection, pattern classification, model selection and performance estimation in a unified framework. Regarded abstractly, this framework reflects the commonalities of the class of tasks characterized in Section 3.1. Adapting *SynOpSis* to another such task (e.g. a different imaging modality, input dimension or target object class) means replacing the individual modules for synthesis, pattern detection and classification, while the framework can be reused.

The remainder of Chapter 3 describes the exchangeable modules of *SynOpSis* in an abstract fashion (Sections 3.4 to 3.6) and gives details on how parameter optimization, model selection and performance estimation can be conducted, and how the results of these procedures are applied in analyzing the real sensor input data (Sections 3.7 to 3.10). Chapters 4 to 6 then develop the concrete realizations of the abstract modules for synthesis, pattern detection and classification, targeting *PAMONO* data analysis. While the related work presented in *this* section primarily focused on *SynOpSis* as whole and on a broader context of kindred image

processing pipelines, *those* chapters provide more specific related work, where the peculiar subproblems of *PAMONO* data analysis are tackled.

3.4 Synthesis Stage

Synthesis is the first stage to be executed in *SynOpSis*, cf. left side of Figure 3.2. It uses a signal model to generate a synthetic dataset of images for which ground truth pattern locations and classification are known. Defining a signal model is application-specific: A concrete example for *PAMONO* can be found in Chapter 4, while in the following, the properties required for such a model, and the role of synthesis in *SynOpSis* are depicted abstractly.

3.4.1 Signal Model

The purpose of the signal model in *SynOpSis* is generating synthetic images for which ground truth pattern locations and classification are known. This ground truth is used to define automatically evaluable objective functions measuring the quality of data analysis results. Having a method to assess analysis quality is equivalent to having a method to assess the quality of the algorithmic parameters employed in the analysis. This in turn enables automatic determination of suitable algorithmic parameters by optimizing the objective functions with respect to them.

A synthetic signal can, by construction, easily be annotated with ground truth information because the target patterns[12] are *created* by the model. The key requirement for the signal model is that it must mimic the real sensor data to be analyzed in such a way that processing parameters that work well on the synthetic images also work well on the real data, and furthermore, that a classifying model learned from the synthetic images, also classifies the real patterns well. To indicate this fact, the signal model in Figure 3.2 receives real images recorded by the sensor as input: It must as accurately as possible mimic the signal properties of the real sensor data to be analyzed. This goal may e.g. be reached by a physical simulation [MMB+05; WSP+10] that is based on signal properties found in the real data, or by data-driven synthesis [Lea06; SSK+13; SLW+14] creating synthetic data from observed real data.

When the physical parameters of the sensor are modified during different experiments (e.g. in sensor prototype experimentation and development) and if these modifications change the properties of the recorded real signal, these changed properties must be captured by the model in order to be adequately reflected within the signal computed in the *Synthesis* stage. The signal properties (e.g. noise level, irradiance, focus) of the synthetic signal must be as close as possible to the real sensor data to be analyzed. This is due to the role of the synthetic signal in *SynOpSis*: Optimized parameters for pattern detection, pattern classification and the predictive model classifying the patterns are derived from it. The better the signal model mimics the real data, the better these results transfer to the real data.

Applying the signal model forwardly to generate a synthetic, ground truth-annotated dataset is its primary purpose in the context of *SynOpSis*. If the employed signal model has the property of *explaining* image formation, it may serve a further purpose when the backward direction of image formation is regarded: The inverse problem of separating the

[12] In case multiple classes of target patterns are to be distinguished, all these classes must be represented in the synthetic data.

individual components contributing to the overall image can be facilitated by considering the signal model. For example a separation of the desired signal from noise and artifact signals may be desired. In such contexts, the signal model may regularize the inversion of the image formation, as e.g. in constrained least squares filtering [DK77]. More abstractly speaking, the signal model may also be used to guide image processing.

3.4.2 Synthetic Ground Truth Patterns and Classification

Besides synthetic images, a synthetic dataset must contain ground truth about all information to be provided by the pattern detector and classifier because that information is needed for comparison with detector and classifier results within the *Optimization* stage in Figure 3.2.

For the detector, ground truth locations of all target patterns in the synthetic images must be known, whereas the locations of non-target patterns need not be specified, but are given implicitly: Any pattern found by the detector that does not correspond to a ground truth target pattern must be a non-target pattern, i.e. a spurious detector response. Therefore, all patterns in the ground truth are target patterns. How the ground truth pattern locations are represented[13] and how detector results are matched[14] for determining correspondences to the ground truth is application-specific, cf. Section 5.8 for a concrete example.

For the classifier, ground truth class labels of the *detected* patterns are required for assessing the quality of classification results. The class labels can be transferred from the ground truth patterns to the detected patterns via correspondence: Any pattern in the ground truth is a target pattern, and any detected pattern that matches a ground truth pattern is hence labeled a target pattern[15]. Any detected pattern not corresponding to a ground truth pattern is labeled a non-target pattern. Besides being used in measuring classification quality, the ground truth-labeled *detected* patterns are furthermore employed in supervised learning of a predictive model for classifying detected patterns, as discussed abstractly in Section 3.6 and concretely in Chapter 6.

3.5 Pattern Detector

A further essential component in the *SynOpSis* approach is the pattern detector. The patterns to be detected in the input images are regions that are candidates for containing objects of interest. The pattern detector appears twice in Figure 3.2: Firstly, it is used in the *Optimization* stage, during the process of finding optimized parameters with respect to synthetic ground truth-annotated data. Secondly, it is used with the optimized parameters in the *Application* stage. Here it produces pattern detection results for the real sensor images that are input to *SynOpSis*.

[13] Possible representations encompass points, points with radii, a segmentation mask, or polygons. Further representations are conceivable, particularly in application scenarios beyond *PAMONO*.

[14] Several definitions of a 'match' between a ground truth pattern and a detected pattern are conceivable, depending on the representation of both and on application requirements. Possible definitions may e.g. measure point distance, point-to-polygon-incidence, polygon-centroid-to-polygon distance or polygon overlap area. A concrete example is given in Section 5.8.

[15] If multiple classes of target patterns have to be distinguished from the non-target patterns and from each other, the ground truth target patterns need to be labeled accordingly. The application-specific matching procedure can be used to transfer class labels from the ground truth patterns to the detected patterns to provide a comparison between ground truth labels and the labels predicted by the classifier. All classes of target patterns to be distinguished must be represented in the synthetic images.

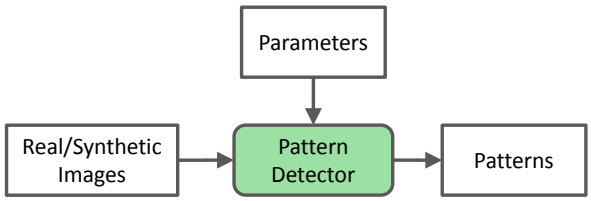


Figure 3.3: Pattern Detector Input and Output. The pattern detector module of *SynOpSis* receives as input data either real images recorded by the sensor or synthetic images generated by a signal model mimicking the sensor. In addition to the data, its second input is a parameter set configuring the algorithms constituting the detection process. The output consists of the patterns that were detected in the input data, using the input parameters.

Depending on the type of images supplied, and the objects of interest to be found, the concrete implementation of the pattern detector may vary considerably. In the abstract depiction of *SynOpSis*, given in this chapter, the pattern detector is hence regarded as an abstract module that needs to be implemented in a detection-task-specific way, as e.g. presented in Chapter 5. Only its input and output interface is specified here (Section 3.5.1), and a choice of possible objectives that can be used in optimizing its application-specific parameters is presented (Section 3.5.2).

3.5.1 Input and Output

Figure 3.3 repeats the pattern detector from Figure 3.2 and its in- and outputs for convenience. As can be seen from here, as well as from both instances of the pattern detector in Figure 3.2, its input consists of the images to be analyzed and the algorithmic parameters configuring it. Consequently, the pattern detector is assumed to contain at least one parametric algorithm, otherwise the *Optimization* stage would be trivial. The number and types of algorithmic parameters of the pattern detector are very application-specific. Since in *SynOpSis* the task is finding target patterns in images, examples of possible parameters are those arising in image processing, like kernel sizes/shapes for filtering and searching, the choice and parameterization of denoising methods, or detection thresholds. A set of parameters encompassing values for *all* free parameters of the pattern detector is denoted a detector **parameter set**. Such a parameter set yields one certain instance/configuration of the pattern detector.

Parameter sets must be clearly distinguished from the data that is input to the pattern detector. This data consists of the images in which the patterns are to be detected. The domain of the input images must match that handled by the pattern detector. For example in *PAMONO*, the images are from the spatiotemporal domain, i.e. time series of consecutive images are analyzed.

The output of the pattern detector consists of a set of patterns localizing candidate regions for objects of interest in the images. These regions can be represented in a multitude of ways, e.g. points with radii, a segmentation mask, or polygons. The concrete choice of representation depends on application requirements.

3.5.2 Objectives

Suitable and automatically evaluable objective functions are the key component in using optimization to automatically configure the parameters of the pattern detector. Such objective

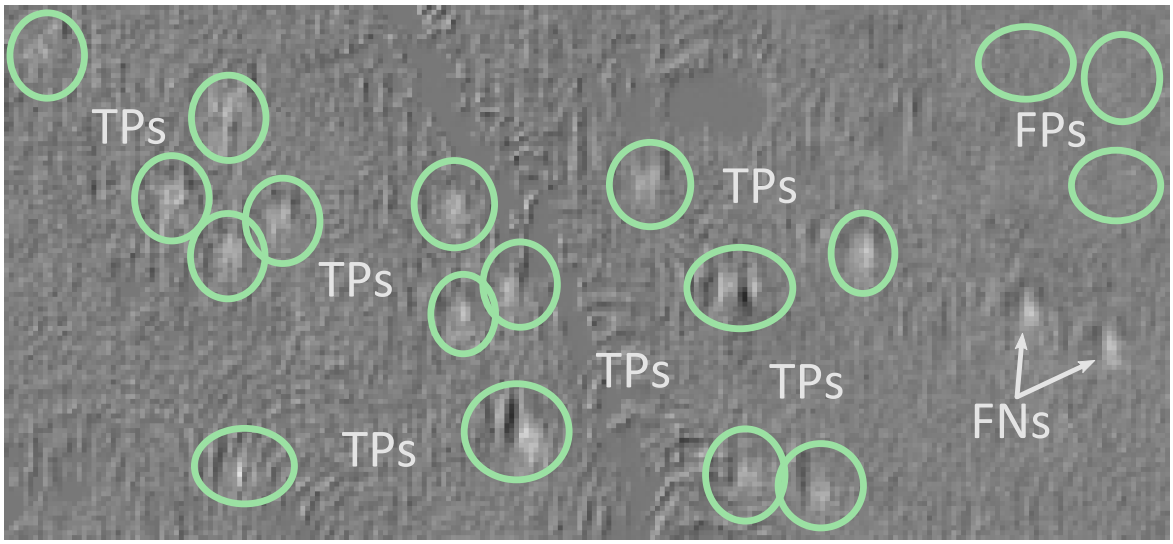


Figure 3.4: Confusion Matrix of a Detection Task – Illustration. An exemplary *PAMONO* image is shown. Patterns found by the detector are indicated by green ellipses. In the top right corner, three non-target pattern detections can be seen, counting as False Positive (FP) detector responses in the confusion matrix, cf. Table 3.1. Below that, two target patterns were missed (no detector responses), resulting in False Negative (FN) entries in the confusion matrix. All other patterns are target patterns that were correctly detected and hence count as True Positives (TPs). The case of a True Negative (TN) does not arise in detection because a detector only yields positive responses.

functions will be presented here. They are used in the *Optimization* stage in Figure 3.2. Before these objectives can be defined, the adopted conception of the detection task must be stated. In the following, the task of detecting patterns in input images is regarded as the task of identifying salient image regions that are candidates for containing objects of interest. The detector can only yield positive responses, i.e. detections, cf. green ellipses in Figure 3.4. Anything not causing a detector response is not represented in the output of the detector. To determine whether these responses are related to target patterns and hence correct, they have to be matched to the ground truth, which by Section 3.4.2, contains solely target patterns. If a ground truth match can be found for a detected pattern, this is a True Positive (TP) detector response cf. the exemplary *PAMONO* image in Figure 3.4 (all ellipses marked as TPs contain a faint blob which is the target in this case because these blobs are indicative of nano-objects attaching to the surface of the *PAMONO* sensor). For the current discussion, it is assumed that the case of multiple detected patterns matching a single ground truth pattern does not occur. This case is treated extensively later in this section. If no ground truth match can found for a detected pattern, this is a False Positive (FP) detector response (three ellipses in the top right of the figure are marked as FPs because they do not contain a blob). Furthermore, there can be ground truth patterns for which no corresponding detected pattern exists. Such a situation arises if a ground truth target pattern is *missed* by the detector and is thus denoted as a False Negative (FN) (on the right side of the figure there are two blobs without an ellipse, hence marked as FNs). Note that this negative case does not conflict with the statement that the detector can only yield positive responses because an FN is not a detector response and can thus only be determined owing to the ground truth. The case of True Negatives (TNs) does not exist in detection tasks [WHS+12; SLN+09] because such

Table 3.1: Confusion Matrix of a Detection Task. The confusion matrix of a detection task tabulates the total numbers of TP, FP, FN and TN detection outcomes. A True Positive (TP) is a detector response that matches with a ground truth pattern, while a False Positive (FP) is a response for which no matching ground truth pattern exists. A False Negative (FN) is a pattern in the ground truth for which no detector response exists, and hence an object of interest that has been missed by the detector. A True Negative (TN) is not defined in this context because it corresponds to something that has neither been detected, nor marked in the ground truth, and hence can not be counted, thus TN = 0 [WHS+12; SLN+09]. TPs are the desired responses, while FPs and FNs are undesired. FPs can be remedied by subsequent classification, while FNs (misses) can not be recovered.

		Ground Truth	
		Target	Non-Target
Detector	Response	TP	FP
	No Response	FN	TN = 0

an event would neither cause a detector response, nor is it represented in the ground truth, which by definition consists solely of target pattern locations. TNs do not relate to discrete events in the analyzed data, so the number of TNs is defined to be zero.

Basics: Confusion Matrix

In order to state these presented cases more formally, and as the basis for defining objective functions, the concept of a **confusion matrix** is employed, which is a popular tool in assessing the quality of machine learning algorithms for classification [Pow11]. Generally, a confusion matrix tabulates outcomes of classification algorithms, which are often called **predictions** and which assign predicted class labels to the examples to be classified. In the confusion matrix these predictions are contrasted with the actual ground truth class labels that a perfect classifier would have predicted. The confusion matrix $\mathbf{E}$ is a $C \times C$ matrix, where $C \in \mathbb{N}_{\geq 2}$ is the total number of classes to which the examples to be classified can belong. Each matrix entry $e_{i,j} \in \mathbb{N}_{\geq 0}, i, j \in \{1, \ldots, C\}$, equals the number of examples that were predicted to belong to the class named c_i, while the ground truth assigns them to the class named c_j. Entries $e_{i,j}$ with $i = j$ count correct predictions, while all other entries count erroneous predictions. Hence, the confusion matrix can be used to asses classifier quality.

Table 3.1 shows the peculiar confusion matrix of the detection task discussed here. It is a 2×2 matrix, due to $C = 2$ classes: target and non-target. The peculiarity of the confusion matrix arising in detection is that the entry for TNs is always zero; otherwise it has the same properties as a confusion matrix from two-class, i.e. binary, classification. Therefore any measure of binary classification quality that does not divide by TN can be computed to measure detection quality, cf. Appendix A for examples of such measures.

Automatic optimization of detector parameters in *SynOpSis* employs such measures as objectives and hence requires automatic computation of the confusion matrix. As pointed out, this can be achieved by implementing a matching procedure between detected and ground truth patterns that suits the application task at hand, cf. Section 5.8 for an example. Given such a matching procedure, the TPs, FPs and FNs can simply be counted, setting up the confusion matrix. Therefore, the components required for automatically evaluating

the quality of detector parameter sets are present, enabling utilization of these measures as objective functions during parameter optimization.

Objective 1: Recall

For the sake of brevity, this section lists only the objective functions that were finally applied in optimizing *PAMONO* pattern detection. Depending on the application and its requirements, other objective functions may be advisable, hence Appendix A provides a wider selection of possible objectives, along with references for further reading. As stated in Section 3.1, the analysis task is divided into a detection and a classification part. The pattern detector follows the goal of not missing any target patterns, i.e. it strives to keep the FN entry in the confusion matrix as small as possible. The reason is that FNs do not result in detector responses and can thus not be remedied during the subsequent classification process. This goal of high sensitivity may be pursued at the possible cost of many spurious, i.e. non-target, detector responses: The FP entry may be large, because FPs can be eliminated by the subsequent pattern classifier.

As a consequence, choosing Recall as an objective function for parameter optimization is a natural choice [PKC09]. Recall [HG09] is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.1}$$

and yields the ratio of target patterns in the ground truth (denominator) that was found by the detector (numerator). Optimizing for maximized Recall keeps the number of uncorrectable misses small by penalizing FNs in the denominator. It may result in a large number of FPs because they are not penalized, thus only high Sensitivity[16] is rewarded. In order to avoid overly sensitive parameters that simply cover the entire image domain with rather random detector responses, the number of detector responses is allowed to be at most a times the number of ground truth target patterns. Any parameter set attaining more responses is discarded. For example in *PAMONO* $a = 5$ was chosen, cf. Section 5.8 and Section 7.3.1. Imposing a threshold on the maximum number of admissible detected patterns can be regarded as introducing a hard constraint on the minimum required value of Precision = $\frac{\text{TP}}{\text{TP+FP}}$. Besides avoiding overly sensitive parameters, this heuristic can save computational resources because the number of detected patterns can be determined at low computational cost, whereas actually computing Recall and Precision involves matching detection results to the ground truth. As the number of ground truth patterns is constant, matching cost increases linearly with the number of detections. Thus, the heuristic particularly avoids the more expensive matching computations involving many detected patterns. Doing so is particularly beneficial in the *Optimization* stage because objectives have to be evaluated many times, and parameter sets may be examined that generate large numbers of detected patterns.

Note that the account of Recall given here does not consider the case where the detector yields multiple responses to a single target pattern, which is a case that is not reflected in confusion matrices arising in classification, but that can occur in detection. This case is treated further below, where Recall is revisited. Before that, the second objective function is presented because revisiting Recall requires arguments and a definition given in the context of the second objective.

[16] Recall is synonymously denoted as Sensitivity.

Objective 2: M-Rate

Besides FPs, repeated detections of the same object of interest may pose an issue, as encountered e.g. by Han et al. [HBR+12] in the application case of detecting cell nuclei in 2-D images. In *PAMONO* data analysis, this issue is aggravated by the additional temporal dimension and the resulting fact that the same target patterns are visible in multiple subsequent images. Hence avoidance of repeated detections is used as an objective in *PAMONO*.

Stated in the terminology employed in this thesis, the case of a **repeated detection** is constituted by multiple detector responses matching a single ground truth pattern. This additional type of spurious detector response is particularly undesirable because it is the detection of an actual target pattern and hence a TP, adding to the TP entry in the confusion matrix. Whether a detector response is a repeated or a first detection can only be determined by matching it to the ground truth, followed by testing whether it matches to a ground truth pattern that already has a matching partner. The problem is that this information is not available in the absence of ground truth and hence in analyzing real data. A parameter set yielding many repeated detections is prone to overestimating the number of target patterns present in a dataset because each repeated detection is likely to be assigned to the target pattern class during classification. The reason for this is that a repeated detection is caused by an actual target pattern, and thus exhibits signal features like a desired first-time detection. So the classifier can not be expected to sort out repeated detections, but will rather classify them as target patterns. These are excess patterns which result in an overestimation of the number of target patterns in the images.

As a conclusion, this issue must be dealt with during optimization. One option to do so is by rewarding parameter sets that minimize the number of repeated detections. Multi-Detection-Rate (M-Rate) is proposed as a measure to quantify the ratio of repeated detections incurred by a parameter set. In *PAMONO* data analysis it is employed as a second objective function in optimizing the pattern detector. It is defined as

$$\text{M-Rate} = -1 + \frac{\text{TP}}{\widehat{\text{TP}}}, \tag{3.2}$$

where $\widehat{\text{TP}}$ is the number TP detector responses, excluding secondary and all further repeated detections. If there are no repeated detections $\widehat{\text{TP}} = \text{TP}$ and thus M-Rate = 0. Note that because $\widehat{\text{TP}} \leq \text{TP}$ it holds that M-Rate $\in \mathbb{R}_{\geq 0}$. Besides penalizing repeated detections in the optimization of the pattern detector, further heuristics can be applied to prevent them. Such heuristics can e.g. exploit proximity between first-time and repeated detections (cf. Section 5.6.3 for an example), at the cost of incurring a possible decrease in the capability to resolve closely neighboring target patterns. Another possible heuristic for this issue was proposed by Arteta et al. [ALN+12], where a structured Support Vector Machine (SVM) [THJ+04] is used to classify detected regions, and one-to-one correspondence of these regions to a ground truth dot annotation is rewarded by an additional term in the loss function of the structured SVM.

Objective 1 Revisited: Recall

With the issue of repeated detections discussed, and the number $\widehat{\text{TP}}$ of TPs cleansed from repeated detections defined, the first objective in optimizing the pattern detector is now

revisited. The reason is that defining Recall with respect to the number of TPs has its roots in the context of classification, where it is the accepted textbook definition. In the context of detection however, this definition has an adverse effect concerning repeated detections of the same target pattern: Consider two sets of detector parameters that differ only in the number of repeated detections they incur. Let TP_1 be the number of TPs found by the first parameter set and $\mathrm{TP}_2 = \mathrm{TP}_1 + D, D > 0$ the analogous number for the second set, incurring D more repeated detections than the first one. Recall_1 and Recall_2 as attained by the two parameter sets are calculated by substituting TP_1, respectively $\mathrm{TP}_1 + D$ for TP in Equation (3.1). Comparing the results yields that

$$\mathrm{Recall}_1 = \frac{\mathrm{TP}_1}{\mathrm{TP}_1 + \mathrm{FN}} < \frac{\mathrm{TP}_1 + D}{\mathrm{TP}_1 + D + \mathrm{FN}} = \mathrm{Recall}_2$$

for $D > 0 \wedge \mathrm{FN} > 0$. It can be seen that this definition of Recall prefers parameter sets with more repeated detections over those with fewer. For increasing D and constant $\mathrm{FN} > 0$, Recall approaches its maximum of 1.

However, it is desirable that Recall in a detection task depends only on the number of ground truth target patterns that were detected and does not reward repeated detections. Therefore it is more appropriate to define Recall with respect to the number $\widehat{\mathrm{TP}}$ of TPs cleansed from repeated detections. Doing so makes the thus defined version of Recall invariant to the number of repeated detections, giving the final definition of detector Recall:

$$\mathrm{Recall} = \frac{\widehat{\mathrm{TP}}}{\widehat{\mathrm{TP}} + \mathrm{FN}} \tag{3.3}$$

Note that by this definition, a parameter set that incurs more repeated detections than another, while everything else is unchanged, attains the same value for Recall and a higher value for M-Rate and is thus Pareto-dominated[17] by the other parameter set, i.e. it performs worse in at least one objective while not performing better in any other.

Any value of Recall that is computed with respect to the detector in the course of this thesis uses the definition in Equation (3.3). This includes the Recall values employed in detector optimization as well as in reporting results.

Note that letting measures of detection quality drive the search for algorithmic parameters means that the image processing algorithms within the detector adapt to the data in terms of the best detection-specific image enhancement, not in terms of image restoration.

3.6 Pattern Classifier

In the previous section, a pattern detector was presented along with reasons for using Recall as one objective function in optimizing that detector: A highly sensitive detection is required in order to prevent missing any target patterns in the data. This increases the risk of detecting many non-target patterns, e.g. due to noise or other unwanted artifacts in the input images. Stated as entries in the confusion matrix of detection from Table 3.1, a high number of FP detector responses is accepted for the sake of decreasing the number of FNs.

Now the pattern classifier serves to separate the detector responses into target and non-target patterns, making up for the adverse consequences of highly sensitive detection. For

[17] The notion of Pareto-dominance is discussed in more detail in Section 3.7.4.

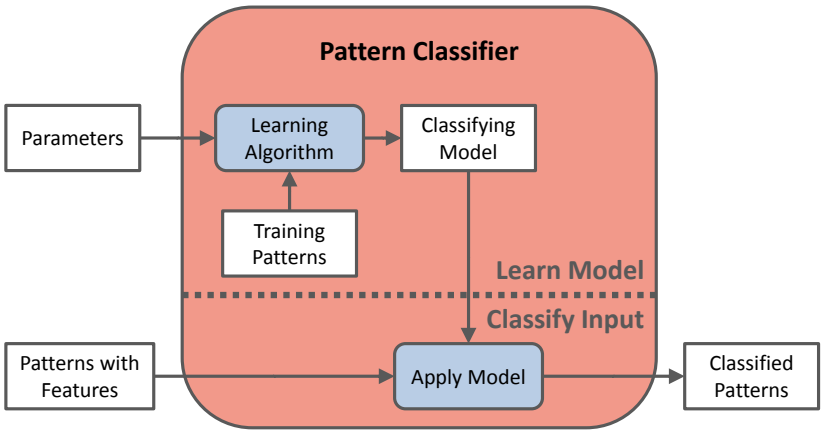


Figure 3.5: Pattern Classifier Input and Output. The pattern classifier module of *SynOpSis* receives detected patterns annotated with feature vectors as input data. In addition to the data, its second input is a parameter set configuring the employed learning algorithm. This learning algorithm computes a classifying model from training data with known ground truth class labels which are available e.g. via synthesis. The classifying model can be regarded as a mapping from feature space to class label space and is used to assign predicted class labels to the input patterns. Thus the output of the pattern classifier consists of the input patterns, annotated with predicted class labels.

this purpose it uses patterns detected in synthetic data with known ground truth labels, to train a classifying model that predicts whether an input pattern belongs to the target or to the non-target class. After training, this model can be used to classify the real input data.

As to be described in Section 3.6.1, besides training data, many learning algorithms require configuration in terms of parameter sets. Careful tuning of these parameter sets can considerably increase results quality. Like with the detector, parameter tuning is automated by optimizing suitable objective functions which are presented in Section 3.6.2.

3.6.1 Input and Output

Figure 3.5 shows a detail view of the pattern classifiers appearing in Figure 3.2. Furthermore, it illustrates the process of machine learning-based classification. The first input of the pattern classifier is a parameter set configuring the employed learning algorithm. The choice of learning algorithm determines the number and kinds of these parameters. In *SynOpSis*, supervised classification is used. Examples of applicable supervised learning algorithms and some of their parameters are the number of regarded neighbors in k-Nearest Neighbors (k-NN) [HTF09], the number of features available for splitting at each node in a Random Forest [Bre01] or the regularization parameter and choice of kernel function in an SVM [MMR+01]. These algorithms can all be filled in as the learning algorithm in Figure 3.5.

Independent of the concrete learning algorithm chosen for classification, its task is to compute a classifying model that predicts class labels for observed data examples. A data example is represented as a feature vector $\mathbf{f}$ from a certain feature space. This space can be mixed, containing continuous and discrete variables. Discrete variables may be from finite or infinite sets and may be ordered or unordered. Not every classifier supports all types of variables, thus type conversions like discretization may be necessary. The classifying model output by the learning algorithm can be represented as a mapping $\xi(\mathbf{f}) = p$ from feature

space to label space, i.e. p is the label *predicted* by the classifying model, and p originates from the set $\{c_1, \ldots, c_C\}$ of C possible class labels.

In order to compute this classifying model, the learning algorithm needs, besides its parameters, training data: In supervised learning the training data consists of a set of pairs $(\mathbf{f}, t)$, each containing a feature vector $\mathbf{f}$ and its associated *ground truth* label $t \in \{c_1, \ldots, c_C\}$. The number of such pairs is the size of the training dataset. Computing the classifying model $\xi(\mathbf{f})$ is typically conducted by minimizing a loss function between the predicted labels $\xi(\mathbf{f}) = p$ and the corresponding ground truth labels t over all ground truth-labeled examples $(\mathbf{f}, t)$ used as training data. The classifying model ξ is an abstraction of the training data and a representation of the information learned from executing the learning algorithm on it. Since ξ is defined on the entire feature space, it can not only be evaluated for the training example points but also for any other points in the feature space. Therefore, ξ can be used to compute predicted class labels p for unseen examples from the same feature space. In Figure 3.5, the classifying model ξ is applied to the second input of the classifier, consisting of the observed examples to be classified, which are represented by their feature vectors $\mathbf{f}$. The output of the classifier is a predicted class label $p = \xi(\mathbf{f})$ for each input example to be classified.

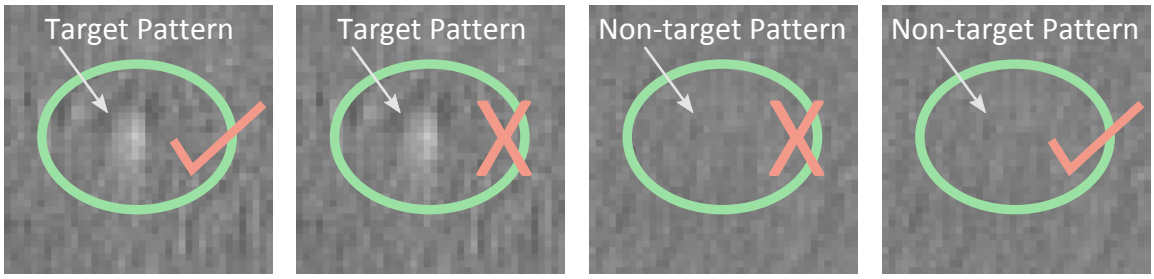
Most learning algorithms are rather generic. Application specifics are located primarily in feature extraction: The features are designed, aiming at the best possible separation of classes in feature space; how this goal can be achieved, heavily depends on application context, cf. Section 6.2.

In *SynOpSis*, the data examples are feature vectors computed for the responses of the pattern detector. Such responses can be from one of two classes: The target class[18] is for TP detector responses, whereas the non-target class is for FP detector responses. Training data with known ground truth class labels for supervised learning is available via synthesis: The training examples are obtained by running the pattern detector on synthetic images, followed by matching its responses to the ground truth patterns. Labeling *detector* results is necessary because the classifying model must learn to classify the patterns as they are provided by the detector, as opposed to classifying the patterns as they are represented in the *ground truth*, since these representations and appearances might differ considerably. Using training data gathered from synthetic images renders manual ground truth labeling unnecessary. It is assumed that the signal model is accurate enough such that a classifying model learned from the synthetic training data yields a good generalization performance towards real data.

In summary, *SynOpSis* uses the *Synthesis* stage to provide the training data for the pattern classifier in Figure 3.5. The parameters of the employed learning algorithm are found by the *Optimization* stage, using the objectives described in the subsequent section. During this optimization, the input patterns originate from synthetic data as well, to enable automatic evaluation of the objectives assessing classification quality. In contrast to that, in the *Application* stage, the input patterns for the classifier are computed from the real data to be analyzed. In both cases the output of the classifier consists in annotating the input patterns with predicted class labels, aimed at separating target from non-target patterns.

Chapter 6 provides details on how the pattern classifier and further associated data processing are realized for *PAMONO* data analysis, including the employed application-specific features in Section 6.2.

[18] In case multiple types of objects of interest are to be distinguished, there can be more than one target class label. Not all learning algorithms and performance measures support the multi-class case.



(a) TP **(b)** FN **(c)** TN **(d)** FP

Figure 3.6: Confusion Matrix of a Two-Class Classification Task – Illustration. Exemplary *PAMONO* data is shown to illustrate classification successes and errors. Detector outputs are indicated by green color (ellipses), while classifier outputs are indicated by red color (checkmarks and crosses). The classifier receives patterns found by the detector as inputs (ellipses). As output, the classifier predicts, whether they belong to the target class (checkmark) or to the non-target class (cross). True Positive (TP) *detector* responses relate to target patterns (a)–(b). Their *classification* can either be correct, constituting a TP of classification (a) or incorrect, constituting a False Negative (FN) (b). Analogously, False Positive (FP) *detector* responses relate to non-target patterns (c)–(d). Their correct *classification* constitutes a TN of classification (c), whereas their incorrect classification constitutes a False Positive (FP) (d).

3.6.2 Objectives

Besides being used in the production of ground truth-annotated training data, synthesis is also used in optimizing the parameter set of the learning algorithm in the pattern classifier: Like with the pattern detector, the *Optimization* stage in Figure 3.2 finds suitable values for the parameter set of the learning algorithm by optimizing automatically evaluable measures of classification quality. These measures are used as objective functions with respect to which the parameters are optimized. Therefore, the parameters of the learning algorithm can be tuned automatically.

Confusion Matrix of a Two-Class Classification Task

As with the detector in Section 3.5.2, the objective functions are defined using the entries of a 2×2 confusion matrix. These entries are visualized in Figure 3.6, illustrating classification successes and errors with respect to exemplary *PAMONO* data: The patterns provided by the detector are marked as green ellipses. Based on the features of each pattern, the classifying model makes predictions of class labels, indicated as red checkmarks and crosses, respectively. Patterns predicted to belong to the target class are indicated by a red checkmark; those predicted to belong to the non-target class are indicated by a red cross. Ground truth class labels are provided as text. There are four cases: Target patterns, shown in (a)–(b), can either be predicted correctly, constituting a TP of classification (a), or incorrectly, constituting an FN (b). These are the two cases relating to TP *detector* responses. Analogously, non-target patterns, shown in (c)–(d), can either be predicted correctly, constituting a TN of classification (c), or incorrectly, constituting an FP (d). These are the two cases relating to FP *detector* responses.

Table 3.2 shows the corresponding confusion matrix of the pattern classifier. The classifier receives all detector responses as inputs, i.e. its TPs and FPs because detection knows only positive responses. Classification means separating these two classes, hence they become

Table 3.2: Confusion Matrix of a Two-Class Classification Task. The input of the classifier are all detector responses, i.e. its True Positives (TPs) and False Positives (FPs) because detection knows only positive responses. Classification means separating these two classes: The detector's TPs correspond to target patterns and its FPs to non-target patterns. The detector's TPs become the positive ground truth class of classification (left column). They can either be classified correctly (TP classifier prediction) or incorrectly (False Negative (FN) classifier prediction). The detector's FPs become the negative ground truth class of classification (right column). They can either be classified correctly (True Negative (TN) classifier prediction) or incorrectly (FP classifier prediction).

		Ground Truth	
		Positive (TPs of detector)	Negative (FPs of detector)
Classifier	Positive	TP	FP
	Negative	FN	TN

the ground truth labels, listed as column headings in the matrix. The TPs of the *detector* become the ground truth positive class of *classification* (left column). If the classifier predicts such an example positively, it counts in the TP entry, otherwise it counts as FN because a target pattern was predicted as non-target by the classifier. The FPs of the *detector* become the ground truth negative class of *classification* (right column). If the classifier predicts such an example positively, it counts in the FP entry because a non-target pattern was predicted as target, otherwise it counts as a TN.

Note that it is vital to strictly distinguish between the confusion matrix of the detector and that of the classifier: Even though the entries have the same names, they characterize results from very different processing stages. Defining the objective functions for classification, which is done now, *always* refers to the quantities in the confusion matrix of classification.

For brevity, the following paragraphs discuss only the measures used as objectives in optimizing *PAMONO* pattern classification. Appendix A gives a selection of further objectives, suitable for other applications. In addition to that, Sokolova and Lapalme [SL09] provide an overview of performance measures for binary and multi-class classification tasks.

Objective 1: Classifier Recall

The first objective with respect to which classifier parameters are optimized was also used in optimizing the detector: Recall [HG09], which is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{3.4}$$

The difference between this definition and detector Recall is that classifier Recall refers to the entries of the classifier confusion matrix in Table 3.2. Optimizing the classifier for Recall is beneficial because the goal of not missing any target patterns in the data also exists in classification. However, the reason for having a classifier in *SynOpSis* was to use it for separating target from non-target patterns, and optimizing it solely for Recall does not promote classifier parameters that are good at achieving this goal. Therefore, Recall is complemented with another objective doing exactly that.

Objective 2: Classifier Precision

Precision [Pow11] measures the ratio of patterns correctly predicted to be target patterns among all patterns predicted to be target patterns:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{3.5}$$

Hence optimizing Precision rewards decreasing the number of FPs and serves as a complementary "force" to detector and classifier Recall: Classifiers with high values for Precision are more likely to label non-target patterns as negatives because this ability is measured by Precision.

Note that in the given scenario of post-detection classification, penalizing a high number of FPs is to be preferred over rewarding a high number of TNs. The reason is that if classification serves to sort out the FPs of detection, TNs are not relevant: The goal is to attain as many TPs as possible, while avoiding errors, i.e. FPs and FNs. TNs in classification are errors in detection and as such they should not be rewarded by the objectives to be optimized: Parameter sets that produce more TNs, while all other entries of the confusion matrix remain unchanged are not better than parameter sets producing fewer TNs. Parameter set quality is *invariant* to the number of attained TNs.

Precision and Recall in Conjunction

Optimizing the classifier simultaneously for Precision and Recall captures exactly the three relevant entries from the confusion matrix in Table 3.2. TPs are rewarded, while FPs (type I errors) and FNs (type II errors) are penalized with equal weight. Optimizing both objectives *simultaneously* seeks to attain high values in both, hence avoiding the trivial extremes: Recall = 1 can trivially be achieved by a classifier always predicting the positive class, but results in poor Precision if there are many examples in the negative class. Conversely, a classifier that finds at least one TP and predicts all other examples to belong to the negative class achieves Precision = 1 but incurs poor values for Recall if there are many examples in the positive class. An advantage of simultaneously optimizing multiple objectives is that good values in one objective can not "remedy" bad values in other objectives. The latter is typically a problem when only a single composite objective is optimized, that is computed as a weighted sum of several original objectives.

A possible alternative if single-objective optimization is demanded is Area under the ROC Curve (AUC) [Faw06]. AUC is defined as the integral of the Receiver Operating Characteristic (ROC) curve, the computation of which requires a classifier that yields not only a binary classification, but also a measure of confidence in that classification. Details on the computation of ROC/AUC and further possible objective functions for classifier optimization are given in Appendix A. If single-objective optimization is desired but the classifier does not provide confidence values, another possible alternative is using F_β score [Chi92] as the objective, which is also defined in Appendix A. F_β score is the weighted harmonic mean of Precision and Recall and allows to control the relative importance of both via the free parameter β. By being derived from Precision and Recall, F_β score is invariant to the number of TNs. In *SynOpSis*, aggregate objectives like AUC and F_β score enabling single-objective classifier optimization are not considered because the necessity of also optimizing the detector makes the task an inherently multi-objective one. However, a different approach, using

desirability functions to enable single-objective optimization in *SynOpSis*, is described in Section 3.8. The advantages of this approach are that it can aggregate over detector *and* classifier objectives, while integrating expert preferences and demanding for good values in *every* constituent objective.

Reducing Undue Optimism in Objective Values

In the *Optimization* stage in Figure 3.2, the pattern classifier and the evaluation of its objectives are run in a K-fold cross-validation [Koh95] on the synthetic input (cross-validation is neither shown in Figure 3.2 nor 3.5 to avoid cluttering). This means that the synthetic input is divided into K disjoint subsets of approximately equal size. A classifying model is trained using the union of $K-1$ such subsets of detected patterns as training data. This model is used to classify the patterns in the remaining subset, and objectives are evaluated on the obtained classification. This is repeated K times, until every subset has been classified once, and objectives are averaged over these K folds. Consequently, the scheme in Figure 3.5 and the evaluation of objectives are run K times. Computing objectives in a cross-validation avoids optimism in the reported objective values, which would otherwise be caused by overfitting, if objectives were evaluated on the same data that was used for training. Cross-validation and its merit in parameter tuning is explained in more detail in the context of model selection and performance estimation. Theory behind both topics is provided in Section 3.9, while the concrete setting used in *PAMONO* data analysis is part of the experimental setup, and thus given in Section 7.3.4. Computation of the final model used in the *Application* stage of Figure 3.2 depends on the setting described in that latter section and is thus detailed in Section 7.3.5.

3.7 Optimization Stage

Manual tuning of algorithmic parameters to make them suit a given problem instance has been called "more of an art than a science" by Bergstra et al. [BBB+11]. They furthermore state that recent improvements of results for image classification benchmarks are often due to finding better parameters for existing approaches, rather than being due to better new approaches. These reasons make automatic tuning of algorithmic parameters a desirable goal. Furthermore, in the context of *PAMONO* data analysis, having an automatic procedure for determining algorithmic parameters bears huge practical benefits because lab workers using the *PAMONO* sensor are no longer required to possess extensive knowledge of how to configure the algorithms involved in data analysis. Instead of conducting tedious manual search in a high dimensional parameter space, they can focus on their actual tasks.

In the *SynOpSis* approach, automatic parameter tuning is realized in the *Optimization* stage, cf. top right part of Figure 3.2. The parameter space of the optimization problem is spanned by the computational parameters of the algorithms constituting the detector and classifier (the parameters for *PAMONO* are listed in Sections 5.7 and 6.8). Suitable parameters for both these modules are found by optimizing a number of objective functions with respect to those parameters (the objectives for *PAMONO* are listed, and their computation is explained in Sections 3.5.2 and 3.6.2). This process automatically adapts them to changing physical parameters in a sensor setup.

The remainder of this section is organized as follows: Section 3.7.1 presents related work on optimization which can be used in implementing the *Optimization* stage in *SynOpSis*. Among the algorithms sketched here, Section 3.7.2 identifies Multi-Objective Genetic Algorithms (MOGAs) as a suitable choice for optimizing *PAMONO* data analysis. Subsequently, this class of algorithms is briefly sketched, while introducing the necessary terminology: Section 3.7.3 gives a short introduction to genetic algorithms in general, while Section 3.7.4 covers the multi-objective case. Section 3.7.5 concludes the general part by summarizing Non-Dominated Sorting Genetic Algorithm II (NSGA-II) as a successful representative of the class of MOGAs and giving the rationale why it was chosen in implementing the *Optimization* stage. Finally, Section 3.7.6 describes two different fashions in which *SynOpSis* can be optimized: a sequential and a global fashion. Note that Sections 3.7.3 to 3.7.6 are prerequisites for the depiction of the overall genetic algorithm setup used for *PAMONO* data analysis, which is given as part of the experiment description in Section 7.3.2.

3.7.1 Related Work

Numerous techniques for the optimization of objective functions have been proposed over the last decades. Some of these techniques date back to the 1960s and earlier and are still successfully applied today. A small selection of these techniques, providing an overview, is listed here.

- **Brute force** approaches can be applied if the parameter space is very small or objective function evaluations are very fast/cheap. They work by simply enumerating a large number of parameter sets and evaluating the objectives. Two popular brute force approaches are **grid search** [BB12] and **Latin Hypercube Design (LHD)** [MBC79].
- The **method of steepest descent** [Kel99] requires (an approximation of) the gradient of the objective function to be optimized. It iteratively determines new parameter sets by stepping through parameter space in the direction of the negative gradient, i.e. the direction of steepest descent (in case of minimization). Step lengths must be chosen carefully in each iteration.
- **Newton's method** and its offspring like the **Gauß-Newton method** or **quasi-Newton** methods like **Broyden-Fletcher-Goldfarb-Shanno (BFGS)** (all of these methods are described in [NW06]) analytically optimize local quadratic model functions obtained by second-order Taylor expansion around iterated expansion points. The minimizer determines the Gauß-Newton step through parameter space, and after taking that step, the method is iterated. The listed methods differ mainly in whether and how they approximate the Hessian matrix in the local quadratic model. Again, step lengths must be chosen carefully in each iteration. Another hazard arises by the fact that in regions that are far from local minimizers, the Newton-steps can not be guaranteed to improve the objective.
- The **Levenberg-Marquardt algorithm** [Mar63] addresses this issue by continuously blending between Gauß-Newton and steepest descent steps, depending on estimated proximity to a local minimizer: When closer to a minimizer, the Levenberg-Marquardt step becomes more similar to the Gauß-Newton step because here Gauß-Newton converges faster than steepest descent. When further away, the Levenberg-Marquardt step becomes more similar to the steepest descent step because unlike the Gauß-Newton step, it is guaranteed to improve the objective.

- In contrast to the previous three methods, the **Nelder-Mead algorithm** [NM65] requires no derivatives of the objective function. It is the nonlinear generalization of the linear simplex algorithm [Dan98]. For initialization in a d-dimensional parameter space, it firstly computes objective values for $d+1$ points, that are affinely independent. These points thus span a simplex in parameter space. Traversal of parameter space is conducted by iterated replacement of the worst-performing point with a new point that is computed on the basis of the other simplex points and that aims to improve the objective.

Note that all these algorithms, except for the brute force ones and Nelder-Mead, require availability of (partial) derivatives of the objectives to be optimized.

More recent approaches that are more specific to optimizing parameter sets of algorithms were already discussed as related work in Section 3.3. None of those methods require derivatives since in that scenario derivatives are rarely available. The discussed methods traverse parameter space solely by function evaluations like the model-free methods in [HHL+09; BSP+02; BYB+10], or by meta-models that are created solely from function evaluations, like in [BLP10; KKF+11; HHL11; BBB+11; BMT+12]. The third discussed class of derivative-free methods are Evolutionary Algorithms (EAs) [AST09; MMB+14a; MMB+14b; Luk13; Deb01] which simulate biological evolution among a population of candidate parameter sets to create better candidates by mutating and crossing them over. Genetic Algorithms (GAs) [SP94] are a widely used class of EAs, but further classes, like evolution strategies and genetic programming exist [Luk13; Deb01]. Hybrid or (synonymously) Memetic Algorithms (MAs) [Mos89] are EAs that combine evolutionary methods with local search. In a wider scope, the more general term of *evolutionary computation* includes additional biology-inspired optimization techniques, e.g. Ant Colony Optimization (ACO) [DB05] and Particle Swarm Optimization (PSO) [Ken10; RC06]. EAs readily extend to multi-objective optimization.

As a summary, the preceding paragraphs listed derivative-based and derivative-free algorithms for optimization, some of which can handle the multi-objective case. Provided that the examined optimization problem fulfills the assumptions of the respective algorithm, any of the presented algorithms can potentially serve to implement the *Optimization* stage in Figure 3.2. This is enabled by the modular architecture of *SynOpSis*.

3.7.2 Algorithm Choice for Optimizing PAMONO Data Analysis

Concretely choosing the algorithm for realizing the *Optimization* stage in Figure 3.2 is equivalent to selecting how to implement the "Update Parameters" box in the figure. In order to choose a certain optimization algorithm, the particular problem to be solved must be considered. As this thesis focuses on *PAMONO* data analysis, an optimization algorithm suiting this application case is identified in the remainder of this section. From the optimization point-of-view, the *PAMONO* data analysis task has the following properties:

- It has a **large parameter space**: The pattern detector for *PAMONO* has 28 free parameters, cf. Section 5.7. In addition, there are between zero and three parameters for the pattern classifier, cf. Sections 6.6 and 6.8. This large number of parameters leads to combinatorial explosion, rendering exhaustive approaches like grid-based optimization infeasible.

- The parameter space is a **mixture of continuous and discrete variables** (mixed-integer [BKL+13]), with approximately equal numbers of boolean, integer and floating point parameters.
- **Approximating derivatives is impeded** by the large number and mixed-type nature of the parameters.
- *PAMONO* data analysis involves **multiple objectives**, cf. statements of objectives in Sections 3.5.2 and 3.6.2.
- The objectives are **non-linear** and **non-convex**, as can be seen by empirically evaluating a number of sample points.

Given these properties of *PAMONO* data analysis, the decision was made in favor of using a Multi-Objective Genetic Algorithm (MOGA) due to the following advantages of this technique:

- Evolutionary approaches to optimization are general and easy to use, while at the same time very successful in practice as reported in, amongst many others, [AST09; MMB+14a; MMB+14b; ARR+07; BN07; DK07; DPM00; GBG05; HBK10; KBM+09; MKB09]. Evolutionary approaches are a quick way of exploring and quantifying the potential for optimization in a given problem. They can serve as an initial step for developing a more in-depth understanding/model of the underlying problem and for identifying the components of a solution that contain the largest potential gain.
- Genetic Algorithms (GAs) can cope with the large parameter space in *PAMONO* and are flexible enough to handle the mix of boolean, integer and floating point variables arising in this application.
- No derivatives are required for running a GA. The only requirement is that the fitness, i.e. the objective function(s), must be evaluable in some way. Hence, e.g. even energy measurements can be optimized, cf. [Tim12; LSW13; LMS+14; NLE+15; LKD+14].
- GAs do not make any assumptions about the underlying objective landscapes. They can handle non-convex objectives [AST09] and escape from local optima due to their randomized subcomponents [Luk13].
- By extending them to MOGAs, GAs can readily handle the multi-objective optimization arising in *PAMONO*. Several MOGAs that have been demonstrated to be successful in practice are available, e.g. NSGA-II [DPA+02], SPEA2 [ZLT01], PAES [KC99] or SMS-EMOA [BNE07].
- One of the goals during *PAMONO* prototype development is identification of the trade-offs between opposing objective functions. In this context, the fact that MOGAs are population-based pays off: They do not only evolve a single parameter set, but create a so-called front of Pareto-optimal points (cf. Section 3.7.4), containing different parameter sets that are non-dominated with respect to the objectives. From this front, the trade-offs between objectives can easily be identified. In this regard, the mechanisms of diversity preservation that are present in most MOGAs are another beneficial feature: Diversity preservation encourages the parameter sets to be widely spread across objective space, such that points succeeding in either of the objectives can be found. Doing so enables thorough exploration and examination of the trade-offs between objectives.
- Randomization is advantageous in scenarios with a high-dimensional parameter space but low effective dimensionality [BB12]. In GAs, randomization occurs in the initialization,

mutation and crossover components, cf. Section 3.7.3. The advantageous effects of randomization can be summarized as obtaining better results quality while using considerably fewer function evaluations. These effects were shown in comparison to grid search for optimization problems where the parameter space consists of dimensions with different importance for the objective value: Randomized search strategies better handle[19] the case of low effective dimensionality, i.e. the case where only a subset of the parameter dimensions has a large influence on the objective. In practice, this is often the case [CMO97].

- Despite employing randomized subcomponents, GAs harvest the advantages of exploiting knowledge from previous evaluations, as e.g. meta-modeling does. Knowledge acquired during simulated evolution is in effect where new parameter sets are generated from well-performing old parameter sets.
- As a last point, GAs lend themselves to parallelization: Objective values for multiple parameter sets can be computed simultaneously by distributing the parameter sets to different compute nodes. Communication overhead is very low because the data to be processed needs to be transferred only once, and parameter sets typically are very small in comparison to that data. For *PAMONO* data analysis, each compute node needs to have a Graphics Processing Unit (GPU) because it executes the bulk of the computation. A system for distributing *PAMONO* evaluation across a compute cloud is presented in [LMS+14].

With the rationale for choosing a MOGA in implementing the *Optimization* stage of *SynOpSis* given, this class of algorithms is now briefly summarized, and required terminology is introduced. Section 3.7.3 covers genetic algorithms in general, while Section 3.7.4 provides extensions for the multi-objective case. These abstract depictions of concepts lead to a concrete MOGA implementation in terms of NSGA-II [DPA+02] in Section 3.7.5. Finally, Section 3.7.6 describes how optimization of *SynOpSis* can be conducted in a global and in a sequential fashion. Note that the depiction of the overall genetic algorithm used to optimize *SynOpSis* in the context of *PAMONO* is given later, in Section 7.3.2 because it is closely associated with the experiment description.

3.7.3 Genetic Algorithms

A key difference between Genetic Algorithms (GAs) and many other search heuristics is that GAs are population-based methods, maintaining more than one candidate solution at a time. In conjunction with the fact that GAs are heavily inspired by biological evolution, this leads to a whole new terminology employed in the context of GAs, which will be introduced in this section, following the textbook by Luke [Luk13]. The order of presentation is bottom-up.

Terminology 3.2. *A **gene** corresponds to one parameter to be optimized and thus to one dimension of the parameter space. A fixed length vector of genes is denoted a **chromosome** and can be used to represent a point in parameter space, i.e. a complete parameter set. The objective functions, mapping chromosomes from parameter space to objective space are denoted **fitness functions**. Consequently, the point in objective space obtained by evaluating all fitness functions for a given chromosome is called its **fitness**. A candidate solution is denoted an **individual**. One can refer to an individual in parameter space as well as in objective space,*

[19] Figure 1 in [BB12] very well illustrates the reason for this.

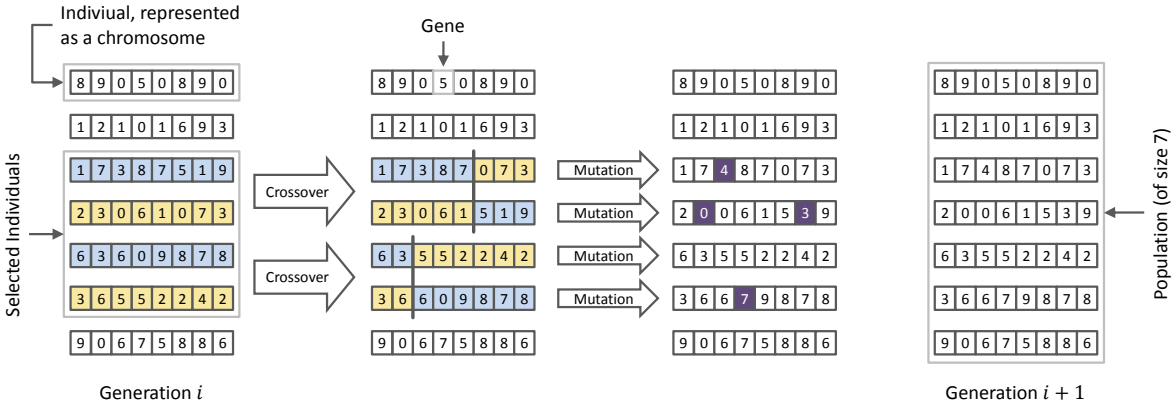


Figure 3.7: Genetic Algorithm (GA) Terminology and Operations. Each of the four columns shows a **population** consisting of seven **individuals**. Each individual is represented as a **chromosome**, which is a vector of **genes**. In the example, the number of genes is eight. Each gene encodes one parameter to be optimized. A GA first selects individuals, e.g. by tournament **selection**. The selected individuals are recombined via **crossover**, i.e. by exchanging genes. The figure shows two instances of **one-point** crossover. Subsequently individuals undergo **mutation**, i.e. randomly picked genes are assigned new values. After crossover and mutation, the resulting **child** individuals become the next **generation** of the GA, along with the **elite** (i.e. best-performing) individuals in the **parent** generation (here: first two and last line). Figure adapted from [Luk13].

*denoting the chromosome of the solution and its fitness, respectively. A set of individuals is called a **population**. One iteration of a GA is called a **generation**. Homonymously, the respective population generated in that iteration can also be denoted as a generation.*

Figure 3.7 illustrates this terminology and furthermore introduces three crucial operations involved with GAs: selection, crossover and mutation. These are described now, in the context of a detailed explanation of Algorithm 3.1, following [Luk13]. This algorithm is a basic, single-objective GA with elitism, i.e. the best-performing individuals in each generation are maintained in the subsequent one. Elitism has been demonstrated to increase speed of convergence for Multi-Objective Genetic Algorithms (MOGAs) [ZDT00; Rud99], that will be discussed in Sections 3.7.4 to 3.7.5. Algorithm 3.1 receives as inputs the size S of the population in each generation, along with the number G of maximum allowed generations and the number E of elite individuals that are passed on to the subsequent generation. The output of the algorithm is the chromosome $\mathbf{x}^* \in \mathbb{P}^P$ of the best-performing individual that was encountered in the course of the GA. $\mathbb{P}^P$ is the P-dimensional parameter space, where $\mathbb{P}$ is a placeholder for other sets: Each dimension of $\mathbb{P}^P$ may be from a different set, e.g. booleans, ordered/unordered discrete values or real values.

Initialization of the population $\mathbf{X}$ with S individuals from $\mathbb{P}^P$ is the first step in the GA. Typically, an individual $\mathbf{x} \in \mathbb{P}^P$ is initialized by drawing each of its gene values randomly from the set of values allowed for that gene. The distribution from which it is drawn may be biased towards regions known to contain good values for that gene. Furthermore, it is possible to include entire individuals known to perform well, or handcrafted individuals into the initial population.

After initialization of the population, the best-performing individual $\mathbf{x}^*$ is initialized to the dummy individual $\square$ to indicate that no individual was evaluated yet. Furthermore, the

Algorithm 3.1 Genetic Algorithm (GA) with Elitism [Luk13]

Input: Population size S, number of generations G, elite size E, with $S - E$ even
Output: Best-performing individual $\mathbf{x}^*$

$\mathbf{X} \leftarrow$ `initializePopulation`(S) // ..... e.g. random values for all genes
$\mathbf{x}^* \leftarrow \square$
$g \leftarrow 1$ // ..... initialize generation counter
$d \leftarrow$ `false` // ..... 'done'-flag for termination
while $\neg$d **do**
  for each individual $\mathbf{x}^i$ in $\mathbf{X}$ **do**
    $y^i \leftarrow \varphi(\mathbf{x}^i)$ // ..... evaluate and remember fitness
    if $(\mathbf{x}^* = \square) \vee (y^i \triangleright y^*)$ **then**
      $\mathbf{x}^* \leftarrow \mathbf{x}^i$ // ..... $\mathbf{x}^i$ performs better than previous best $\mathbf{x}^*$, thus update
      $y^* \leftarrow y^i$ // ..... also track best observed fitness
    end if
  end for
  $\widehat{\mathbf{X}} \leftarrow$ `getElite`$(\mathbf{X}, E)$ // ..... E fittest individuals from $\mathbf{X}$, random choice for ties
  for $(S - E)/2$ iterations **do**
    $\mathbf{x}^a \leftarrow$ `selection`$(\mathbf{X})$ // ..... select parent $\mathbf{x}^a$ from $\mathbf{X}$
    $\mathbf{x}^b \leftarrow$ `selection`$(\mathbf{X})$ // ..... select parent $\mathbf{x}^b$ from $\mathbf{X}$
    $\widehat{\mathbf{x}}^a, \widehat{\mathbf{x}}^b \leftarrow$ `crossover`$(\mathbf{x}^a, \mathbf{x}^b)$ // ..... create two children via crossover
    $\widehat{\mathbf{x}}^a \leftarrow$ `mutation`$(\widehat{\mathbf{x}}^a)$
    $\widehat{\mathbf{x}}^b \leftarrow$ `mutation`$(\widehat{\mathbf{x}}^b)$
    $\widehat{\mathbf{X}} \leftarrow$ `append`$(\widehat{\mathbf{X}}, \widehat{\mathbf{x}}^a, \widehat{\mathbf{x}}^b)$ // ..... append new individuals to $\widehat{\mathbf{X}}$
  end for
  $\mathbf{X} \leftarrow \widehat{\mathbf{X}}$ // ..... population for next generation
  $g \leftarrow g + 1$
  $d \leftarrow (y^* = y^{\text{best}}) \vee (g = G)$ // ..... terminate on best-possible fitness or last generation
end while
return $\mathbf{x}^*$

generation counter g is initialized to 1 and the termination flag d is set `false`. Then, as long as d remains `false`, the following procedure is carried out: Firstly, for each individual $\mathbf{x}^i$ in the current population $\mathbf{X}$, the fitness function $\varphi(\mathbf{x}^i)$ is evaluated, yielding the fitness value y^i. If $\mathbf{x}^*$ is the dummy $\square$ or if y^i is better than the current best observed fitness y^* of $\mathbf{x}^*$, then $\mathbf{x}^i$ and y^i become the new $\mathbf{x}^*$ and y^*. Whether y^i is better, is determined with the *better than*-relation $\triangleright$ between scalar objective values: If objective φ is to be maximized, $y^i \triangleright y^*$ means $y^i > y^*$ and if it is to be minimized, it means $y^i < y^*$.

With all fitnesses known for the current population $\mathbf{X}$, its elite is determined by identifying the E best-performing individuals in $\mathbf{X}$. Among individuals with identical fitness, random draws are conducted until the number E has been reached. The elite is stored in the population for the next generation, denoted as $\widehat{\mathbf{X}}$. The remaining $S - E$ individuals are generated within $(S - E)/2$ iterations of a **breeding** process, consisting of selection, crossover and mutation, as described now. Each iteration of this process creates two new individuals.

Selection is the first operation in the breeding process. The selection technique determines how individuals are picked for breeding. The most common selection technique is **tournament selection** [Luk13]: In a tournament of size $T \in \mathbb{N}_{>0}$, one individual is drawn at random from a population to sequentially compete against $T-1$ other individuals. In each iteration, a competitor individual is drawn at random from the population and if it is better in terms of fitness, it becomes the new best individual observed within this tournament. The best individual after the last iteration wins the tournament and is selected. Note that for $T = 1$, tournament selection becomes **random selection** and that for increasing T, it becomes increasingly selective because high fitness values become more and more important in order to win a tournament. Further selection techniques, like **fitness-proportionate selection** and a variation thereof, called **stochastic universal sampling**, can be found in the literature [Luk13].

Crossover follows selection and uses the two selected parent individuals to form (usually) two child individuals by exchanging parts of their chromosome. Figure 3.7 shows two examples of the so-called **one-point crossover**: A random crossover point is selected in the chromosomes of the two individuals to be crossed over, and the gene values following that crossover point are swapped between the two individuals. Note that in one-point crossover the probability of breaking e.g. the first and last entry of the chromosome vector apart is much higher than for two adjacent entries in the vector. This can cause so-called linkage problems, namely if parameters that need to work together to attain a good fitness have a high probability of being broken apart. To resolve this, one can either place them closer together on the chromosome, thus increasing the probability that they will be crossed over en bloc, or one can choose two-point or uniform crossover: **Two-point crossover** chooses two crossover points instead of only one, and swaps the gene values between those points. **Uniform crossover** swaps each gene value independently with a certain probability. Note that a simulated evolution using solely one of these crossover mechanisms can not generate individuals outside the bounding box in parameter space of the initial population: Such an evolution would be a local and not a global search.

Mutation is a mechanism that enables the simulated evolution to escape the bounding box of its initial population and that makes it a global search. In general, mutation means that randomly chosen genes in the chromosome of an individual are assigned new values determined in some randomized way, as illustrated for four individuals in Figure 3.7. Assigning randomized values to randomly picked dimensions in the parameter space obviously allows the search to escape the bounding box in parameter space of the initial population. Furthermore, it makes every point in parameter space reachable with non-zero probability, hence making the search global. Mutation is carried out immediately after crossover in Algorithm 3.1. Note that different types of genes require different mutation procedures. For genes encoding binary variables, the **bit-flip mutation** is very common. It inverts the bit value in each binary gene with a given probability. For genes encoding integer variables, **integer randomization** can be used, where randomly chosen genes are assigned new random values from their sets of allowed values (as in Figure 3.7). An alternative is **random walk mutation**, where the old gene value is used as the starting point of an integer random walk, the end point of which becomes the new gene value. For genes encoding floating point variables, usually **Gaussian convolution** is used as the mutation operator: For randomly chosen genes, values drawn

from a Gaussian distribution with zero-mean and a certain variance are added to the value of the gene.

After selection, crossover and mutation, Algorithm 3.1 continues by appending the two newly bred individuals to the population $\widehat{\mathbf{X}}$ containing the elite from $\mathbf{X}$. After $(S-E)/2$ iterations of this breeding loop, $\widehat{\mathbf{X}}$ contains S individuals and is used as the population of the next generation. The generation counter is increased by one and termination criteria are checked: If y^* already is the best-possible attainable value of the objective, or if the maximum number G of generations has been reached, the main loop terminates, and the parameters $\mathbf{x}^*$ of the best-observed individual are returned.

3.7.4 Multi-Objective Genetic Algorithms

Algorithm 3.1 from the previous section specifies a basic, elitist GA for the *single-objective* case, i.e. only one objective function is optimized. The key difference between the single-objective and the multi-objective case is *comparability* between fitness values: In single-objective optimization, fitness values are scalar and therefore inherit the ordering of the underlying set, usually $\mathbb{R}$. Given two scalar fitness values $y^a, y^b \in \mathbb{R}$, it can be decided, which one is better, using the $<$ or the $>$ relation on $\mathbb{R}$, so comparability between individuals is not an issue. In contrast to that, in multi-objective optimization, fitness values are vectorial quantities. Given two vectorial fitness values $\mathbf{y}^a, \mathbf{y}^b \in \mathbb{R}^O$, where O is the number of objectives, it can occur that $\mathbf{y}^a$ is better than $\mathbf{y}^b$ in one objective while worse in another, and fitness values can not readily be compared. In order to generalize GAs to the multi-objective case, this comparability issue must be handled, requiring new terminology which is introduced now. Extending GAs in this direction gives rise to the concept of Multi-Objective Genetic Algorithms (MOGAs). Following [Deb01] and adapting the notation used therein, the following definition is made:

Definition 3.1. *Let $\mathbf{x}^a, \mathbf{x}^b \in \mathbb{P}^P$ represent two individuals a and b as chromosome points in the P-dimensional parameter space $\mathbb{P}^P$. Let $\varphi_i(\mathbf{x}) \in \mathbb{R}, i \in \{1, \ldots, O\}$ denote the value of objective function φ_i as obtained by a parameter space point $\mathbf{x}$. As the number of such objective functions is O, the objective space is $\mathbb{R}^O$. Let $\rhd$ denote the* better than*-relation between scalar objective values: If objective i is to be maximized, $\varphi_i(\mathbf{x}^a) \rhd \varphi_i(\mathbf{x}^b)$ means $\varphi_i(\mathbf{x}^a) > \varphi_i(\mathbf{x}^b)$ and if it is to be minimized, it means $\varphi_i(\mathbf{x}^a) < \varphi_i(\mathbf{x}^b)$. Then the parameters $\mathbf{x}^a$ are said to* ***dominate*** *the parameters $\mathbf{x}^b$ if the following conjunction of conditions holds:*

$$\begin{aligned} &\forall i \in \{1, \ldots, O\} : \varphi_i(\mathbf{x}^b) \not\rhd \varphi_i(\mathbf{x}^a) \\ \wedge &\exists j \in \{1, \ldots, O\} : \varphi_j(\mathbf{x}^a) \rhd \varphi_j(\mathbf{x}^b). \end{aligned} \tag{3.6}$$

The first line requires that the dominating point $\mathbf{x}^a$ is not worse than $\mathbf{x}^b$ in any objective i, while the second line requires that it is strictly better in at least one objective j.

In that case, a reasonable decision-maker would prefer $\mathbf{x}^a$ over $\mathbf{x}^b$, which can be regarded as an ordering of $\mathbf{x}^a$ before $\mathbf{x}^b$, making these individuals comparable. Besides this simple case, it can also occur that for two parameter sets $\mathbf{x}^a, \mathbf{x}^b$, neither $\mathbf{x}^a$ dominates $\mathbf{x}^b$, nor $\mathbf{x}^b$ dominates $\mathbf{x}^a$. If the individuals are not identical, then there is at least one objective that is better in $\mathbf{x}^a$ and one that is better in $\mathbf{x}^b$. In that case, a reasonable decision-maker can not (without introducing assumptions about the relative importance between objectives, bearing

the danger of comparing apples and oranges) decide which one to prefer. If this is the case, $\mathbf{x}^a$ and $\mathbf{x}^b$ are called **incomparable** [ZBT07]. A particularly important set of incomparable points is the set of non-dominated points, introduced in the following definition [Deb01]:

Definition 3.2. *A point $\mathbf{x}^a \in \mathbb{P}^P$ is called **non-dominated** if there exists no $\mathbf{x}^b \in \mathbb{P}^P$, that dominates $\mathbf{x}^a$. The set of all such non-dominated points is called the **non-dominated set**, or alternatively the **Pareto front** or **Pareto-optimal set**.*

Note that this definition assumes knowledge of objective values $\varphi_i(\mathbf{x})$ for every $\mathbf{x} \in \mathbb{P}^P$. Usually in optimization, no analytic forms of the φ_i are available (only in toy problems) and their evaluations are expensive. Therefore, in practice, the terms from Definition 3.2 are usually regarded with respect to the points $\mathbf{x}$ that were already visited during the optimization. That means the Pareto front consists of those parameter vectors that are non-dominated among the already visited (and kept[20]) parameter vectors. This concept of a non-dominated set is also referred to as an **approximation set** [ZBT07] because its elements may be dominated by the exact Pareto-optimal set considering *all* points in parameter space [Deb01], cf. Figure 3.8a for an illustration. The key difference in the quality of MOGAs lies in how well their computed approximation set captures the true Pareto front. This involves not only the distances between the computed fitness vectors from the true front, but also how well these fitness vectors are spread over the *entire extension* of the front. This concept of spread over the front is denoted as the **diversity** of the approximation set. High diversity is required for analyzing the trade-offs between objectives. Furthermore, it enables a reasonable decision-maker to pick a design point from a wide range of options in objective space. The ideal is a uniform distribution of the approximation set on the entire (unknown) theoretical Pareto front, enabling thorough examination of the available options.

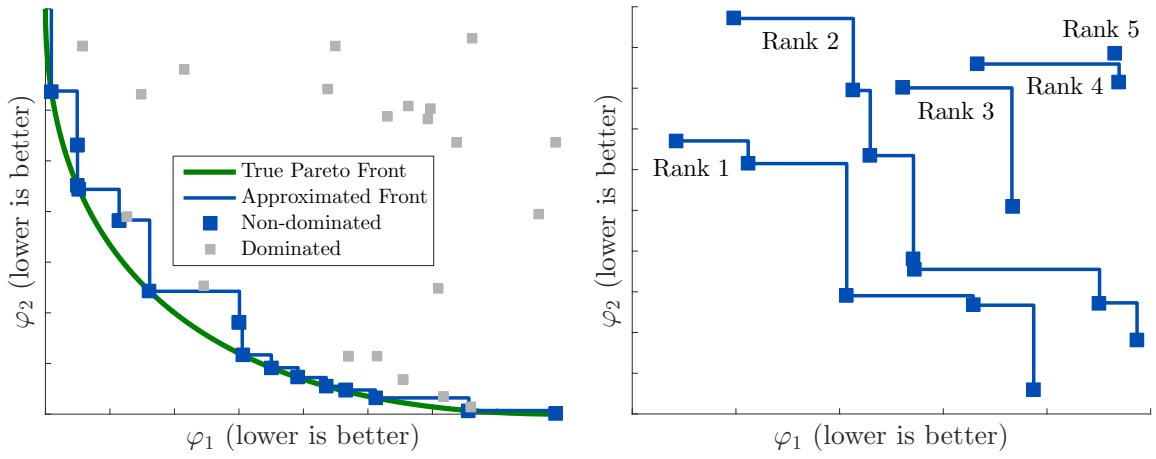
Given these prerequisites, the following issues can be identified that are to be addressed when designing a MOGA:

- **Comparison** of vectorial fitnesses as arise in multi-objective optimization must be enabled. The comparison relation is required for selection (and for determining the elite, in case of an elitist algorithm). Comparison is usually based on the Pareto-dominance relation. The tricky part is to decide between individuals that are incomparable with respect to the Pareto-dominance relation.
- **Diversity** of the computed Pareto front should be encouraged by the search strategy. Preservation of diversity may be used as a criterion aiding comparison and thus help in addressing the previous point.
- **Elitism** can optionally be implemented in a MOGA. Doing so has been demonstrated to increase convergence speed [ZDT00; Rud99].

Typically, MOGAs must use larger population and elite sizes than their single-objective counterparts because the number of individuals required to cover all regions of a Pareto front is larger than for finding a single optimized individual: It grows exponentially in the number of objectives to be optimized [Luk13].

Now that the goals, required terminology and issues to be addressed in designing a MOGA have been presented, a popular representative of this class of algorithms is summarized: NSGA-II [DPA+02].

[20] Deciding which points to keep in the population is an important problem in population-based methods, especially in the presence of multiple objectives, cf. Section 3.7.5.



(a) Pareto Front **(b)** Pareto Ranks

Figure 3.8: Pareto Front and Ranks. (a) shows the true Pareto front (in objective space) of an analytical toy problem as a green curve and an approximation to this front, found by a MOGA, as blue stairs. In real-world problems, the true Pareto front is typically unknown. The non-dominated set found by the MOGA is depicted as large blue squares, whereas dominated points are marked by small gray squares. The non-dominated points may lie on the true front as well as behind it, while the dominated points necessarily lie behind the front. The goal of a MOGA is to converge to an approximation set located as closely as possible to the true front, while being spread uniformly across it. (b) illustrates the concept of Pareto ranks as used e.g. by NSGA-II: The non-dominated points have Rank 1. If these points were removed from the set, the non-dominated points of the reduced set have Rank 2, analogously for higher ranks. Figures adapted from [Luk13].

3.7.5 Non-Dominated Sorting Genetic Algorithm II (NSGA-II)

Among the multitude of MOGAs available in the literature (e.g. NSGA-II [DPA+02], SPEA2 [ZLT01], PAES [KC99] or SMS-EMOA [BNE07]), the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) by Deb et al. [DPA+02] was chosen as the algorithm governing the *Optimization* stage of *SynOpSis* (cf. Figure 3.2). The reason for this choice was its consistent success in very diverse applications, as reported e.g. in [ARR+07; BN07; DK07; DPM00; GBG05; HBK10; KBM+09; MKB09].

NSGA-II will be briefly summarized now, with a presentation that proceeds along the list of issues to be addressed in designing a MOGA, as given at the end of the previous section. Unless otherwise stated, the depiction follows the original paper by Deb et al. [DPA+02].

Comparison of vectorial fitness values is done with respect to two quantities: Pareto ranks and sparsity. Pareto ranks are the primary criterion and will be explained here, while sparsity is treated as a black box for now and will be explained in the next paragraph about diversity. Figure 3.8b illustrates the concept of Pareto ranks: It shows a 2-D objective space where both objectives are to be minimized. Rank 1 consists of the non-dominated points and hence equals the computed Pareto front. If the points with rank 1 were removed and the front recomputed, the resulting new front consists of the points in rank 2, analogously for all further ranks. Generally speaking, rank i contains all points that are dominated only by the points in ranks $i-1, i-2, \ldots, 1$. Computing Pareto ranks is denoted as non-dominated sorting which gives NSGA-II its name. One of the major contributions of NSGA-II is a fast exact

algorithm for non-dominated sorting: A naïve implementation of non-dominated sorting has time complexity $\mathcal{O}\left(OS^3\right)$ and storage complexity $\mathcal{O}\left(S\right)$, where O is the number of objectives and S the population size. NSGA-II reduces time complexity to $\mathcal{O}\left(OS^2\right)$ while increasing storage complexity to $\mathcal{O}\left(S^2\right)$. Another major contribution of NSGA-II is an approach to comparing vectorial fitnesses: It uses the crowded-comparison relation $\succ_n$, which employs Pareto ranks as the primary, and sparsity as a secondary criterion in comparing vectorial fitnesses. Up to its description in the next paragraph, sparsity can be taken as a black box quantity, measuring for each individual, how rare similar individuals are in objective space. The comparison relation $\succ_n$ is defined as follows: For two objective vectors $\mathbf{y}^a, \mathbf{y}^b \in \mathbb{R}^O$, the relation $\mathbf{y}^a \succ_n \mathbf{y}^b$, meaning $\mathbf{y}^a$ is preferred over $\mathbf{y}^b$, holds if one of the following conditions holds:

- The Pareto rank of $\mathbf{y}^a$ is smaller than $\mathbf{y}^b$. This means $\mathbf{y}^a$ dominates $\mathbf{y}^b$ in the classical sense of Definition 3.1.
- The Pareto rank of $\mathbf{y}^a$ is equal to that of $\mathbf{y}^b$, i.e. $\mathbf{y}^a$ and $\mathbf{y}^b$ are incomparable in the classical sense, *and* the sparsity of $\mathbf{y}^a$ is larger than that of $\mathbf{y}^b$, i.e. individuals performing similarly to $\mathbf{y}^a$ are rarer in objective space.

Otherwise $\mathbf{y}^b \succ_n \mathbf{y}^a$. To put it shortly, NSGA-II compares individuals by preferring lower Pareto-ranks, and it breaks ties in ranks by preferring higher sparsity. Note that objective values are not used directly in the definition of $\succ_n$, but indirectly, within the computation of Pareto ranks and sparsity. NSGA-II uses the $\succ_n$ relation typically in a tournament selection scheme with tournament size $T = 2$. Furthermore, $\succ_n$ finds application in determining the elite individuals.

Diversity preservation and encouragement follow the goal of obtaining a set of non-dominated points that spreads over the entirety of the theoretical Pareto front of an optimization problem. In NSGA-II, diversity is preserved and encouraged by using sparsity as a secondary criterion in comparing individuals. Sparsity of an individual is computed by the following procedure: For each objective separately, only individuals in the same Pareto rank are regarded, and among those, the two individuals with the next smaller respectively next larger value in that objective are determined. If one of these neighbors can not be found, the individual resides on the boundary of its rank and is assigned infinite sparsity. If both neighbors exist, their difference in the regarded objective is normalized by the range of that objective. These normalized differences are accumulated over all objectives, and the resulting sum defines the individual's sparsity. Hence the further away the neighbors in objective space, the higher the individual's sparsity and the rarer similar individuals are in objective space. Preferring individuals with higher sparsity makes individuals in less crowded regions of objective space win tournament selection in case of equal Pareto ranks. This obviously preserves and encourages diversity of the computed front.

Elitism in NSGA-II is achieved by maintaining two separate populations in each generation: $\mathbf{X}^p$ is the parent population, corresponding to the elite individuals, which mate and generate the children population $\mathbf{X}^c$. Now the parent and children populations $\widehat{\mathbf{X}}^p, \widehat{\mathbf{X}}^c$ for the next generation are obtained as follows: Pareto ranks for the union of the previous $\mathbf{X}^p, \mathbf{X}^c$ are computed using non-dominated sorting. The elite parent population $\widehat{\mathbf{X}}^p$ for the next generation is filled up to the elite size E, starting with the individuals of rank 1, i.e. with

the non-dominated Pareto front of the union of $\mathbf{X}^p$ and $\mathbf{X}^c$. This is continued until the rank i is reached that would exceed E if added to $\widehat{\mathbf{X}}^p$. For this rank, the $\succ_n$ relation is used to determine the best individuals, which are appended to $\widehat{\mathbf{X}}^p$ until it contains E individuals. Now from this next generation elite population $\widehat{\mathbf{X}}^p$, the next generation working population $\widehat{\mathbf{X}}^c$ is bred, running a breeding loop as in Algorithm 3.1, with tournament selection using the $\succ_n$ relation and typically tournament size $T = 2$. This process is iterated in the next generation, starting from $\widehat{\mathbf{X}}^p$ and $\widehat{\mathbf{X}}^c$. Note that NSGA-II in its original implementation uses an elite size E that is equal to the population size S, i.e. the elite population contains as many individuals as the working population. This choice is not a necessity but just a heuristic, and Deb et al. do not argue for it [DPA+02]. Keeping the elite and working populations separate constitutes a difference to Algorithm 3.1, where the elite is a subset of the working population and only $S - E$ individuals undergo the breeding process. Instead, NSGA-II determines its elite of size E from non-dominated sorting of $S + E$ individuals and stores it in an external archive, from which S new individuals are bred.

3.7.6 Global versus Sequential Optimization of SynOpSis

With NSGA-II selected as the algorithm to implement the *Optimization* stage of *SynOpSis*, two approaches to conducting this optimization can be taken, that will be explained in the following paragraphs.

Global optimization is the approach visualized in Figure 3.2: The top line of the optimization loop, starting with the pattern detector and ending with the classifier is executed as a whole before any evaluation of objective functions is done. That means parameters of the detector and classifier are optimized simultaneously, and one pass of the optimization loop involves both, detector and classifier. Consequently, one individual in this optimization consists of detector *and* classifier parameters. Computing the fitness of an individual means computing the objectives of the detector (e.g. Recall and M-Rate, cf. Section 3.5.2) and those of the classifier (e.g. Recall and Precision, cf. Section 3.6.2), and these objectives are optimized simultaneously. Doing so enables devaluation of individuals containing detector parameters that produce patterns which are hard[21] to be classified correctly: In this case, objectives measured for the classifier will have poor values, and the associated individual can easily be dominated in these objectives. This can have a positive influence on results quality in practice because it implements a feedback mechanism from the classifier back to detector parameters.

Sequential optimization can not benefit from such a positive influence: In sequential optimization, the detector and the classifier are optimized separately, with classifier optimization depending on the results of detector optimization, but without any possibility of influencing it. In contrast to the global optimization depicted in Figure 3.2, sequential optimization involves two optimization loops, each with separate evaluations of objectives. The first loop optimizes detector parameters with respect to the detector objectives. Feature extraction is not necessary. Instead, the objectives are evaluated immediately on the detected patterns,

[21] Exemplary causes making patterns hard to classify can be adversely shaped pattern representations, or image processing settings in pattern detection that exert negative effects on the subsequent computation of features.

which is possible because they relate solely to detection performance. This optimization yields a Pareto front of parameter sets for the detector, among which the best-performing one is selected using the desirability approach presented in Section 3.8. The patterns detected using this parameter set are then passed to feature extraction, and the feature-annotated patterns are the input of the second loop that optimizes classifier parameters, solely with respect to classifier objectives.

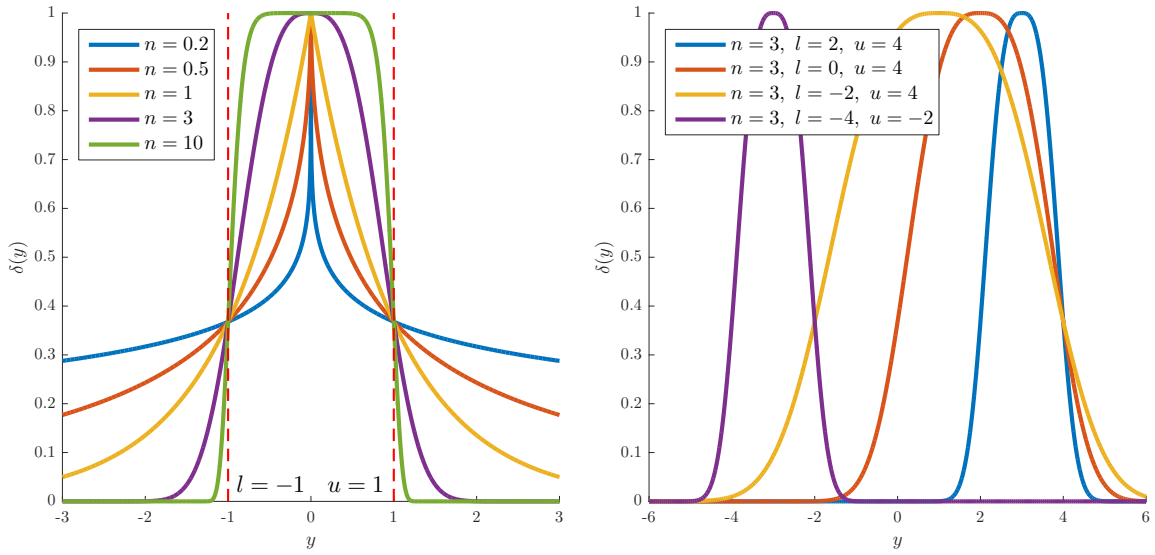
Regardless of whether the *Optimization* stage is run in a global or sequential fashion, the type of output is the same, consisting of optimized detector and classifier parameters that have been chosen from their Pareto front(s) by the desirability approach in Section 3.8 in combination with the model selection in Section 3.9. How these parameters are used in analyzing the real sensor input data is described in the context of the *Application* stage from Figure 3.2, cf. Section 3.10. The overall configuration of the MOGA as used in the *PAMONO* scenario belongs to the experiment description that can be found in the evaluation chapter, cf. Section 7.3.2.

3.8 Desirability Functions for Formalizing Expert Preferences

The concept of desirability as a formal way of expressing expert preferences was first introduced by Harrington [Har65] and has since then been used extensively in the context of multi-objective optimization [BM91; Wu04; MT06; PN06; MTT07; JK09; TM09]. Two key benefits of using desirability in multi-objective optimization are as follows:

1. **Automatic selection of the most desirable individual from a Pareto front** is enabled by the desirability approach because it allows stating preferences of expert users in a formal way: The desirability approach is a nonlinear technique for aggregating multiple objectives in a scalar number. Nonlinearity allows modeling more complex relations between objectives than is possible with linear scalarization techniques like weighted summation. Once formalized, the expert preferences can be automatically applied to a given Pareto front, yielding the single most desirable individual on the front. In this scenario, the desirability approach is used solely *after* optimization.
2. **Narrowing the search to the relevant part of the Pareto front**, on the other hand, is enabled by applying the desirability approach *during* optimization: Instead of searching across the entire Pareto front, the limited population and elite sizes are used more efficiently by focusing them in the relevant part of the front. This capability is achieved by transforming each dimension of the original objective space with a so-called Desirability Function (DF). After this transformation, individuals in undesirable parts of the front in the original objective space are likely to be dominated in the DF-transformed objective space. In turn, they are unlikely to be selected for breeding new individuals. Thus the search is directed toward the desirable regions of the front.

Section 3.8.1 provides the relevant background about Harrington's DFs, applied to normalize single objectives, while Section 3.8.2 presents Desirability Indices (DIs) as a way of aggregating DFs to scalar objectives. After these backgrounds have been established, Section 3.8.3 depicts how DFs and DIs are used in *SynOpSis* for picking a single best individual from a Pareto front and for guiding optimization. The concrete desirability setting used for *PAMONO* is part of the experiment description and hence given in Section 7.3.3.



(a) Varying n **(b)** Varying l and u

Figure 3.9: Two-Sided Harrington Desirability Functions – Examples. (a) shows Desirability Functions (DFs) with varying kurtosis parameter n and constant lower and upper specification limits $l = -1, u = 1$. (b) keeps $n = 3$, while varying l and u. Note that the peak of the DFs is attained halfway between l and u, and that changing l and u while keeping $u - l$ constant yields translates of the same function.

3.8.1 Harrington Desirability Functions

Computation of the Desirability Function (DF) for a given objective can be regarded as a normalization step: Regardless of the range occupied by an objective, and whether it is to be minimized or maximized, its DF is always between zero and one and must be maximized. Besides actual objectives, (especially soft) constraints can also be rephrased in terms of DFs, where the desirable values (close to one) correspond to the interval where a constraint is fulfilled.

SynOpSis uses Harrington's two-sided DF [Har65] for the reasons given after presenting its formal definition which follows [TW06]:

$$\delta(y) = \exp\left(-\left|\frac{2y - (u + l)}{u - l}\right|^n\right). \tag{3.7}$$

Here, $\delta(y) \in]0, 1]$ is the desirability of objective value $y \in \mathbb{R}$, and $l, u \in \mathbb{R}$ are the lower, respectively upper specification limits, which determine the interval of values for y that are deemed desirable. Finally, n is the kurtosis parameter determining the peakedness of the DF.

Figure 3.9a illustrates the kurtosis parameter n by plotting example functions obtained from varying n, while keeping $l = -1, u = 1$ constant. Increasing n makes the peak of the resulting curve flatter, resulting in a plateau close to one of nearly equally desirable corresponding values of y. Decreasing n makes the peak more spiky, narrowing the interval of fully desirable values of y, but widening the tails of the function. Figure 3.9b demonstrates the effect of changing the lower and upper specification limits l, u, while keeping n constant. The peak of the function is always located at $l + \frac{(u-l)}{2}$. Changing l and u while keeping the difference $u - l$ constant yields translates of the same function.

For *SynOpSis*, Harrington DFs were chosen because their non-zero tails give them the ability to distinguish different points below the lower specification limit (up to numerical precision). This is an advantageous property because the optimizer is pulled into the direction of improvement even for those individuals that do not meet the lower specification limit: Degrees of undesirability in undesirable individuals can be distinguished and "less undesirable" individuals are more likely to take part in breeding in the context of MOGAs. As a downside, Harrington DFs also distinguish different points above the upper specification limit, unless modified accordingly. In some contexts, e.g. if for satisfied constraints desirability should be plainly one, this is an adverse property because distinction of different degrees of constraint satisfaction is not always intended. For such cases, Derringer-Suich DFs [DS80] can be used, which are flat one for values above the upper specification limit. Note however, that these in turn, do not distinguish points below the lower specification limit. In *SynOpSis*, indifference between points above the upper specification limit is not essential because for the optimized objectives (cf. Sections 3.5.2 and 3.6.2) "better is always better".

Besides the two-sided Harrington DF from Equation (3.7), there is a one-sided version using a double exponential [TW06]. It is called one-sided because the desired interval is bounded on only one instead of two sides. In *SynOpSis*, the objectives are in fact one-sided: They are either to be minimized or maximized. However, they are bounded and thus also have a target value, which enables using two-sided DFs with the peak placed over the target value. Hence both kinds of DFs are eligible to be used in this case. The two-sided version was chosen because it provides a measure of controlling kurtosis via n, which the one-sided version does not. This additional control enables highly spiky DFs with long non-zero tails. This is useful for objectives like detector Recall: More detector Recall is always more desirable, but it must also be possible to distinguish low values of it to guide optimization into the right direction. This heuristic can very well be modeled using the two-sided Harrington DF.

3.8.2 Desirability Indices

While computing DFs is a normalization step applied separately to *multiple* objectives, computing a Desirability Index (DI) is a scalarization step, aggregating the DFs of multiple objectives into a *single* scalar number. The most commonly used function for aggregating DFs is the geometric mean [TW06], and the geometric mean DI is defined as

$$\Delta(\mathbf{y}) = \left(\prod_{i=1}^{O} \delta^i(y_i) \right)^{1/O}, \tag{3.8}$$

where O is the number of objectives, δ^i is the DF using n^i, l^i, u^i specific to the ith objective, and y_i denotes the value attained in that objective. The geometric interpretation of the geometric mean of O values is that for an O-dimensional hyperrectangle with side-lengths equal to these values, it gives the side-length of a hypercube containing the same hypervolume as the hyperrectangle. Notably, it is zero if one of the input side-lengths is zero, since the hypervolume enclosed by the hyperrectangle then becomes zero.

As a consequence, if DFs are aggregated using the geometric mean as DI, *all* objectives need to have desirable values to assign an individual a high DI. One objective with low or zero desirability devalues[22] the entire individual. Therefore, individuals from the front that

[22] This is a strong contrast to linear scalarization methods, where bad values in one objective can be compensated for by good values in another.

fail in at least one objective would, despite their Pareto-dominance, never be chosen as the best individual from the front. Another advantage of the geometric mean DI is that it enables easy identification of cases where the optimization failed to produce an individual that is reasonably strong in all objectives: Then the DI is below a certain threshold, and the failure can be reported to the user, instead of continuing analysis with possibly very bad results. Furthermore, the following important property holds for the geometric mean DI:

Lemma 3.1. *If the geometric mean DI over strictly monotonic DFs of multiple objectives is optimized as a single objective, the obtained maximum is non-dominated in the original objective space, among the individuals visited during that optimization, i.e. it resides on the approximated Pareto front in multi-objective space.*

Proof. Assume the individual with maximum DI is not on the Pareto front in the original objective space. Not being on the front means that there exists an individual on the front that dominates the individual with maximum DI. Using Definition 3.1, the dominating individual is better in at least one objective, and not worse in any other. By strict monotonicity of the employed DFs, it follows that at least one DF in the geometric mean in Equation (3.8) is larger for the dominating individual. Then, as a product grows if one of its factors grows, the geometric mean in Equation (3.8) is larger for the dominating individual, contradicting the assumption that the geometric mean of the individual with maximum DI is maximum. □

Similarly, optimizing objective DFs in a multi-objective fashion yields a Pareto front in desirability space that is non-dominated in the original objective space, if the employed DFs are strictly monotonous. An alternative to the geometric mean DI is taking the minimum among objective DFs as the DI [TW06]. This does, however, not guarantee a property analogous to Lemma 3.1.

As a conclusion from this section, arbitrary subsets of objectives and constraints can be scalarized in DIs, after or during optimization. It is possible e.g. to create a combined DI for all objectives and another one for all constraints, enabling monitoring of objective attainment and constraint violation over a large number of objectives and constraints in a single 2-D plot. Maxima with respect to strictly monotonous DFs or with respect to the geometric mean DI are non-dominated in original objective space, among the individuals created during optimization.

3.8.3 Desirability in SynOpSis

In the introduction of this section, automatic selection of the most desirable individual from a Pareto front and narrowing search to the relevant part of the Pareto front were named as the two key benefits of using the desirability approach in multi-objective optimization. *SynOpSis* makes use of both, but in different scenarios.

The "desk" scenario takes the perspective of algorithm design and problem analysis. Here, the desirability approach is applied solely *after* optimization, for automatically selecting the best individual from the obtained Pareto front. Optimization is multi-objective in the original objective space, enabling its thorough exploration and analysis of trade-offs between the objectives because the full front is searched.

The "lab" scenario is more application-oriented, taking the perspective of lab workers applying *SynOpSis* in analyzing *PAMONO* or similar data. Here the search is narrowed to the relevant part of the Pareto front by applying the desirability approach already *during* optimization, which can be done in two ways: The first way is multi-objective optimization of DFs, integrating expert preferences into searching in a subregion of the original Pareto front, located in the relevant part of objective space. The second way is single-objective optimization of the DI, searching for a single best individual that performs well in *every* constituent objective. In comparison to not applying the desirability approach, both ways, in theory, enable two benefits: Solutions of the same quality may be found with fewer evaluations, and if fitness is not yet saturated, better solutions may be found with the same number of evaluations. These benefits are important in the lab, where time is a constrained resource: Optimization must be as quick as possible, while maintaining results quality. Note that the concentration on the desirable part of the Pareto front seemingly contradicts the purpose of the diversity preservation mechanisms common in MOGAs. However, desirability mechanisms are a way of constraining diversity in a *controlled* and *target-oriented* manner. The resulting search in the desirable part of the front still benefits from diversity in that desirable part.

The concrete desirability settings used in *PAMONO* data analysis are part of the experiment description and hence given in Section 7.3.3.

3.9 Model Selection and Performance Estimation

Bergstra et al. consider manual tuning of algorithmic parameters to make them suit a given problem instance as something "more of an art than a science" and suggest that "hyper-parameter optimization should be regarded as a formal outer loop in the learning process" [BBB+11]. While referring to machine learning tasks only, their suggestion may prove beneficial in other contexts as well. *SynOpSis* explores this suggestion in the context of its consecutive pattern detection and classification task by implementing optimization of all relevant parameters as an outer loop around the approach depicted in Figure 3.2. Unleashing the full potential of parameter optimization requires mechanisms that avoid overfitting the parameters to the dataset they are optimized on [SH97]. Here **overfitting** means, that the parameters may perform well on the dataset upon which they were optimized (because they were optimized to do so) but they do not **generalize** well, i.e. their good objective values do not carry over to other datasets. Therefore, objective values that were measured on the dataset used in optimization are *not the quantities of interest*. Instead, objective values measured with respect to *unseen* datasets are considered, i.e. the **generalization performance** of the parameters is estimated. Techniques for doing so are summarized in Section 3.9.1. These techniques can be applied in parameter tuning, aiding the selection of parameters that generalize best. This process is also referred to as model selection, and is the topic of Section 3.9.2. Generalization performance is furthermore important as an estimate of the performance of the finally selected parameters, that reduces undue optimism, cf. Section 3.9.3. An example of how these mechanisms for model selection and performance estimation can be implemented is given for *PAMONO* data analysis. It belongs to the experiment description and can thus be found in Section 7.3.4. Computation of the final classifying model used in the pattern classifier depends on this implementation and is therefore detailed in Section 7.3.5.

3.9.1 Generalization Performance

In machine learning, the term **generalization performance** refers to the quality a learned predictive model attains in making predictions about unseen data. In a more general context, it can be regarded as the results quality an algorithm attains for previously unseen input, given a set of parameters. 'Unseen input' means that this input was not used in determination of the parameters, eliminating the possibility of the parameters being overfitted to it. Note that generalization performance does not relate to a specific performance metric: Any performance metric can be used since the defining property of generalization performance is simply that the metric is measured on previously unseen data. The goal of such measuring is reducing undue optimism that would arise if quality was measured on the same dataset used in determining parameters.

Various strategies for estimating generalization performance have been proposed in the literature, many of which are summarized in [HTF09] for the context of machine learning. The following list is not complete, but summarizes some of the most common strategies. They all share the idea of disjointly dividing a given ground truth-annotated dataset into a **training** set, used as input to train/optimize the algorithm upon, and a **test** set, used solely for estimating generalization performance. Presentation of all strategies follows the work by Kohavi [Koh95].

- **Holdout** simply divides the annotated data by sampling without replacement a training set from it and using the complement as the test set. Typically, the training set comprises $\frac{2}{3}$ of the overall data.
- **Random subsampling** is also known as repeated holdout: The holdout strategy is repeated K times with different random samples. This enables computation of e.g. the mean generalization performance and its standard deviation, thus providing a measure of confidence.
- **Bootstrap** sampling constructs a training set of size N by sampling with replacement from the original dataset, also of size N. The data items not used in the training (multi-)set are used as the test set. Like in random subsampling, this is repeated K times.
- **K-fold cross-validation** partitions the original data into K subsets of approximately equal size. The algorithm is trained/optimized K times, each time using another of the subsets as test set and the union of the remaining $K-1$ sets for training. Mean and standard deviation of estimated generalization performance can be computed over the K folds. For classification algorithms, bias and variance of this estimate can be improved in most real-world tasks [Koh95], by creating the K subsets with **stratified** sampling. This means the data is sampled such that the label proportions in each subset are approximately equal to the label proportion in the original dataset. Stratification can also be applied in the previously listed methods. If the K in K-fold cross-validation equals the size N of the input dataset, the procedure is referred to as leave-one-out cross-validation.

Any of these methods can be used as the strategy for generating the training and test sets that are required for the model selection, respectively performance estimation procedures presented in the next two sections.

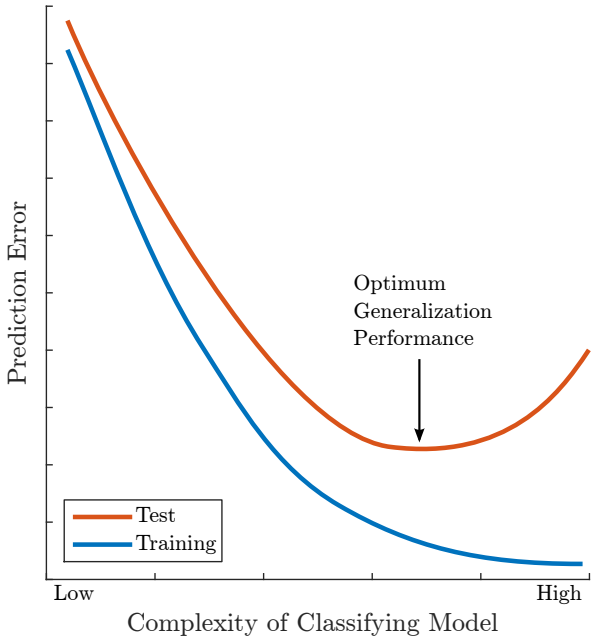


Figure 3.10: Generalization Performance over Complexity of Classifying Model. Prediction error incurred on the training set monotonously decreases with increasing complexity of the classifying model because more complex models can overfit the training data more, fully memorizing them in the extreme case. Test set error exhibits a minimum at a certain complexity. Increasing complexity beyond that minimum creates models that capture spurious peculiarities of the training set that do not generalize to the test set. Model selection for classifying models aims at finding the parameters producing the classifying model with optimum complexity. Figure adapted from [HTF09].

3.9.2 Model Selection

'Model selection' is a term that originates in statistics and machine learning, where it denotes the selection of a statistical/predictive model from a set of candidate models [HTF09]. The underlying technique of estimating generalization performance to avoid overfitting the training data is as well beneficial and applicable in optimizing algorithmic parameters. For this context, the term 'model' in 'model selection' can be regarded as denoting a parameter set configuring an algorithm.

While this term at first sight appears oddly defined, it maintains consistency with machine learning terminology, where optimization of the parameters of learning algorithms is typically carried out via model selection: A parametric learning algorithm is used to create a set of candidate predictive models, given a set of training data and varying parameter sets. Model selection then means selecting that parameter set which exhibits the best estimated generalization performance on the test set. This parameter set is then used to learn the final predictive model from the *entire* available data (training and test set). Therefore, in this context, the entity that is selected during model selection is a parameter set configuring an algorithm, and this concept can be transferred to optimizing parameter sets of other algorithms, e.g. the pattern detector used in *SynOpSis*.

The key purpose of model selection lies in reducing the risk of overfitting the training data by selecting a model with respect to estimates of its generalization performance. This is beneficial in *any* parameter optimization task. Optimizing the parameters of a supervised

classifier, however, provides a very intuitive notion of the merit of model selection, which will be presented now as an example. In this context it is important to strictly distinguish between the model, defined as the parameter set of the supervised classifier, and the classifying model, defined as the predictive model obtained by running the learning procedure of the supervised classifier, using training data and the given parameter set as inputs, cf. also Figure 3.5. The parameter set of a supervised classifier controls, amongst others, the complexity of the resulting classifying model and thus its ability to overfit the training data: The more complex the classifying model, the more training data it can memorize. One extreme example is k-Nearest Neighbors (k-NN) [HTF09] for $k = 1$, where the classifying model consists of all training data points (lazy learning), and each new point is predicted to belong to the class of its nearest neighbor in the training set. When applied to the training set itself, this classifying model yields perfect results and constitutes a worst case of overfitting. Less complex classifying models must abstract further from the training data which is usually beneficial for generalization performance. Figure 3.10 demonstrates this: Prediction error for the training set monotonously decreases with increasing complexity of the classifying model, while for the test set, there is an optimum point. The divergence of the curves demonstrates that training error underestimates test error, especially after complexity increases beyond the optimum point. Model selection aims at finding the parameters producing the classifying model with the complexity that optimizes generalization performance. For these parameters, the classifying model abstracts far enough from the training set to generalize well, but not as far as to become too simplistic to capture the concepts in the data.

While being demonstrated here for parameters of supervised classifiers, like the pattern classifier in *SynOpSis*, the phenomenon of overfitting parameters to the training data applies to optimizing its pattern detector as well: In this context, parameters can overfit the training data by adapting specifically to spurious peculiarities in the training set that do not appear in the test set and hence do not generalize. Here the relation to complexity is less obvious than in classifying models, but it still exists: For example if a parameter needs to have a certain value to work well on the training set, but is irrelevant for test set performance, then a parameter set *with that specific* value can be considered more complex than one *with any different* value. Algorithms with more parameters allow for more such cases and generally for more overfitting, making model selection more important, the more parameters are to be optimized.

Summing up, model selection for supervised classifiers and other algorithms optimizing a loss function with respect to ground truth-annotated data is carried out as follows: The available ground truth-annotated input data is divided into training and test sets, using e.g. one of the division strategies presented in the previous section. A set of candidate models, i.e. parameter sets, is created by optimizing the loss function on the training data. For each such model, its generalization performance in the measures of interest (e.g. Recall, Precision, ...) is estimated with respect to test data, each measure yielding either a single value (e.g. holdout), or a mean and its standard deviation (e.g. random subsampling, bootstrap, cross-validation). Each measure assigns an estimated generalization performance to a parameter set, similar to the test curve in Figure 3.10, but without subsuming the parameter space in a single complexity axis. Model selection then outputs the parameter set yielding the optimum estimated generalization performance, i.e. the best test set performance. In case of multiple performance measures, the desirability approach from Section 3.8 can be used to obtain a single combined measure, thus resolving Pareto-incomparability. By being computed

on data not used in parameter optimization, the obtained measures do not involve the undue optimism that would be caused by measuring on training data, cf. the training curve in Figure 3.10. Therefore, model selection yields a model that generalizes well to unseen data, instead of selecting the one that best (over)fits the training data.

3.9.3 Performance Estimation

Performance estimation, also denoted as 'model assessment' [HTF09], denotes the process of estimating the performance a model will attain, when used on the original data which it is supposed to handle. Performance estimation *without* prior model selection can simply be done analogously to model selection, by estimating generalization performance, e.g. with one of the strategies from Section 3.9.1. The only difference is that in performance estimation, this process needs to be run only once, because there is only one model.

Performance estimation *with* prior model selection is slightly more complex, for the following reason: Generalization performance of the finally selected model as computed during model selection should not be taken as an estimate of the performance that model attains on further unseen data because the model was *selected to optimize* performance on this test set. This act of selection is again an optimization, just like the optimization that created the candidate models for model selection. Hence the situation is exactly the same as in model selection, taken one tier higher: The generalization performance computed in model selection is an unduly optimistic estimate of generalization performance with respect to further unseen data. The optimism arises because selecting a model with respect to performance attained on a certain dataset, constitutes a transfer of information about that dataset, encoded in model choice: The model/parameters are selected as to optimize performance on that dataset and thus form a channel for information transfer. This observation is due to Scheffer and Herbrich [SH97] (cf. their Figure 1) and was made in the context of machine learning algorithms. The issue, however, affects any scenario of model selection for parameter tuning, followed by performance estimation. The effect aggravates, the more parameters there are to be tuned because increasing numbers of parameters "widen the channel" across which the information transfer can occur. For example in *PAMONO* data analysis this is particularly severe because the detector alone has 28 parameters to be tuned, cf. Section 5.7.

As for being an iterated version of the issue that was already encountered in model selection, the same methods can be applied for resolving it: Again, generalization performance should be measured, but this time with respect to the process *including* model selection. This introduces the need for a further dataset that is disjoint to the datasets used in training and model selection. Following Hastie, Tibshirani, and Friedman [HTF09], the following terminology is used:

Terminology 3.3. *The **training** set denotes the data used as input in optimizing the parameter set of an algorithm. The **validation** set denotes the data used in model selection, i.e. in choosing the parameters to be finally used. The **test** set denotes the data used in performance estimation, i.e. in estimating how well the finally selected parameter set will perform on further unseen data.*

Note that this definition of the term 'test set' contrasts with the typical usage of the term in the context of model selection. Again, data division into these sets can be carried out using e.g. one of the strategies in Section 3.9.1. Conducting performance estimation after model

selection, by computing generalization performance with respect to the previously untouched test set, avoids incurring the undue optimism described by Scheffer and Herbrich [SH97].

As can be seen from this and the previous section, model selection and performance estimation exhibit a number of design choices, that should be taken in consideration of the intended application scenario. The design choices are e.g. data division strategy and number of repetitions or folds, if applicable. This needs to be done for both, model selection and performance estimation. For *PAMONO* data analysis using *SynOpSis*, the design choices are explained in the experiment description in Section 7.3.4. As computation of the final classifying model depends on these choices, it is described after that, in Section 7.3.5.

3.10 Summary of SynOpSis and Application Stage

The *Application* stage is the last stage of *SynOpSis* and is illustrated at the bottom of Figure 3.2. In contrast to the offline *Synthesis* stage and *Optimization* stage, it can be executed in real-time, provided its components support this. The pattern detector, feature extraction and pattern classifier used in the context of *PAMONO* data analysis are real-time capable [LST+13a; LST+13b], cf. Section 7.5.8.

The following description will very briefly recap the way from optimizing parameters to applying them to the real sensor input data, along the flow of data in Figure 3.2. References to the sections that contain the respective details are given in footnotes. Two types of references are distinguished: *SynOpSis* references provide abstract and general depictions of methods used in *SynOpSis*, while *PAMONO* references describe concrete and application-specific implementations of the respective methods, as they are used in *PAMONO* data analysis.

After the *Synthesis* stage[23] generated ground truth-annotated data, the *Optimization* stage[24] finds a Pareto front of non-dominated parameter sets for the pattern detector[25] and classifier[26], using a Multi-Objective Genetic Algorithm (MOGA)[27]. A model selection[28] with respect to an unseen validation set is conducted to pick the final parameter set for detector and classifier from the Pareto front. Pareto-incomparability of objective vectors measured during model selection is resolved using Desirability Indices (DIs)[29]. Performance estimation[30] for the finally chosen parameters is carried out with respect to an unseen test set.

After the final parameters have been determined, they are passed as inputs to the *Application* stage, as illustrated in Figure 3.2. Here they are used in analyzing the real sensor input data, which is done the same way as the synthetic data was analyzed during the *Optimization* stage. Therefore, the *Application* stage is implemented simply by replicating a subset of the *Optimization* stage, as can be seen from comparing the *Application* stage to the top row of the *Optimization* stage in Figure 3.2. Now, the pattern detector is run on the real sensor input data, using its optimized parameter set. The output consists of unclassified patterns from which features are extracted for classification. The feature-

[23] *SynOpSis*: Section 3.4, *PAMONO*: Chapter 4
[24] *SynOpSis*: Section 3.7, *PAMONO*: Section 7.3.2
[25] *SynOpSis*: Section 3.5, *PAMONO*: Chapter 5
[26] *SynOpSis*: Section 3.6, *PAMONO*: Chapter 6
[27] *SynOpSis*: Section 3.7.4, *PAMONO*: Section 7.3.2
[28] *SynOpSis*: Section 3.9.2, *PAMONO*: Section 7.3.4
[29] *SynOpSis*: Section 3.8, *PAMONO*: Section 7.3.3
[30] *SynOpSis*: Section 3.9.3, *PAMONO*: Section 7.3.4

annotated patterns are passed to the pattern classifier, which applies a classifying model[31] that was learned beforehand, given the optimized classifier parameters and using the *entire* available ground truth-annotated data for training (the entire available ground truth-annotated data is constituted by the synthetic training, validation and test set; as the classifying model does not depend on any of the inputs of the *Application* stage, it can be learned beforehand to attain real-time-capability). The classifying model is applied to classify the patterns detected in the real data, and the classified patterns are output by the *Application* stage. Estimates of the quality of this detection and classification result are output during performance estimation.

[31] *SynOpSis*: Section 3.6, *PAMONO*: Section 7.3.5

Chapter 4

Synthesis Stage for PAMONO

Contents

As discussed in Chapter 3, synthesis is a crucial component of *SynOpSis*. While Section 3.4 abstractly discussed the properties required for a signal model to be used in synthesis, and the role of synthesis in *SynOpSis*, this chapter concretely describes how to generate synthetic data in the context of the *PAMONO* sensor. After the introduction with a discussion of related work in Section 4.1, a signal model for *PAMONO* is proposed in Section 4.2. Application of this model is discussed in Section 4.3, involving a specialized experimental protocol for *PAMONO* measurements in Section 4.3.1, and the computation of synthetic *PAMONO* imagery in Section 4.3.2. Conclusions are drawn in Section 4.4. The depictions given in the entire chapter are a more detailed version of the work in [SLW+14], embedding it into the context of a more advanced version of *SynOpSis* than was used in that paper.

4.1 Introduction

Including a *Synthesis* stage in the *SynOpSis* approach enables it to benefit from ready availability of large amounts of ground truth detection and classification results, to be used as training data in the *Optimization* stage, as well as for model selection and performance estimation. Being able to generate large amounts of ground truth fast and with little manual effort, enables keeping the datasets used in each of these three contexts disjoint, which avoids overfitting and optimism in the results. Ground truth-annotated synthetic data can be easily produced without the need for a user to manually segment and classify this large amount of data: A small set of exemplary patterns suffices to seed synthesis. Details on the data division strategy for creating multiple disjoint datasets for optimization, model selection and performance estimation are covered in the experiment description in Section 7.3.4, while this chapter presents the signal model in general, cf. Section 4.2, along with a procedure for generating *one* synthetic dataset, cf. Section 4.3.

As this procedure implements the abstract *Synthesis* stage from Figure 3.2, its output accordingly consists of synthetic images, annotated with ground truth pattern locations and their ground truth classification. For the concrete *PAMONO* scenario, this *Synthesis* stage

is resolved in more detail in Figure 4.2, which is explained throughout this chapter. The *PAMONO* application is about detecting patterns in the data and classifying them as being or not being related to nano-objects attaching to the sensor surface. Knowing ground truth pattern locations and their classification serves to make the objective functions[1] used in the *Optimization* stage[2] automatically evaluable, allowing to assess the quality of the algorithmic parameters undergoing optimization. This way, the parameters can automatically adapt to changing experimental setups of the *PAMONO* sensor, and an according classifying model can be learned. Furthermore, generalization performance can be automatically estimated in the context of model selection and performance estimation[3].

The amount of manual segmentation effort required for initializing the *Synthesis* stage is considerably smaller than for manually producing a sufficient amount of ground truth-annotated real data, as required e.g. by [HBR+08; PKC09; HBR+12]. Han et al. [HBR+08] require about 500 to 1300 examples per class for training a supervised classifier, while synthesis in *SynOpSis* requires only a few[4] representative target pattern examples. These suffice to synthesize much larger ground truth-annotated datasets.

Related Work

Physical simulation of the processes interacting with a sensor, along with simulation of the sensor itself provides a way of creating ground truth-annotated synthetic sensor data. An example from this category is the work by Majumdar et al. [MMB+05], who use Monte Carlo (MC) physics simulations to create training data for distinguishing γ-ray-initiated atmospheric light showers from those caused by hadrons. This involves, amongst others, simulating both kinds of light showers, light scattering in the atmosphere as the medium, and simulating the mirrors and photomultiplier electronics of the so-called Major Atmospheric Gamma Imaging Cherenkov Telescopes (MAGIC), which are the employed sensors.

A physical simulation-based method closer to the *PAMONO* context is presented in the work by Wang et al. [WSP+10]: Their field of application is a sensor similar to *PAMONO* and as well capable of detecting biological viruses. Synthesis of the sensor signal is carried out by simulating wave propagation on the sensor surface using the COMSOL Multiphysics software [COM15]. This simulation provides idealized appearances of the viruses on the surface, and it does not capture adverse effects like noise, artifacts and background signal. Hence the simulated data can not be used in optimizing a detector or training a classifying model for real data.

Real-data-based synthesis is an approach capable of capturing adverse components corrupting sensor signals. For example Learned-Miller [Lea06] creates generative image models by firstly transforming a set of input images (e.g. handwritten digits) such as to remove affine variability between them, and then modeling the latent (wanted) images and the remaining variability (nuisance variables) separately, which can be used to synthesize new data for training. Corruptive signal components can not only be captured and generated, but also analyzed in terms of the nuisance variables that are part of the signal model.

[1] Detector objectives are listed in Section 3.5.2, classifier objectives are listed in Section 3.6.2.

[2] The *Optimization* stage is described in Section 3.7.

[3] Both are explained in Section 3.9.

[4] In the evaluation in the context of *PAMONO*, 20 target patterns were manually segmented for each synthetic dataset, cf. Section 7.3.4.

Shotton et al. [SSK+13] utilize computer graphics to render synthetic depth images mimicking those captured by the Microsoft® Kinect® sensor [Zha12]. These are used as training data for human body part labeling in the context of human pose recognition. Real data enters into this process in terms of motion capture data, aimed at covering the diverse range of human poses. They report that learning from synthetic training data provides high accuracy on real test data in their application case. Furthermore, they found that having large amounts of training data (cheaply available via synthesis) is the decisive factor in attaining this high accuracy.

The model for the *PAMONO* sensor as presented now [SLW+14] aims at delivering similar benefits to *PAMONO* data analysis. It uses physical modeling only on a high level, describing image formation on the sensor in terms of signal components and their composition, cf. Section 4.2. Computation of synthetic imagery is, however, driven by real sensor data, ensuring that the output data captures even small changes in the physical parameters of the sensor setup, cf. Section 4.3.

4.2 A Signal Model for the PAMONO Sensor

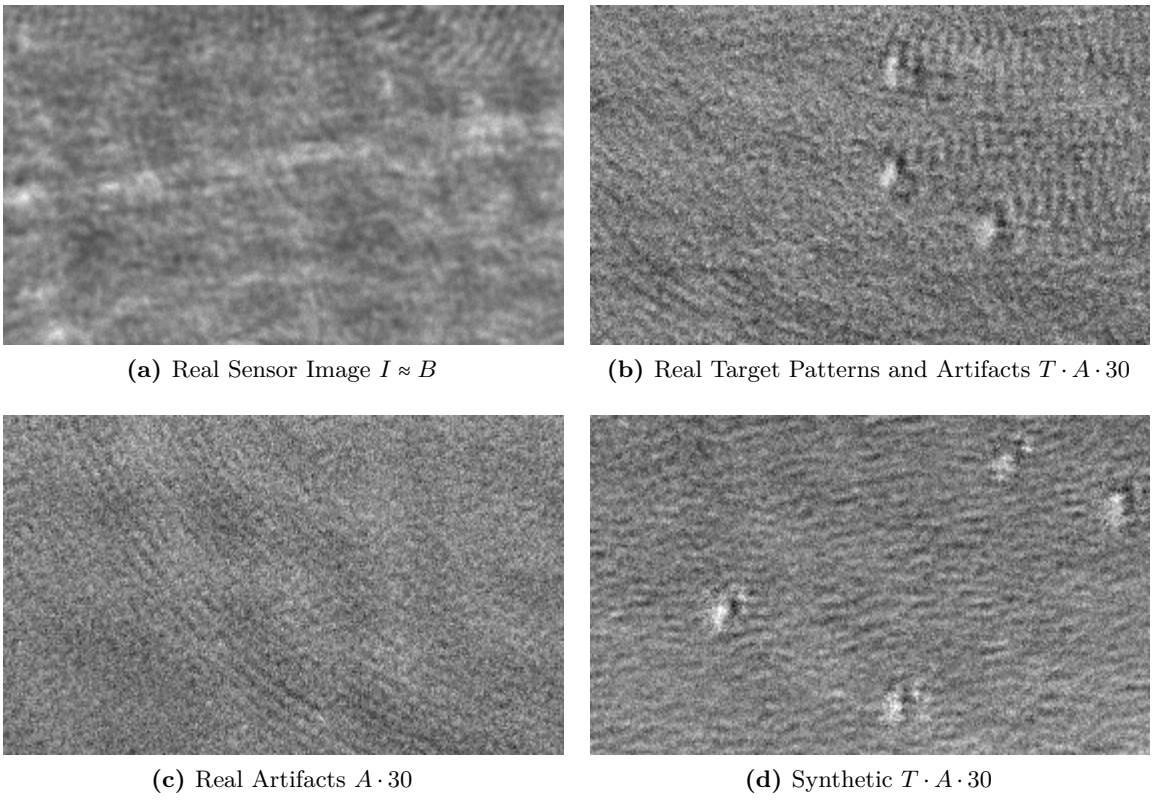
Image formation on the Charge-Coupled Device (CCD) sensor in *PAMONO* (cf. Figure 2.2 for the sensor setup) can be modeled by regarding the involved signal components and their composition. For being a time series of images, *PAMONO* data is a spatiotemporal signal with two spatial dimensions x, y and one temporal dimension t. It is composed of a background signal B, the target patterns signal T caused by nano-objects attaching to the sensor surface, an artifacts signal A with nuisance structures, and sensor noise N. Signal composition is modeled by the following equation [SLW+14]:

$$I(x, y, t) = B(x, y) \cdot (T \cdot A)(x, y, t) + N(x, y, t). \tag{4.1}$$

I is the spatiotemporal intensity signal as recorded by the CCD sensor, cf. Figure 4.1a for an example. The intensities in I are dominated to a large extent by the background component B, which remains constant over time and is thus only indexed by (x, y). B is an image of the gold surface of the sensor, along with interference patterns arising there, both of which do not change within one measurement.

The desired signal of the measurement is the target patterns signal T, which is multiplied with B. The target patterns in T are due to the nano-object adhesions to the sensor surface and serve as their indirect proof. Every signal component other than T is an impediment to the analysis process. T disperses with low amplitudes around 1, i.e. around the neutral element of multiplication. In lighter areas like those affected by the upward step functions[5] in the *central* part of a nano-object adhesion, T is greater than 1. In darker areas like the downward step functions in the *outer* part of a nano-object adhesion, T is smaller than 1. Multiplicative composition of the target patterns signal T with the background B models the proportionality between the magnitude of the Surface Plasmon Resonance (SPR)-based effects causing T and the local intensity observed in B, cf. Equation (3) in [ZSS+17]. This proportionality means

[5]The role of step functions in *PAMONO* images as an indirect proof for the adhesion of a nano-object to the sensor surface was explained in more detail in the context of the physics behind the *PAMONO* sensor, cf. Section 2.2.



(a) Real Sensor Image $I \approx B$

(b) Real Target Patterns and Artifacts $T \cdot A \cdot 30$

(c) Real Artifacts $A \cdot 30$

(d) Synthetic $T \cdot A \cdot 30$

Figure 4.1: Components of the PAMONO Signal Model – Illustration. Examples for the components of Equation (4.1) are shown. (a) is a real image I as measured by the *PAMONO* sensor. It approximately equals the background component B which dominates the intensities of I. (b) shows the same real image, after approximate removal of the background B and noise N, leaving the product of the target patterns component T with the artifacts component A. Three nano-object adhesions in T are revealed, corrupted by wave-like artifacts in A and residual noise. For comparison, (c) shows a real image of only the artifacts A and residual noise, without any target patterns. (d) show the same components as (b), i.e. target patterns, artifacts and residual noise. The difference is that (b) shows real data, while (d) was created synthetically, using the signal model from Equation (4.1). Note that the signals in (b)–(d) were amplified by a factor of 30 in comparison to (a), cf. text for more details.

that, with all other things held constant, the SPR effect has lower amplitude for nano-objects attaching in areas of darker background B, than for those attaching in areas of brighter background B. Dividing by B to eliminate the multiplicative influence of the background (cf. Section 5.2) reveals the linear relationship between nano-object diameter and signal intensity, cf. Figure 4c in [ZKG+10].

The artifacts signal A shares the multiplicative nature of T because it is due to the same physical effect (SPR). Its intensity range also disperses with low amplitudes around the neutral element 1 of multiplication. However, the artifacts in A are an undesired nuisance signal, containing everything impeding nano-object detection. This also includes any departure of the background signal B from the constancy assumption. Figure 4.1b shows an approximation to the product $T \cdot A$, obtained by applying background elimination (to remove B, cf. Section 5.2) and denoising (to remove the noise N, cf. Section 5.3) to the real sensor image I from Figure 4.1a. Doing so reveals three nano-object adhesions as part of T, corrupted by wave-like

artifacts as part of A, plus residual noise that survived the denoising procedure. Note that the intensity dominance of B in I is strong enough to render the $T \cdot A$ signal from Figure 4.1b visually imperceptible in Figure 4.1a. Or stated the other way around: The effect of amplitude change exerted by the $T \cdot A$ factor on the background signal B is considerably smaller than B itself. Therefore, signal intensities in Figure 4.1b have been amplified by a factor of 30 in comparison to Figure 4.1a, for visualization purposes. The low amplitude of the desired signal T is the reason why using a 12-bit CCD is crucial for *PAMONO*: As the desired signal resides in the lesser significant bits, a high resolution intensity range is required to avoid its falling victim to quantization. Figure 4.1c (also amplified by factor 30 in comparison to Figure 4.1a) shows *only* the artifacts component A and the residual noise, without any target patterns. The image was recorded before any nano-objects were inserted into the flow cell of the sensor. This artifacts signal is the main source of False Positive (FP) detector responses. Separating T from A on the pixel level is an ill-posed inverse problem because only an estimate of their product is available for real data. These two facts in conjunction motivate using the pattern classifier in Chapter 6, which separates T from A on the pattern level, by classifying patterns as being caused by either T (target) or A (non-target), using local intensity-, shape- and other statistical features.

The noise component N in Equation (4.1) is modeled additively because it mainly consists of the additive Gaussian readout noise incurred by the CCD [FM06]. Besides that, there is signal-dependent shot noise arising from photon statistics [BB00], which is not modeled separately because the amount of available light is a controllable physical sensor parameter, and it is always selected to be large enough to make shot noise negligible, cf. the "Diode (Light Source)" caption in Section 7.2.1. The nature of the noise is assumed constant over time because Gaussian readout noise depends on the CCD settings, and shot noise depends on the available light, both of which are kept constant within one measurement.

Finally, Figure 4.1d (also amplified by factor 30 in comparison to Figure 4.1a) shows synthetic data, obtained by applying the signal model from Equation (4.1) as according to the subsequent Section 4.3. Like with Figure 4.1b, the background B and noise N were suppressed to obtain an approximation of $T \cdot A$. This reveals four nano-object adhesions, along with artifacts similar to those in Figure 4.1b, demonstrating the capability of the model to mimic real data.

4.3 Applying the Model

The central requirement for synthetic data in *SynOpSis* is its ability to capture the properties of real sensor data with regard to two tasks: Firstly, algorithmic parameters that work well on the synthetic data should also work well on the real data. Secondly, a classifying model learned from synthetic data should generalize well to real data. One way of obtaining such synthetic data is to let real data drive the synthesis process, as will be described now. The components of the signal model from Equation (4.1) are captured as explained in the experimental protocol in Section 4.3.1 and assembled to form the final synthetic signal, as described in Section 4.3.2.

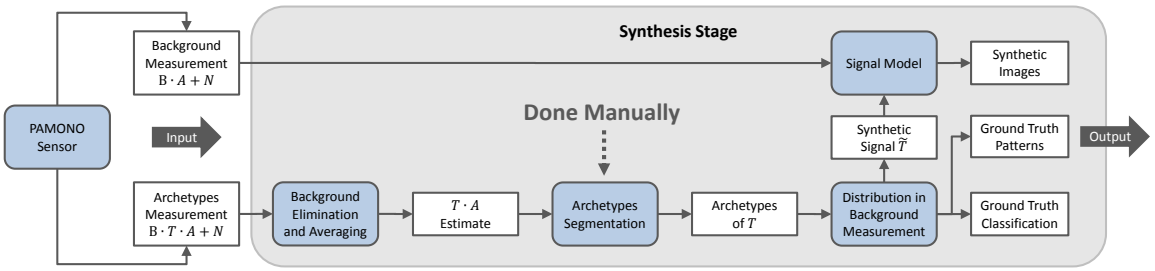


Figure 4.2: Synthesis Stage for PAMONO. A background measurement provides a real signal $B \cdot A + N$, composed of background B, artifacts A and noise N. This measurement is done before any nano-objects have been inserted to the flow cell of the *PAMONO* sensor. After that, a positive sample of nano-objects is inserted, and an archetypes measurement is conducted, yielding archetypal target patterns which are manually segmented by the user. From these, a synthetic target patterns signal $\widetilde{T}$ is generated, for which ground truth is known. $\widetilde{T}$ is combined with the background measurement as according to the signal model from Equation (4.1), and the synthetic images, along with the ground truth constitute the output of the *Synthesis* stage.

4.3.1 Experimental Protocol

Computation of the *PAMONO* signal model from Equation (4.1) builds on four components to create a synthetic time series of sensor images I. These are determined using real empirical data. Three of the four components, namely background B, artifacts A and noise N can be measured in composition, by letting the sensor record a series $I^{T=1}$ of images prior to inserting the nano-objects sample into the flow cell. This corresponds to a target patterns signal T that is constantly 1, as indicated by the superscript. The resulting time series of images and the act of recording it is in the following denoted as the **background measurement**.

Empirical data for the fourth component, namely the target patterns signal T, can be approximated by the following procedure: After the background measurement has been done, a positive sample of the nano-objects to be detected is injected into the flow cell of the sensor. The resulting signal and the act of recording it is called the **archetypes measurement**. It serves to determine archetypal instances of target patterns that are used in creating a synthetic target patterns signal $\widetilde{T}$. Determining target pattern archetypes is done interactively: The user manually segments a small, representative amount of exemplary target patterns, i.e. nano-object adhesions, in the archetypes measurement, by delineating them with polygons. To enable this, the background must be eliminated and noise removed, which is done by applying the averaging background elimination technique from Section 5.2.1 which is briefly summarized here, stating the concrete parameters that are suitable in segmenting archetypes for synthesis: An estimate of the time-dependent, low-amplitude $T \cdot A$ components in I is computed by removing the temporally constant background B and the time-dependent noise N. This is done by sliding two windows along the temporal axis. Each window is 40 images long, and the windows are 20 images apart. The average of the images in the "earlier" window is taken as an estimate of the constant background (incorporating earlier temporally variant information), while the average of the "later" window is taken as an estimate of the temporally variant $T \cdot A$ component, multiplied with the constant background. The $T \cdot A$ estimate is extracted by dividing the later average by the earlier one, revealing the changes that occurred since then. Note that averaging 40 images in both windows eliminates much of the noise component N, thus the additional denoising techniques from Section 5.3 are not applied in preparing images for archetype segmentation. To make the $T \cdot A$ estimate

discernible for the user, it is amplified by a user-selectable factor, making the result look e.g. like Figure 4.1b. In this type of data, the user manually segments a small and representative amount of archetypal target patterns. This manual segmentation *spatially* separates T from the artifacts A, and the obtained target pattern archetypes are used in creating a synthetic target patterns signal $\widetilde{T}$. In order to do so, the **representation of an archetype** consists not only of the polygon segmented by the user, but also of the local intensities in the $T \cdot A$ estimate, that are observed in and around that polygon. These intensities do not undergo any amplification because they are utilized in synthesizing $\widetilde{T}$, as described in the next section.

Note that unlike the target patterns signal T, the artifacts signal A can not be captured adequately by archetypes due to its very diverse nature: Any structure in the data that is not a target pattern may be regarded as an artifact. Therefore it is more efficient to identify artifacts as "not being a target pattern", by comparing ground truth target patterns to detector results as described in Section 5.8. This also saves the need for manual segmentation of artifacts.

Summarizing the procedures above and embedding them into the *SynOpSis* workflow from Figure 3.2, the experimental protocol for conducting a run of multiple *PAMONO* measurements with a new sensor setup is as follows:

1. A background measurement is conducted by running the sensor before any nano-objects have been inserted into the flow cell. This yields $I^{T=1} = B \cdot A + N$, i.e. real background, artifacts and noise. The target pattern component T is neutral, i.e. constantly 1. The background measurement occupies the top row in Figure 4.2.
2. An archetypes measurement is recorded after a first positive sample with the nano-objects to be detected has been inserted into the flow cell of the sensor. Background elimination yields an estimate of $T \cdot A$, like in Figure 4.1b, and the user manually segments archetypal target patterns. This process occupies the left side of the bottom row in Figure 4.2.
3. Synthesis assembles the background measurement with the archetypal target patterns via the signal model from Equation (4.1) and outputs synthetic images, along with ground truth pattern locations and classification, which is described in detail in the following Section 4.3.2 and schematically visualized in the remainder of Figure 4.2.
4. The *Optimization* stage and the rest of the offline part of *SynOpSis* are carried out, cf. top part of Figure 3.2.
5. Utilizing the algorithmic parameters and the classifying model determined in the offline part, the *PAMONO* measurement runs of interest are conducted. The measured data is analyzed in real-time via the *Application* stage in the bottom part of Figure 3.2, as described in Section 3.10. This can be done until experimental conditions change again, after which a new run of this protocol might be required in order to ensure optimized quality of analysis results.

4.3.2 Computation

After the background and archetypes measurement and the manual segmentation of archetypal target patterns (steps 1 and 2 of the experimental protocol in the previous section) have been conducted, all data required for computing synthesis is available. The pivotal point for this computation is the signal model from Equation (4.1). Before applying it, a synthetic target patterns signal $\widetilde{T}$ with known ground truth must be generated, which is done as follows:

Uniformly random draws with replacement from the set of manually segmented archetypes are conducted, and each such archetype is placed at a uniformly random spatiotemporal location in the domain (x, y, t) of the background measurement $I^{T=1}$. This is done until the desired density of target patterns has been reached. Placing an archetype at position (x_c, y_c, t_c) in the domain of $I^{T=1}$ means that the centroid of its manually segmented polygon is placed at coordinate (x_c, y_c, t_c) in a volume of ones, the size of $I^{T=1}$. Then the intensities of the archetype, which disperse with low amplitudes around 1, are multiplied with that volume, at their respective locations around (x_c, y_c, t_c). This is repeated for all subsequent images in the volume, i.e. for all $t > t_c$ because once a nano-object attaches to the sensor surface, it remains fixated there, and its appearance does not change. In multiplying the intensities with the volume, a feathering-like weighting [Sze06] is applied to fade out intensities outside the segmented polygon, while its insides are kept as they are. Then the next archetype is multiplied with the *same* volume, until it is filled with target patterns at the desired density. The resulting volume is the synthetic target patterns signal $\widetilde{T}$, for which ground truth is known in terms of the manually segmented polygons at their randomly drawn positions. $\widetilde{T}$ is then multiplied with the background measurement $I^{T=1}$, corrupting target patterns originating from the same archetype with different background, artifacts and noise, thus increasing variation. This composition yields the synthetic image sequence $\widetilde{I}$ with nano-objects at known locations and no other nano-objects because $I^{T=1}$ contains none. The entire process described here occupies the right part of the scheme in Figure 4.2.

Note that multiplying $I^{T=1}$ with $\widetilde{T}$ means multiplying $\widetilde{T}$ with the two summands constituting $I^{T=1}$, as according to Equation (4.1). In the first summand, $\widetilde{T}$ assumes the role of the target patterns signal T in full agreement with the equation: $\widetilde{T}$ is multiplied with the undesired artifacts component A and with the temporally constant background signal B. However, the noise summand N will also be multiplied with $\widetilde{T}$, hence violating the model. This is ignored because it is negligible for two reasons: firstly because $\widetilde{T}$ disperses around 1 with very low amplitudes and secondly because of the noise nature of N.

The result of this real-data-driven approximation to Equation (4.1) is a synthetic dataset containing real background, artifacts and noise, along with archetypal synthetic target patterns that are created from real data. It captures the appearance of fully real data, as can be seen from comparing the real $T \cdot A$ estimate in Figure 4.1b to the synthetic one in Figure 4.1d. Incorporating a real artifacts signal renders the manual segmentation of archetypal artifacts unnecessary because all detections that do not match a synthetic target pattern can be considered as belonging to the non-target class caused by artifacts or noise. The synthetic image sequence $\widetilde{I}$, along with the ground truth target pattern locations and implicit two-class classification are utilized by the remainder of *SynOpSis*.

4.4 Conclusion

A signal model for the *PAMONO* sensor was presented and computed using empirical data from a specialized experimental protocol. Applying this signal model forwardly generates synthetic *PAMONO* imagery, annotated with ground truth, which can be used to automatically evaluate objective functions within *SynOpSis*. Applying the signal model backwardly, in this case by making efforts to extract the signal of interest T from the observed composite signal I, provides guidance for designing further processing steps. Specialized methods can be targeted at handling the different components of I modeled in Equation (4.1): The background component

B is targeted within the pattern detector, by the background elimination techniques discussed in Section 5.2. The noise component N is addressed by the denoising strategy presented in Section 5.3, which is as well part of the detector. The separation of the target patterns in T from the artifacts in A occurs on the pattern level and is the task of the pattern classifier covered in Chapter 6.

Evaluation of the *Synthesis* stage for *PAMONO* is conducted by measuring how well the algorithmic parameters and classifying model determined from synthetic data generalize to the real data to be analyzed. As a consequence, this evaluation relies on the full *SynOpSis* approach, including the pattern detector and classifier. It is thus given in the overall evaluation of *SynOpSis* in Chapter 7, after the pattern detector and classifier have been presented. The cross-validation strategy in which *SynOpSis* is run also encompasses the *Synthesis* stage. Details on how synthesis is applied within cross-validation can be found in Section 7.3.4, describing the experimental settings relating to cross-validation.

Chapter 5

Pattern Detector for PAMONO

Contents

As depicted in Figure 3.1, the task of finding nano-objects in a given time series of *PAMONO* sensor images is divided into a detection and a classification part. The pattern detector to be presented in this chapter is a concrete realization of the abstract pattern detector displayed in Figure 3.3. It has been developed specifically for the *PAMONO* application scenario and exhibits the same interface in terms of input and output as the abstract pattern detector. Therefore it can implement the two instances of the pattern detector in the *SynOpSis* overview, depicted in Figure 3.2.

This *PAMONO* pattern detector is real-time-capable and runs on the Graphics Processing Unit (GPU). As discussed in the context of the abstract pattern detector in Section 3.5, one of the goals of detection is not missing any target patterns in the data, i.e. not missing any nano-objects attaching to the sensor surface in the context of *PAMONO*. This goal is pursued by using Recall as one objective in optimizing detector parameters. Doing so comes at the possible cost of incurring many spurious detector responses, i.e. detections of non-target patterns. These are sorted out later, by the pattern classifier to be presented in Chapter 6. Hence the patterns output by the detector are to be regarded as *candidates* for being caused by actual nano-objects attaching to the sensor surface.

The central challenges arising in finding all such candidates are due to the adverse properties of the time series of input images provided by the *PAMONO* sensor (cf. Section 4.2): The low amplitude desired signal caused by the target patterns is modulated upon an artifacts signal of similar amplitude and a background signal of considerably larger amplitude. Additionally, there is the inevitable noise of recording. These challenges are addressed by specific modules, which in total constitute the pattern detector.

After Section 5.1 provided an overview of these modules and of the pattern detector as a whole, each module is presented in detail in Sections 5.2 to 5.6. Section 5.7 summarizes all algorithmic parameters configuring the modules. These parameters are optimized within the *Optimization* stage of *SynOpSis* with respect to the objectives listed in Section 3.5.2. In order to do so, Section 5.8 describes a procedure enabling to evaluate the objective functions for given parameter sets in an automatic fashion. This is done by matching synthetic ground truth to detector results. Finally, Section 5.9 concludes the chapter and leads over to the pattern classifier in Chapter 6.

5.1 Introduction

This section gives an overview of the pattern detector to be presented in the following sections. In a basic version, this detector was described in [SWL+11] and [LST+13a], and it has been heavily extended since then. Its extended version will be described in detail here, including references to further publications as they are used. The detector is realized as a real-time-capable streaming pipeline on the GPU. A schematic depiction of this pipeline is given in Figure 5.1. The input stream consists of a time series of *PAMONO* images, obtained either by synthesis (detector in *Optimization* stage in Figure 3.2) or from the sensor (detector in *Application* stage in Figure 3.2). Its output are patterns represented as polygons, delineating areas in the images that are candidates for being affected by nano-object adhesions. Furthermore, a processed variant of the time series of input images is part of the output. From these enhanced images, intensity-based local features are extracted that serve in classifying

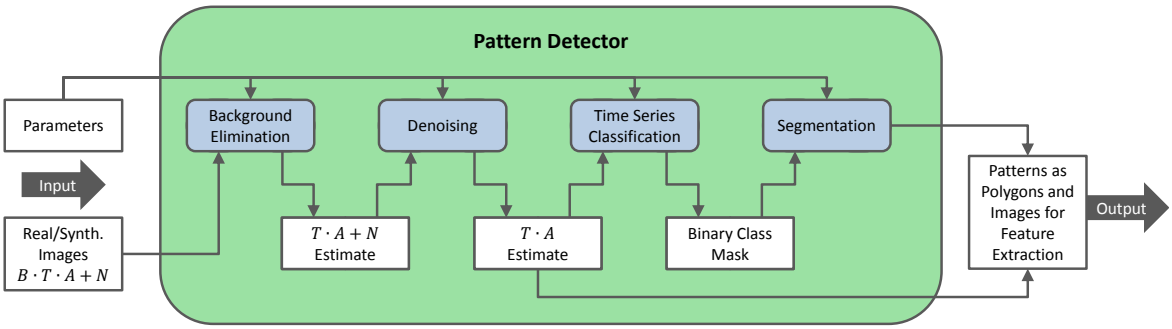


Figure 5.1: Pattern Detector for PAMONO. A concrete version of the abstract pattern detector from Figure 3.3 is shown that has been specifically designed for *PAMONO* data analysis. It takes real/synthetic images as its input, along with a parameter set configuring the algorithms in the parametric pipeline constituting the detector. Background elimination and denoising remove the constant background B and the noise N from the input signal, resulting in the product $T \cdot A$ of target patterns T and artifacts signals A. Time series classification identifies salient coordinates in $T \cdot A$, which are aggregated to polygons within segmentation. These polygons are the output patterns and are candidates for delineating areas affected by a nano-object adhesion. Furthermore the $T \cdot A$ signal is part of the output as it serves as the basis for extracting intensity-based local features from the areas covered by the polygons, cf. Section 6.2. These features are used in a subsequent classification step to sort out False Positive (FP) detector responses, cf. Chapter 6.

the output polygons as being related either to target patterns (nano-objects) or to non-target patterns (artifacts), cf. Section 6.2.

In the following depiction of the four modules of the pattern detector in Figure 5.1, the input time series of images is denoted as I. This I is a placeholder for both, synthetic and real input, which also holds for the signal components of which I consists, as modeled by Equation (4.1). For convenience and due to their frequent use in this chapter, these components are briefly summarized here: A *PAMONO* measurement I is composed as $I = B \cdot T \cdot A + N$, consisting of

- B, the background component modeled as being constant over time, exhibiting intensities that dominate all other components,
- T, the low amplitude target patterns signal that is caused by nano-objects attaching to the sensor surface,
- A, the artifacts component with an amplitude similar to T but containing only signals that are due to non-target patterns or deviations of B from the constancy assumption, and
- N, the noise incurred by the sensor.

Hence, the indirect proof for the nano-objects to be detected resides in the T component, while all other components adversely affect the analysis process.

The pattern detector in Figure 5.1 receives a raw *PAMONO* measurement I as its input, along with a set of algorithmic parameters, configuring its four processing modules. These modules and their tasks in *PAMONO* data analysis are as follows:

1. **Background elimination** tackles the high amplitudes of the background component B by dividing them out, respecting the signal model from Equation (4.1). Furthermore, it serves to separate spatially overlapping target patterns in the temporal domain. Details on background elimination are provided in Section 5.2.

2. **Denoising** alleviates the noise component N and thus extracts an estimate of the $T \cdot A$ component. Details on denoising are provided in Section 5.3.
3. **Time series classification** classifies the time series observed in the $T \cdot A$ estimate as being salient or not. Non-salient time series contain only residual noise, while salient time series exhibit signals that are due to the T or the A component. No decision is made, however, which of the two components is responsible for saliency. Two alternative approaches to this time series classification are presented in Sections 5.4 and 5.5, respectively.
4. **Segmentation** aggregates the saliency class mask output by time series classification to polygons, residing in the same spatiotemporal coordinate system as the input measurement I. These polygons indicate *where* and *when* time series were salient, thus delineating areas that are candidates for being affected by a nano-object adhesion. Details on segmentation are provided in Section 5.6.

Each section presenting a detector module is closed by a subsection, listing and describing the parameters arising in that module. For an overview, Section 5.7 lists *all* parameters of the pattern detector. Evaluations of the proposed methods and their interplay are given in the overall validation of *SynOpSis* in Chapter 7. However, the choice between the two alternatives for time series classification is fixed in advance within this chapter. The corresponding evaluation is given in Sections 5.5.4 and 5.5.5.

The overall goal of the pattern detector can be summarized as marking areas that are candidates for being affected by a nano-object adhesion with polygons. Candidacy information is obtained by analyzing an estimate of the $T \cdot A$ component of the signal model. The artifacts component A and possible estimation errors in this $T \cdot A$ estimate demote the output polygons to be merely *candidates*. The target signal component T is not separated from A on the pixel level because the diverse nature of the artifacts impedes their modeling on the pixel level. Instead, the $T \cdot A$ estimate as computed by the denoising module is a part of the detector output, as indicated by the bottommost connection in Figure 5.1. Along with the polygons, it is processed further by the methods in Chapter 6: Local features are extracted from $T \cdot A$ within the areas covered by the detected polygons, cf. Section 6.2. These features measure properties of the underlying intensities. They are used in classifying the polygons either as actual nano-objects (True Positive (TP) detector responses due to the T component) or as spurious detections (False Positive (FP) detector responses due to the A component). This means $T \cdot A$ is separated on the polygon level, instead of separating it on the pixel level.

5.2 Background Elimination

Background elimination serves to remove the temporally constant, high-intensity background component $B(x, y)$ from the *PAMONO* sensor images $I(x, y, t)$ it receives as its input, cf. Figure 5.1. Its second input are the background elimination parameters that are explained throughout this section. Background elimination works on a per-image basis: For each 2-D image $I(\circ, \circ, t)$, a corresponding output image is computed, i.e. the input time series of *PAMONO* images is transformed into a new time series of images with the background $B(x, y)$ removed. Looking at the signal model $I(x, y, t) = B(x, y) \cdot (T \cdot A)(x, y, t) + N(x, y, t)$ from Equation (4.1), removing $B(x, y)$ means that the output of background elimination is a time series of images $(T \cdot A)(\circ, \circ, t) + N(\circ, \circ, t)$, containing the product of the target

patterns signal T with the artifacts signal A and additive noise N. The large multiplicative influence exerted by the high intensities in B are removed, thus making spatially neighboring intensities in the resulting $T \cdot A + N$ signal comparable.

While partly drawing from techniques applied in fluorescence microscopy (cf. Chapter 12 of [WMC10]), the background elimination approach presented here has been developed specifically for *PAMONO* data, exploiting its spatiotemporal structure on the basis of the signal model from Equation (4.1). Besides the main goal of removing the B component, background elimination furthermore serves to increase data quality because it is realized via methods incorporating temporal denoising. These methods are applied in a sliding window fashion along the temporal axis, which firstly makes background elimination streaming-capable, and secondly facilitates analysis because past nano-objects become part of the eliminated background over time. Therefore, nano-objects that overlap spatially, can be separated in the temporal domain, if they appear at moments in time that are further apart than the temporal resolution provided by the sliding window background elimination kernel.

Besides the *temporally constant* background component $B(x, y)$ from the signal model in Equation (4.1), two new terms, the 'past' and the 'present' signals, are required to denote the *temporally variant* components in background elimination. Both are prerequisites of the following sections and are therefore defined now:

Definition 5.1. *Let t_c be the temporal index of the image $I(x, y, t_c)$ to be currently processed by background elimination, within a stream I of* PAMONO *images as defined by the signal model in Equation (4.1). Then the* ***past*** *$\rho(x, y, t_c)$ at time t_c denotes the temporally constant, high-intensity background component $B(x, y)$* and *any temporally variant signals in any other component that happened* before *t_c. This includes the nano-objects in the target patterns component T, as well as artifacts A. Conversely, the term* ***present*** *$\phi(x, y, t_c)$ at time t_c denotes $B(x, y)$ along with any temporally variant signals in any other component that happen* at *time t_c.*

Due to the temporal constancy of $B(x, y)$, both, the past ρ and present ϕ contain the same B. Background elimination serves to remove this constant B and the temporally variant signal components that occurred *before* t_c. More exactly: For time t_c, background elimination removes the past $\rho(x, y, t_c)$ from the present $\phi(x, y, t_c)$. As an example, a nano-object that attaches to the sensor surface at time $t_a < t_c$ causes a step function at time t_a, and the step plateau remains constant from then on because the nano-object remains attached, cf. Figure 2.2. As $t_a < t_c$, the step function is part of the past $\rho(x, y, t_c)$ at time t_c and will hence be removed by background elimination, facilitating the detection of new nano-objects attaching *later* around the same spatial coordinates.

Therefore, background elimination removes the high intensities of B that occur in both, past and present signal and furthermore extracts exactly those temporally variant signals that change at time t_c, i.e. between ρ and ϕ. Section 5.2.1 presents a linear approach for estimating ρ and ϕ and subsequently provides a technique for background elimination that exploits the signal model from Equation (4.1). The same technique can be applied to the rank order-based nonlinear estimation of ρ and ϕ that is presented in Section 5.2.2.

5.2.1 Averaging Background Elimination

Background elimination operates on a per-image level, i.e. for each input image $I(\circ, \circ, t_c)$, a past image $\rho(\circ, \circ, t_c)$ and a present image $\phi(\circ, \circ, t_c)$ are estimated, from which the resulting

estimate of $(T \cdot A)(\circ, \circ, t_c) + N(\circ, \circ, t_c)$ is computed. With this per-image nature clarified, the index wildcards $\circ$ are in the following replaced again with the actual indices x, y. Looking at Definition 5.1, a simple idea for estimating the past signal $\rho(x, y, t_c)$ at time t_c is to compute a temporal aggregate over all sensor images $I(x, y, t)$ with $t < t_c$. This aggregate can e.g. be obtained by averaging all preceding images over the temporal dimension. Doing so, the resulting $\rho(x, y, t_c)$ fulfills Definition 5.1 because it contains the temporally constant $B(x, y)$ and anything temporally variant that happened *before* t_c. However, averaging over *all* past images means that transient fluctuations influence $\rho(x, y, t_c)$. Their influence becomes the larger, the longer their duration. To counteract this effect, only the preceding image $I(x, y, t_c - 1)$ could be used as $\rho(x, y, t_c)$. This in turn is corrupted by more noise N, which was previously alleviated by averaging over a possibly large number of images. The compromise between these extremes that was chosen for *PAMONO* is to average over a window of the most recent $w^\rho \in \mathbb{N}_{>0}$ images preceding t_c:

$$\rho(x, y, t_c) = \frac{1}{w^\rho} \sum_{i=t_c-w^\rho}^{t_c-1} I(x, y, i). \tag{5.1}$$

The parameter w^ρ trades off a better Signal-to-Noise Ratio (SNR) against temporal resolution (because it smoothes the input signal along the temporal dimension) and is subject to optimization.

The simplest way of obtaining an estimate of the present signal $\phi(x, y, t_c)$ is to use the raw sensor image $I(x, y, t_c)$ recorded at time t_c. It readily fulfills Definition 5.1 because it contains the temporally constant $B(x, y)$ and anything temporally variant that happens *at* time t_c. However, for the same SNR versus temporal resolution trade-off discussed in the context of the past signal, it is beneficial to conduct a windowed estimation of $\phi(x, y, t_c)$ and to leave the window size $w^\phi \in \mathbb{N}_{>0}$ as a parameter for optimization. Therefore, analogously to Equation (5.1), the present at time t_c is estimated as

$$\phi(x, y, t_c) = \frac{1}{w^\phi} \sum_{i=t_c}^{t_c+w^\phi-1} I(x, y, i). \tag{5.2}$$

The parameter w^ϕ introduces latency proportional to its value into real-time analysis: Obtaining the background elimination result for time t_c requires the subsequent $w^\phi - 1$ sensor images to be already available. Note that the windowed average estimations of ρ and ϕ can be regarded as a simple kind of temporal denoising/smoothing with a kernel size defined by w^ρ and w^ϕ.

Given past and present estimates $\rho(x, y, t_c)$ and $\phi(x, y, t_c)$ for time t_c, the background elimination result at t_c is computed as

$$\frac{\phi(x, y, t_c)}{\rho(x, y, t_c)} \approx (T \cdot A)(x, y, t_c) + N(x, y, t_c). \tag{5.3}$$

This approximation can be inferred from the signal model in Equation (4.1) as follows: Both, ρ and ϕ are temporal aggregates of *PAMONO* sensor images $I = B \cdot T \cdot A + N$. For the first part of the argument, the noise summand N is temporarily ignored, making I purely multiplicative. The B component is temporally constant and thus identical in both ρ and ϕ. Therefore it is canceled out by the division in Equation (5.3), removing the irrelevant high intensities. The remaining $T \cdot A$ components are temporally variant: Only those values

that are identical in past and present cancel out. Any values that changed between past and present do not cancel out, yielding a $T \cdot A$ estimate containing only those events that happen at present, as desired. Realizing background elimination as a division accommodates the multiplicative nature of the $B \cdot T \cdot A$ components of the signal formation modeled in Equation (4.1). However, for the noise term N, which has up to now been ignored, the model is violated. Like in synthesis, as described in Section 4.3.2, this is negligible: First of all, the intensities in N are small compared to those in B and are furthermore decreased due to the fact that the windowed average estimations of ρ and ϕ realize a simple temporal denoising. A second argument follows from assuming the availability of noise-free ρ and ϕ to which noise terms N^ρ and N^ϕ are added, respectively. Performing background elimination in accordance with Equation (5.3) yields

$$\frac{\phi + N^\phi}{\rho + N^\rho} = \frac{\phi}{\rho + N^\rho} + \frac{N^\phi}{\rho + N^\rho}.$$

The N^ϕ term in the numerator of the right summand is divided by $\rho + N^\rho$, which is dominated by the B in ρ and thus much larger than N^ϕ. The N^ρ term in the denominator of the left summand is added to ρ. As ρ involves the much larger B, the corruption exerted by N^ρ on the denominator is negligible. The error incurred by ignoring N^ρ and N^ϕ in background elimination is again regarded as (different) noise. This is the N on the right hand side of the approximation in Equation (5.3).

Background elimination is applied to *PAMONO* sensor data in a sliding window fashion: As soon as the first $w^\rho + w^\phi$ images have been recorded, Equations (5.1) to (5.3) can be evaluated for the first time. Every further recorded sensor image results in a new image being output by background elimination, as t_c is increased by one and the past and present estimation windows defined by w^ρ and w^ϕ are slid forward by one. This sliding window approach is well-suited for the streaming scenario that arises in *PAMONO* data analysis.

Furthermore, it introduces a mechanism for "forgetting" past events: Once a nano-object adhesion falls into the past window, it starts to fade and is completely "forgotten" from the moment at which the past window only contains data from the high plateau after the step occurred, cf. Figure 2.2. This prevents clutter and overlap in the spatial domain: Nano-objects that overlap spatially but not temporally can be separated, if they appear at moments in time that are further apart than the temporal resolution imposed by w^ρ and w^ϕ. Forgetting is also useful for feature extraction because then the features are only computed over the most recent events occurring on the sensor surface.

Another crucial contribution that background elimination makes to feature extraction is the removal of the irrelevant high amplitudes of the background component B: Intensities of time series measured in spatially adjacent pixels become comparable because their multiplication with the irrelevant and considerably larger intensities in B is divided out. Therefore, local features of intensity can be computed because spatially adjacent pixels reside on the same intensity scale. Furthermore, this makes the heights of the steps associated with nano-object adhesions independent of whether the step occurs in a bright or a dark area of the sensor surface because the gain exerted by B is factored out. As a result, step heights become more homogeneous, such that their utility in feature extraction increases and the choice of thresholds becomes simpler, cf. Section 5.4.1. A more diagnostic advantage is that according to the underlying physics, similar step heights indicate similar nano-object sizes [ZKG+10], a property that only holds if the influence of B is eliminated.

Besides the window sizes w^ρ and w^ϕ used in estimating the past ρ and present ϕ, a third parameter is optimized for the background elimination module. This parameter is the boolean switch $b^{\text{bg}} \in \{0, 1\}$, selecting whether to use the averaging-based past and present estimation described in this section or the median-based technique presented in the following one.

5.2.2 Median Background Elimination

Median-based background elimination, replaces the sliding temporal average in estimating the past and present signals with a sliding temporal median:

$$\rho(x, y, t_c) = \text{Median}(\{I(x, y, t_c - w^\rho), \ldots, I(x, y, t_c - 1)\}), \tag{5.4}$$

$$\phi(x, y, t_c) = \text{Median}(\{I(x, y, t_c), \ldots, I(x, y, t_c + w^\phi - 1)\}). \tag{5.5}$$

Eliminating ρ from ϕ is done the same way as in averaging background elimination, i.e. by applying Equation (5.3). Arguments and explanations given there carry over to median background elimination because both, average and median, are considered and employed as methods for temporal denoising of the input data. A difference, however, arises in how the techniques respond to a temporal step function used as input, which is covered in the next section.

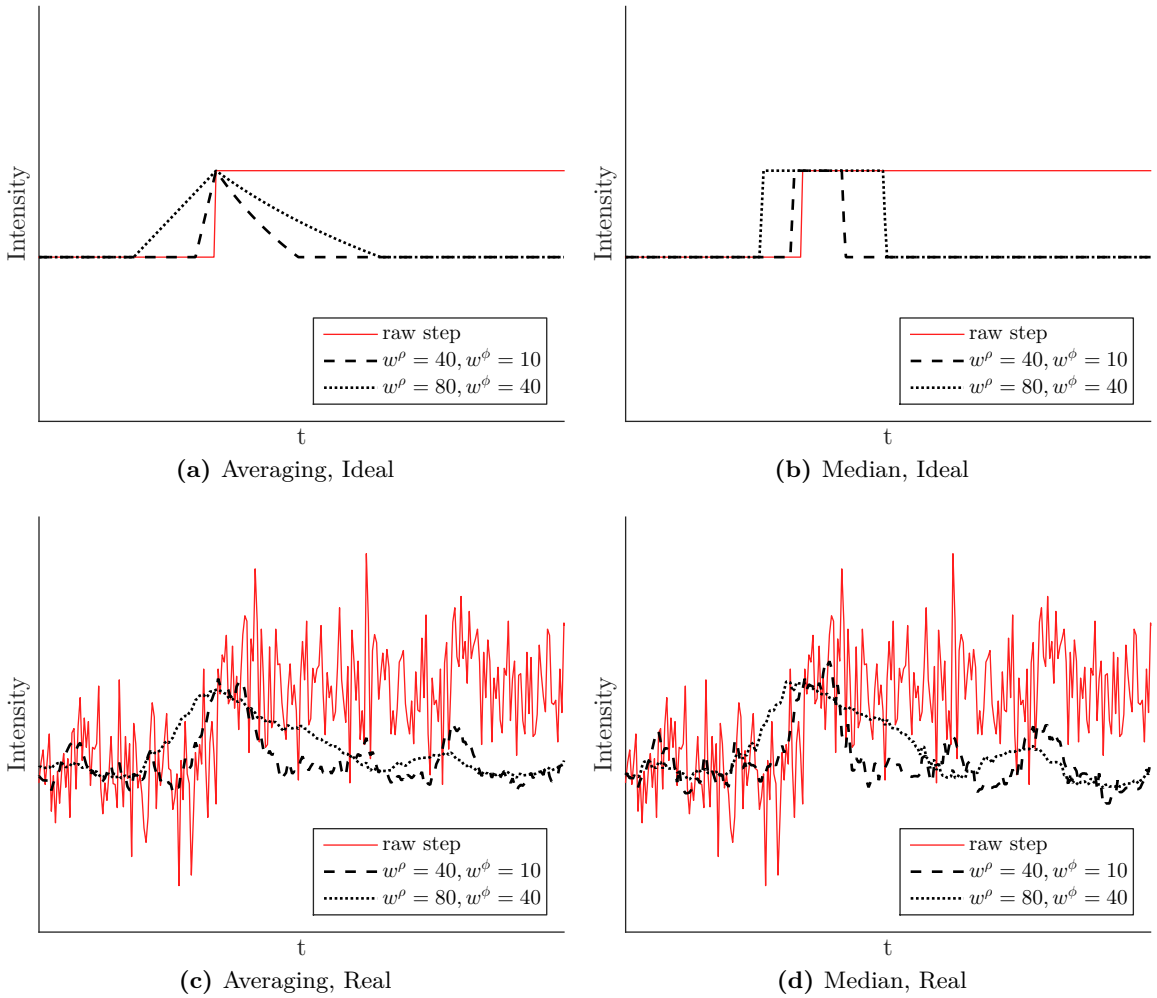
5.2.3 Step Responses of Background Elimination Techniques

Figure 5.2 demonstrates the different results of averaging and median background elimination for temporal step functions used as inputs. Such results are in the following denoted as the **step responses** of background elimination. A step response like the ones shown in the figure is obtained by applying background elimination to a time series of images, followed by selecting a single coordinate (x, y) at which the temporal step occurs and then plotting the observed values after background elimination over t. Red lines in the plots show the step functions that are used as input to background elimination. In (a)–(b), an *ideal* step function is used, while (c)–(d) use a *real* step function, that was observed in a dataset with comparably high SNR. The choice for a step function with high SNR was taken to make it discernible in the figure. Black dashed and dotted lines show the respective background elimination result for different window sizes w^ρ and w^ϕ used in past and present estimation.

Figure 5.2a illustrates for an ideal step input, that averaging smoothes the step approximately[1] to a rising and a falling ramp function. The slope of the rising ramp corresponds to w^ϕ, while the slope of the falling ramp corresponds to w^ρ, as these parameters determine the temporal distance between the maximum of the step response and its beginning (determined by w^ϕ) and ending (determined by w^ρ). Hence the temporal extension of the step response corresponds to the sum of w^ρ and w^ϕ. The maximum of the step response is attained at the time of the original step in the input.

In contrast to averaging, the median is step-preserving, as is demonstrated in Figure 5.2b, by applying it to an ideal step signal. The step response starts when more than half of the input step has entered the present window because then the median changes. Accordingly, the step response ends when more than half of the step has entered the past window. Consequently, the temporal extension of the step response corresponds to the sum of w^ρ and w^ϕ, divided by two. The time of the input step can be inferred from this step response by adding $w^\phi/2$ to

[1] The slightly nonlinear behavior of the falling ramp is due to the division operation in Equation (5.3).



(a) Averaging, Ideal **(b)** Median, Ideal

(c) Averaging, Real **(d)** Median, Real

Figure 5.2: Step Responses of Background Elimination Methods. (a) Averaging background elimination smoothes an ideal step function used as input (red) to a ramp-like function (black). (b) Median background elimination is step-preserving for an ideal input step. (c) Larger averaging window sizes w^ρ, w^ϕ (dotted versus dashed black line) decrease noise in real data, while also reducing temporal resolution. (d) The Step-preservation property of the median suffers in case of a low Signal-to-Noise Ratio (SNR) in the input.

the time of the step in the step response. As a summary, with regard to *ideal* step inputs, the main difference between averaging and median background elimination is step-smoothing versus step-preservation.

In Figure 5.2c, averaging background elimination is applied to *real* data. The obtained step responses behave like their corresponding ideal step responses, but with noise. For larger w^ρ and w^ϕ (dotted versus dashed black line), the noise in the result decreases, while the temporal extension of the step response increases, thus decreasing the temporal resolution of background elimination.

Figure 5.2d shows the result of median background elimination, attained for the same input as in (c). On real data, the step-preservation property of the median suffers from low SNR: Results look approximately like those in (c) because when the sorting that is part of median computations is applied to *very noisy* values, it is less likely to produce a sharp step.

Practical Effect

A consequence of these observations is that subsequent processing modules must adapt to the step response of background elimination, defined by the parameter choices for w^ρ, w^ϕ and b^{bg}. This is of particular concern for the methods used in the time series classification module in Figure 5.1.

Therefore, the template to be used in fuzzy template matching (cf. Section 5.4) needs to be adapted to account for the different step responses caused by the different parameters in background elimination. This works as follows: For averaging past and present estimation ($b^{\text{bg}} = 0$), the according background elimination is applied to an ideal step function, using the given values of w^ρ and w^ϕ. The resulting step response is used as the **ideal template** time series to compare observed time series against. For median past and present estimation ($b^{\text{bg}} = 1$), the step-preservation property could mislead to using an ideal median step response like those in Figure 5.2b as the template. However, for low SNRs as in *PAMONO* time series, it is more advisable to process an ideal step function for given w^ρ and w^ϕ with *averaging* background elimination and use this result as the template, which is motivated by the median step response for real data in Figure 5.2d.

A second approach that can be used as the time series classification module in Figure 5.1 is wavelet-based time series classification (cf. Section 5.5). Here the impact of background elimination is as follows: The time series classifier must be trained using the same background elimination parameters as are later applied to the data to be classified. Both adaptations listed here are handled automatically by the pattern detector.

5.2.4 Parameters

The parameters that are optimized for the background elimination module, along with the examined ranges of values are:

$b^{\text{bg}} \in \{0, 1\}$
: a boolean selecting whether to aggregate the past and present estimate via temporal averaging ($b^{\text{bg}} = 0$) or temporal median ($b^{\text{bg}} = 1$),

$w^\rho \in \{1, \ldots, 40\}$
: an integer controlling the length of the temporal window over which the past estimate is aggregated (cf. Equations (5.1) and (5.4), respectively),

$w^\phi \in \{1, \ldots, 40\}$
: an integer controlling the length of the temporal window over which the present estimate is aggregated (cf. Equations (5.2) and (5.5), respectively).

5.3 Denoising

Background elimination removed the background component B from the signal $I = B \cdot T \cdot A + N$ recorded by the *PAMONO* sensor. Furthermore, it attenuated the noise component N due to its smoothing properties. The denoising techniques presented in this section target the residual noise remaining after background elimination and hence output a denoised estimate $T \cdot A$ of the product of the target patterns and artifacts signal. This time series of $T \cdot A$ images is used in the subsequent time series classification module, as well as in the feature extraction described in Section 6.2, cf. also Figure 5.1. Before this section starts with introducing the

denoising techniques, the order of background elimination and denoising is discussed in the following paragraphs, followed by a note on the choice of denoising techniques to be presented.

On Applying Denoising After Background Elimination

Restorative techniques like denoising are often applied before application-specific image enhancement [GW07] like background elimination. Nevertheless, the preceding paragraph stated that in the pattern detector in Figure 5.1, denoising is applied after background elimination. There are several reasons for choosing this order:

- Firstly, intensity variation in the target patterns signal T is very small, compared to the background B. Denoising the images containing B would involve the highly spatially variant intensities in B entering the filtering process, which could destroy the T signal. This is particularly true for *nonlinear* filters like the median (cf. Section 5.3.1), if applied in the spatial domain: The small intensity variations in T can be easily covered up by the large spatial variations in B.
- Secondly, dark and light areas in B undergo a different amplification, exerted by the division in Equation (5.3): Flaws of a denoising applied before background elimination would be amplified more strongly if they occur where B is dark.
- Thirdly, background elimination already incorporates *temporal* denoising, due to its smoothing properties. As B is assumed constant in *PAMONO*, all samples along the temporal axis contain the same B component, hence temporal denoising does not suffer the problem mentioned in the first point with respect to spatial denoising. The denoising carried out *after* background elimination is intended to *further* denoise the results, targeting the residual noise that evaded the temporal denoising exerted by background elimination. For this purpose, applying background elimination before denoising enables exploiting *spatial* correlations in the data. Any spatial denoising should be carried out *after* background elimination for the reasons stated in the previous points.

Consequently, the denoising techniques described here are spatial and spatiotemporal techniques. They exploit the spatial correlations that background elimination revealed, to firstly address the remaining noise that could not be eliminated by temporal denoising, and to secondly remove any *new* signal flaws introduced e.g. by the division in background elimination.

Choice of Denoising Techniques

Due to the nonlinear operations[2] in background elimination, the nature of the noise remaining after background elimination is no longer the same as in the signal model in Equation (4.1). Assumptions like the Mixed-Poisson-Gaussian (MPG) nature of Charge-Coupled Device (CCD) sensor noise [MSB95] do not hold any longer after background elimination. Lack of the MPG property means that applying a Variance Stabilizing Transform (VST) like Generalized Anscombe Transform (GAT) [SMB98] to alleviate Poissonity, followed by a second filter addressing Gaussian noise is not anymore better justified than applying general purpose denoising techniques. For this reason, the idea followed here is to leave the choice of denoising

[2]The nonlinear operations applied in background elimination are the division in Equation (5.3) and the medians in Equations (5.4) and (5.5).

methods, along with method parameters, open to optimization: The goal of denoising in *PAMONO* data analysis is not signal restoration but extracting an estimate of the $T \cdot A$ components in a fashion that is suitable for further application-specific processing steps. Any method, or combination of methods, accomplishing this task is viable. Hence optimization can arbitrarily combine the denoising techniques presented in the following, by individually enabling or disabling them. Filter order, however, is fixed and is discussed separately, where the respective filters are presented. The order of presentation is from general to more specific. It does not reflect the order in which the filters are applied.

With these points discussed, the structure of this section is as follows: Section 5.3.1 describes basic linear and nonlinear filters for general spatial denoising tasks. Section 5.3.2 presents a fuzzy logic-based spatiotemporal denoising filter to identify and remove impulse noise from video sequences. Section 5.3.3 introduces *PAMONO*-specific heuristics for data cleaning. Finally, Section 5.3.4 lists and describes all parameters arising in the discussed methods that are to be optimized.

5.3.1 Spatial Denoising Techniques

Spatial denoising techniques work on the 2-D image-level: They are applied to an image $I(\circ, \circ, t_c)$ at a given time t_c, independent of images at other times t. Hence these methods can trivially accept a *stream* of images as input. For convenience of notation, the 2-D image $I(\circ, \circ, t_c)$ at time t_c is in the following denoted as $\iota(x, y)$, dropping the temporal index. Analogously, the filtering result is denoted as $\omega(x, y)$.

Average Filter

The first spatial denoising filter to be presented is the average filter [GW07]: For each position (x, y) in the input image $\iota(x, y)$, the value of the output image $\omega(x, y)$ is determined by computing the average intensity in a window of width $K^{\mathrm{w}}_{\mathrm{avg}} \in \mathbb{N}_{>0}$ and height $K^{\mathrm{h}}_{\mathrm{avg}} \in \mathbb{N}_{>0}$, centered at (x, y) in ι. Filling every pixel (x, y) in the output ω by computing such an average is mathematically equivalent to the discrete convolution [GW07] of ι with an averaging kernel $\kappa_{\mathrm{avg}}(x, y)$, where

$$\kappa_{\mathrm{avg}}(x, y) \coloneqq \begin{cases} \frac{1}{K^{\mathrm{w}}_{\mathrm{avg}} K^{\mathrm{h}}_{\mathrm{avg}}} & \text{for } (x, y) \text{ in } K^{\mathrm{w}}_{\mathrm{avg}} \times K^{\mathrm{h}}_{\mathrm{avg}} \text{ window, center at } (0, 0) \\ 0 & \text{otherwise.} \end{cases} \tag{5.6}$$

Note that in computing convolution results for the boundary regions of ω, the support of kernel κ_{avg} leaves the image boundary of ι. Such boundary cases must be treated, e.g. by mirroring ι about its boundaries [GW07]. The parameters to be optimized for averaging-based denoising are the boolean switch $b^{\mathrm{denoise}}_{\mathrm{avg}}$ enabling or disabling it, and the kernel width and height $K^{\mathrm{w}}_{\mathrm{avg}}$ and $K^{\mathrm{h}}_{\mathrm{avg}}$. Increasing $K^{\mathrm{w}}_{\mathrm{avg}}$ increases the amount of smoothing/denoising in the horizontal direction, analogously for $K^{\mathrm{h}}_{\mathrm{avg}}$ and the vertical direction.

Gauß Filter

According to the convolution theorem, convolving an image ι with a filter kernel κ in the spatial domain is equivalent to multiplying them in the Fourier domain (and inversely Fourier-transforming the result) [Wah89]. In the Fourier domain, the averaging kernel κ_{avg} from the

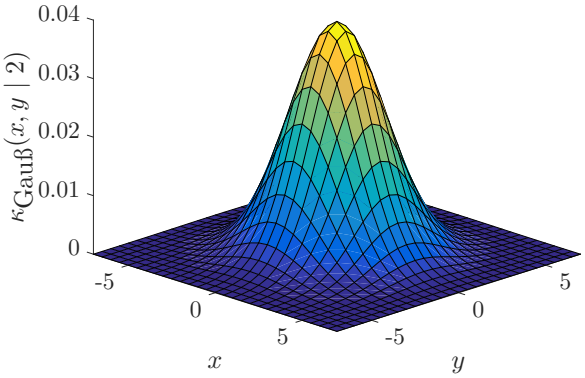


Figure 5.3: 2-D Gauß Kernel. Plot of a 2-D Gauß kernel $\kappa_{\text{Gauß}}$ over x, y for $\sigma_{\text{Gauß}} = 2$.

previous paragraph is a 2-D sinc-function. Multiplying this Fourier spectrum with the Fourier spectrum of ι illustrates that the attenuation of frequencies in ι exerted by the averaging denoising filter is rather "bumpy", owing to the bumpiness of the sinc-function. Depending on the input, this can yield unsatisfactory results.

Convolution of ι with a 2-D Gauß kernel (example in Figure 5.3) constitutes a denoising filter with a frequency attenuation that is not bumpy because the Fourier transform of a Gaussian distribution is again a Gaussian distribution with reciprocal standard deviation [Wah89]. The 2-D Gauß kernel [Low04] is defined as

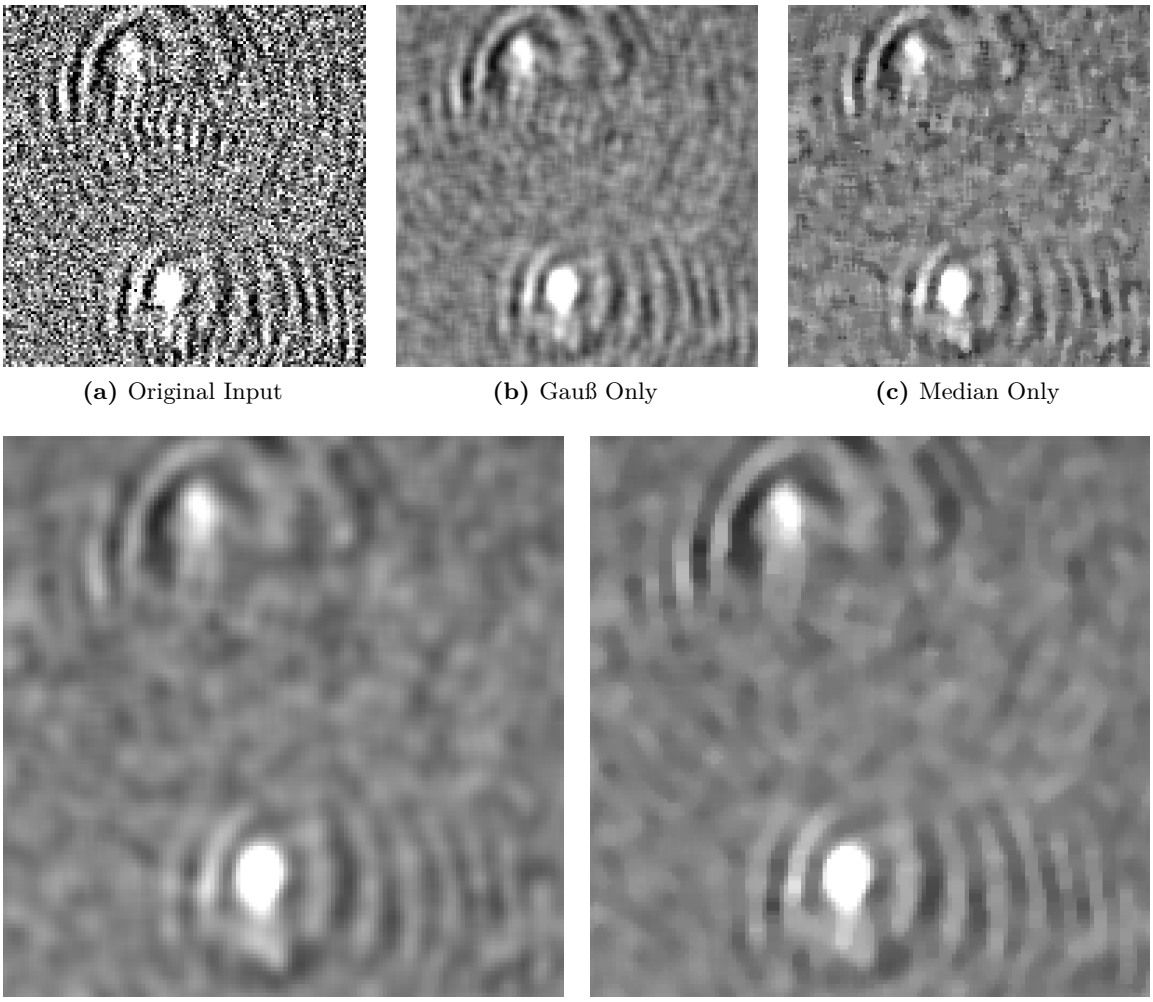
$$\kappa_{\text{Gauß}}(x, y \mid \sigma_{\text{Gauß}}) = \frac{1}{2\pi\sigma_{\text{Gauß}}^2} \exp(-(x^2 + y^2)/2\sigma_{\text{Gauß}}^2). \tag{5.7}$$

Here $\sigma_{\text{Gauß}} \in \mathbb{R}_{>0}$ is the standard deviation of the underlying Gaussian normal distribution. It is a parameter to be optimized for the pattern detector, along with the switch $b_{\text{Gauß}}^{\text{denoise}} \in \{0, 1\}$, enabling the Gaussian denoising filter. Increasing $\sigma_{\text{Gauß}}$ increases the size of $\kappa_{\text{Gauß}}$ in both directions. Stated more exactly, it increases the exerted amount of smoothing/denoising isotropically. Figure 5.4b shows an exemplary result of filtering a *PAMONO* image after background elimination with a Gaussian kernel.

Median Filter

By computing a κ-weighted sum of image intensities ι (i.e. convolution), linear techniques such as averaging or Gaussian denoising are strongly affected by outlier pixels. The reason is that for every image position at which the kernel κ exhibits a non-zero value over an outlier pixel, an outlier summand in the filter response is incurred.

Median filtering [GW07] is a denoising method that is more robust to outliers. Like linear methods, it employs a sliding window over the domain of the image ι to compute the values of the filter response ω. However, it does so in a nonlinear way, using order statistics: $\omega(x, y)$ is defined as the median over the intensities in a $K_{\text{med}}^{\text{w}} \times K_{\text{med}}^{\text{h}}$ window in ι, centered at (x, y). Unlike with linear methods, it is guaranteed that all *intensities* in ω actually exist in ι. As a drawback, since not all *coordinates* in ι necessarily contribute to the result, it is possible to construct pathological inputs where the median filter "hallucinates" structures that do not exist in the input image, e.g. the grating-like structures in Figure 5.4c. A typical use case for



(a) Original Input **(b)** Gauß Only **(c)** Median Only

(d) Median Then Gauß **(e)** Gauß Then Median

Figure 5.4: Filter Order – Example. (a) shows a region from a *PAMONO* image after background elimination, containing two nano-object adhesions. (b) and (c) are the results of applying a spatial Gauß and median filter, respectively, exhibiting undesired structures caused by filtering. Cascading both filters removes these structures (d)–(e). Applying the Gauß filter *before* the median filter (e) provides superior contrast and more accented edges, leading to higher step responses in the time series of the affected pixels. Thus this filter order was chosen as it facilitates time series classification.

the median filter is removing small disturbances in an image, e.g. salt and pepper noise. For a low density of noisy pixels, this type of noise can be removed nearly completely. Furthermore, the median filter introduces less image blur, compared to an averaging filter of the same size [GW07]. The parameters to be optimized for the median filter are the sliding window width $K^{\mathrm{w}}_{\mathrm{med}} \in \mathbb{N}_{>0}$ and height $K^{\mathrm{h}}_{\mathrm{med}} \in \mathbb{N}_{>0}$, along with the switch $b^{\mathrm{denoise}}_{\mathrm{med}} \in \{0, 1\}$ enabling the filter.

Filter Order

Figure 5.4 illustrates the impact of filter order on the obtained results. In (a), an amplified real *PAMONO* image after background elimination is shown, while (b) and (c) show the results of applying solely the Gauß, respectively the median filter. Comparing these images

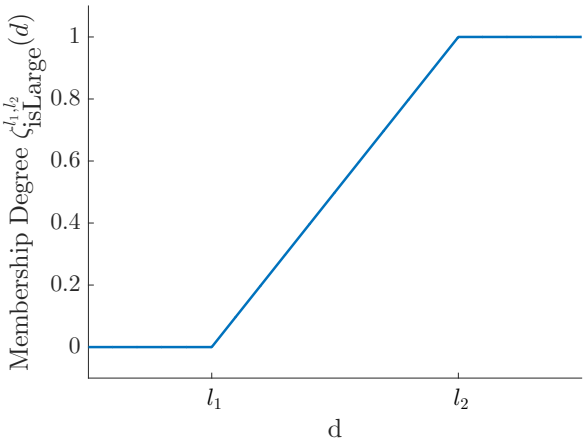


Figure 5.5: Membership Function for the Fuzzy Set of Large Intensity Differences. An intensity difference d can be definitely large ($d \geq l_2$), definitely not large ($d \leq l_1$), or something fuzzy in between ($l_1 < d < l_2$). The membership function is defined in Equation (5.8). Figure adapted from [MNK11].

to the results of cascading both filters, as in (d) and (e), the first thing to note is that this cascade improves perceived image quality, justifying the approach of allowing the optimization to enable the filters simultaneously. Comparing the result of firstly applying the median filter, then the Gauß filter (d) to the reverse order (e) provides an empirical argument for choosing the latter as the filter order: (e) exhibits superior contrast compared to (d), which is especially true for the edges of the bright and dark areas surrounding the nano-object adhesions. Better contrast and less blurring about the edges yields stronger step responses in the affected pixels, thus facilitating subsequent time series classification.

Therefore, in the denoising module of the pattern detector from Figure 5.1, the linear filters (averaging and Gauß filter) are applied *before* the median filter. The internal order between the averaging and Gauß filter does not matter: They are commutative because they are linear filters.

5.3.2 Fuzzy Spatiotemporal Denoising

Fuzzy spatiotemporal denoising takes a noisy time series of images $T \cdot A + N$ as input, along with the parameters configuring the method. Its purpose is to reduce the noise remaining after background elimination by exploiting both, spatial and temporal correlations.

In contrast to classical boolean logic, where propositions can be either true or false, respectively from the set $\{0, 1\}$, fuzzy logic distinguishes continuous degrees of truth from the interval $[0, 1]$. If propositions about the membership of entities to sets are regarded, as will be done in fuzzy denoising, one speaks of **fuzzy sets**, as initially proposed in the seminal work by Zadeh [Zad65]: Membership of an entity to a set is no longer described by boolean values from $\{0, 1\}$, but by a fuzzy set membership function ζ, mapping to the interval $[0, 1]$. If the domain of ζ is not fuzzy, the mapping realized by ζ is referred to as the **fuzzification** of that domain. Unlike probabilities, fuzzy set memberships of an entity can sum to values larger or smaller than one over all sets. Mapping a fuzzy set membership ζ to a crisp set membership from $\{0, 1\}$ is called **defuzzification**.

The method described here adapts the fuzzy video denoising approach by Melange, Nachtegael, and Kerre [MNK11] for *PAMONO* data [LWT12] and runs on the GPU for real-time-capable stream processing [LST+13a]. Melange, Nachtegael, and Kerre developed their method to specifically address random impulse noise in video sequences. In this context, **random impulse noise** is defined as the replacement of intensities at independently drawn, uniformly distributed spatiotemporal positions x, y, t by random values that were drawn independently and uniformly distributed from the range of intensity values. This means the method is designed for addressing *single* pixel noise. Therefore, in *PAMONO* data analysis it is applied *before* the spatially extended smoothing filters from Section 5.3.1, to prevent the average or Gauß filter from "smearing" single pixel noise across the image plane. Note that the *PAMONO* signal model from Chapter 4 does not contain any sources of random impulse noise. Instead, fuzzy denoising serves as a filter for spatiotemporal outliers that can arise due to the low SNR and the division by small values during background elimination.

The key ideas behind the method can be summarized as follows: Within the time series of input images $I(x, y, t)$, pixels (x, y, t) are classified into two fuzzy sets: noisy and noise-free pixels. Each of the two sets is defined as a combination of several fuzzy rules that exploit spatial as well as temporal information from a local spatiotemporal window around (x, y, t). Hence correlations within and between images are taken into account in assigning fuzzy set memberships. Pixel classification works as follows: The fuzzy rules that define the noisy and noise-free pixel sets are evaluated. This assigns to each pixel a degree of fuzzy set membership in the noisy and in the noise-free set. These fuzzy set memberships are turned into a crisp binary classification by assigning to each pixel the class for which the fuzzy set membership is larger. Then, for each pixel that was assigned to the noisy class, the pixel value is replaced by a new value attained from filtering *only noise-free* pixel values in a local spatiotemporal neighborhood. This scheme is iterated in successive steps, progressing from removing more obvious to less obvious noise, using specifically adapted fuzzy rules in each step. This iterative refinement strategy preserves small details, which is important for *PAMONO* data analysis due to the small spatial extension of nano-object adhesions.

Melange, Nachtegael, and Kerre in their original work [MNK11] were concerned with denoising color videos showing natural scenes. Hence their approach is designed for multi-channel input data and incorporates motion compensation by block matching. *PAMONO* sensor videos, however, are recorded in grayscale and exhibit no motion: Nano-objects appear at certain locations that remain fixed from the moment of adhesion. Hence, for *PAMONO*, the fuzzy rules assuming multichannel data are omitted, and motion compensation is skipped. As a result, the iterative refinement scheme described in the following consists of two instead of three steps because the second step from [MNK11], using color information for detecting noisy pixels in moving objects, does not apply to *PAMONO*.

First Iteration

In the first iteration, spatial and temporal consistency with neighboring pixels are evaluated, fuzzily assigning each pixel to the noise-free, respectively noisy class. Consistency is measured in terms of absolute intensity difference d. This difference is fuzzified into the fuzzy set of large

values: Given a lower and an upper fuzzy threshold $l_1, l_2 \in \mathbb{R}_{\geq 0}$ with $0 \leq l_1 < l_2$, membership to the fuzzy set of large values (cf. plot in Figure 5.5) is defined as

$$\zeta_{\text{isLarge}}^{l_1,l_2}(d) = \begin{cases} 1 & \text{for } d \geq l_2 \\ \frac{d-l_1}{l_2-l_1} & \text{for } l_1 < d < l_2 \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

Hence an intensity difference d can be definitely large ($d \geq l_2$), definitely not large ($d \leq l_1$), or something fuzzy in between ($l_1 < d < l_2$). The lower and upper fuzzy thresholds l_1 and l_2 are the parameters to be optimized for fuzzy denoising, along with a boolean $b_{\text{fuzzy}}^{\text{denoise}} \in \{0, 1\}$, deciding whether to apply fuzzy denoising at all.

In order to facilitate a semi-formal statement of the fuzzy rules in terms of continuous text, the following definitions are made:

Definition 5.2. *The property of being large in the sense of Equation (5.8) is in continuous text referred to as being $LARGE_f$. Furthermore, a AND_f b denotes the fuzzy conjunction of two fuzzy set memberships a and b. Following [MNK11], the conjunction is defined as a AND_f $b \coloneqq \min(a, b)$. Analogously, a OR_f b is the fuzzy disjunction, defined as a OR_f $b \coloneqq \max(a, b)$. Fuzzy negation is defined as NOT_f $a \coloneqq 1 - a$.*

Given these prerequisites, the fuzzy rule defining the set of noise-free pixels states that a pixel (x, y, t) is regarded as noise-free, if the absolute intensity difference d to its temporal predecessor $(x, y, t-1)$ is NOT_f $LARGE_f$, AND_f the absolute intensity difference to its temporal successor $(x, y, t+1)$ is NOT_f $LARGE_f$, AND_f there are at least two spatial neighbors (x_1, y_1, t) and (x_2, y_2, t) within a radius of two pixels on the same image t, to which the absolute intensity difference is NOT_f $LARGE_f$. In short: A pixel is considered noise-free, if it is not an outlier within a local spatiotemporal window. A detailed *formal* statement of these and all following fuzzy rules can be found in [MNK11].

Conversely, the fuzzy rule defining the set of noisy pixels states that a pixel (x, y, t) is regarded as noisy, if the absolute intensity difference to $(x, y, t-1)$ AND_f $(x, y, t+1)$ is $LARGE_f$, AND_f there are at least two spatial neighbors with $LARGE_f$ intensity difference to (x, y, t), but NOT_f with $LARGE_f$ intensity difference to each other. In short: (x, y, t) is an outlier/impulse, both temporally and spatially.

After that, the noise-free and noisy set are turned to a crisp binary classification, by assigning pixels to the class associated with the larger fuzzy set membership. Noise-free pixels remain unchanged, while pixels classified as noisy are filtered *solely* on the basis of the noise-free pixels. Filtering works as follows: Each noisy pixel (x, y, t) is replaced by the mean of the corresponding pixels $(x, y, t-1)$ and $(x, y, t+1)$ in the previous and subsequent image, if both are not noisy. Otherwise it is replaced by the median of the noise-free pixels in a window with radius two in the same image.

Second Iteration

A successive refinement of the results from the first iteration is computed in the second iteration. This time the rule defining the fuzzy set of noisy pixels states that a pixel (x, y, t) is considered noisy, if the absolute intensity difference d between pixel (x, y, t) in the result from the first iteration, and pixel $(x, y, t-1)$ in the previous, already fully denoised image is

$LARGE_f$, AND_f the difference between the fully denoised $(x, y, t-1)$ and the unprocessed next image pixel $(x, y, t+1)$ is NOT_f $LARGE_f$. This rule identifies isolated noise pixels that were either not detected or filtered to a bad value in the first iteration. It is the first part of the noisy set rule, to which the following second part is linked via fuzzy OR_f: If for the first iteration result, pixel (x, y, t) resides in an area of homogeneous intensities, i.e. the intensity difference between the second largest and the second[3] smallest intensity in a 3×3 window is NOT_f $LARGE_f$, AND_f pixel (x, y, t) exhibits a large intensity difference to the second smallest OR_f to the second largest element, then (x, y, t) is considered noisy. This rule regards only homogeneous regions, and marks pixels as noisy that deviate largely from their homogeneous neighborhood.

In the second iteration, there is no rule for the noise-free set. Instead, any pixel for which the noisy rule is not equal to one, i.e. the noisy conditions are not fully fulfilled, is considered noise-free.

Filtering in the second iteration is carried out the same way as in the first one, but using the images from that iteration as input, instead of the original input images.

5.3.3 Application-Specific Cleaning Heuristics

Besides the rather general denoising techniques described in the preceding part of this section, the peculiarities of *PAMONO* data analysis demand for additional, more application-specific denoising techniques, which will be presented now.

Brightness Correction

Fluctuations in illumination and temperature of the gold sensor surface, as well as heating of the CCD chip can cause global variations in the brightness of the recorded images. These variations affect all pixels, and in the worst case, they cause a step-like temporal signature in all recorded time series. Brightness correction operates per image and, if enabled, addresses this issue by enforcing the same mean intensity value for each image. The increase in mean intensity that is due to attaching nano-objects is negligibly small and hence does not pose a problem. Brightness correction is applied before *all* other denoising methods, such that image-wide intensity outliers and temporal trends have already been sorted out before fuzzy spatiotemporal denoising and the other filters are applied. The parameter optimized with respect to brightness correction is the boolean switch $b_{\text{bright}}^{\text{denoise}} \in \{0, 1\}$, turning it on and off.

Pixel Overspilling Compensation

As the capabilities of the *PAMONO* sensor live off the measurement of *increases* in intensities, saturated/overexposed pixels in the CCD chip are to be avoided. On the other hand, in order to fully exploit the intensity resolution of the CCD, underexposure is undesirable as well. Therefore, before any measurement, the CCD is configured such that the minimum measured intensity is close to zero, and the maximum measured intensity is (less) close to the value the CCD outputs at the full well capacity of a pixel. Now given this scenario, due to sensor heating and the possibility of adverse laboratory conditions, complete avoidance of saturated pixels can not always be guaranteed. Therefore it is desirable to include a heuristic into the pattern detector that alleviates the negative effects of saturated pixels.

[3] The first place on both ends is reserved for the possibly noisy pixel.

These negative effects arise in the pixels *neighboring* the saturated pixels: Shaking or vibration of the *PAMONO* sensor during measurement may cause the CCD to be displaced with respect to the gold surface. As a consequence, the saturated pixels may spill over to their unsaturated neighbors, yielding a step function-like temporal profile in the raw sensor data. This is undesired because it is a step function not associated with a nano-object adhesion and thus a potential candidate for an FP detector response.

Pixel overspilling compensation addresses this problem: Saturated pixels are detected and these pixels, along with their neighbors, are excluded from pattern detection. Albeit being listed as a denoising method, the pixels excluded by pixel overspilling compensation are determined prior to background elimination because the saturated pixels are identified on the basis of the original values measured by the CCD sensor. The parameter to be optimized with respect to pixel overspilling compensation is the boolean switch $b_{\text{spill}}^{\text{denoise}} \in \{0, 1\}$, turning it on and off.

5.3.4 Parameters

The parameters that are optimized for the denoising module, along with the examined ranges of values are:

Spatial Denoising Techniques

$b_{\text{avg}}^{\text{denoise}} \in \{0, 1\}$
: a boolean switch turning the spatial averaging filter on/off,

$K_{\text{avg}}^{\text{w}} \in \{1, \ldots, 7\}$
: an integer determining the averaging filter kernel width,

$K_{\text{avg}}^{\text{h}} \in \{1, \ldots, 7\}$
: an integer determining the averaging filter kernel height,

$b_{\text{Gauß}}^{\text{denoise}} \in \{0, 1\}$
: a boolean switch turning the spatial Gauß filter on/off,

$\sigma_{\text{Gauß}} \in [1.5, 3.5]$
: a float determining the standard deviation of the Gauß filter kernel,

$b_{\text{med}}^{\text{denoise}} \in \{0, 1\}$
: a boolean switch turning the spatial median filter on/off,

$K_{\text{med}}^{\text{w}} \in \{1, \ldots, 7\}$
: an integer determining the median filter kernel width,

$K_{\text{med}}^{\text{h}} \in \{1, \ldots, 7\}$
: an integer determining the median filter kernel height,

Fuzzy Spatiotemporal Denoising

$b_{\text{fuzzy}}^{\text{denoise}} \in \{0, 1\}$
: a boolean switch turning the fuzzy denoising filter on/off,

$l_1 \in [0.02, 0.1]$
: a float containing the lower soft threshold of fuzzy denoising, determining when an intensity difference is considered to be small,

$l_2 \in [0.1, 0.2]$
: a float containing the upper soft threshold of fuzzy denoising, determining when an intensity difference is considered to be large,

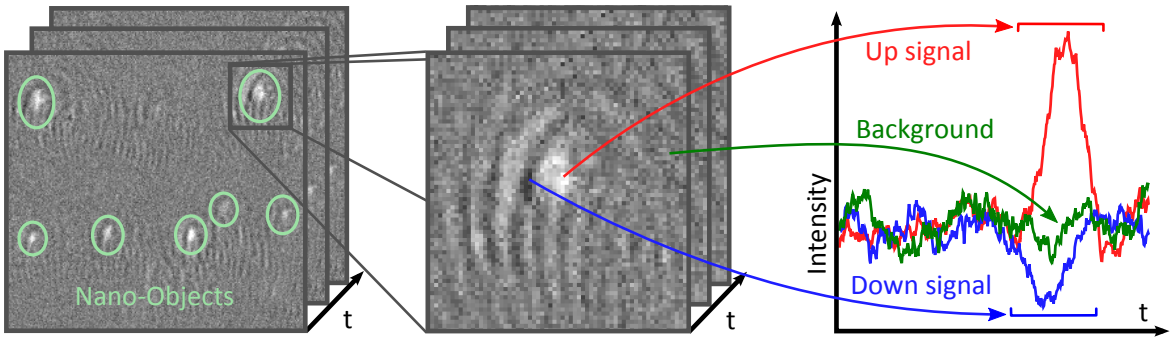


Figure 5.6: Classes of Time Series. In the light center of a nano-object adhesion, upward step functions are measured by the sensor. The background elimination response of such a function is an upward ramp-like function, belonging to the **up signal** class (red). Surrounding the center of a nano-object adhesion there are dark concentric rings, resulting in downward step functions with background elimination responses that belong to the **down signal** class (blue). Areas not affected by a nano-object adhesion result in noise-only responses, belonging to the **background** class (green). The term 'background' alludes to the fact that these time series are not affected by an object of interest. Figure adapted from [SFL+14].

Application-Specific Cleaning Heuristics

$b^{\text{denoise}}_{\text{bright}} \in \{0, 1\}$
: a boolean switch turning brightness correction on/off,

$b^{\text{denoise}}_{\text{spill}} \in \{0, 1\}$
: a boolean switch turning pixel overspilling compensation on/off.

5.4 Time Series Classification via Fuzzy Template Matching

Time series classification receives the denoised $T \cdot A$ estimate as input, cf. Figure 5.1. $T \cdot A$ is the product signal of the target patterns component caused by nano-object adhesions to the sensor surface and the artifacts component caused by imperfections of measurement. The purpose of time series classification is finding time series in $T \cdot A$ that are similar to step responses of the preceding processing modules and that are thus candidates for being affected by a nano-object adhesion. These candidate time series share the property of being *salient* due to their step response-like structure. Nevertheless, time series classification makes no statement about whether saliency in the candidate time series is caused by an event in T or in A. The output of time series classification is a binary class mask Γ, serving as input to the segmentation module described in Section 5.6.

As discussed in more detail in Chapter 4, nano-objects attaching to the surface of the *PAMONO* sensor manifest as upward and downward step functions in the time series of intensities observed at pixels in the affected areas: The central part of nano-object adhesions is characterized by upward step functions, surrounded by downward step functions in the outer parts. Time series classification serves to determine *whether* and *when* a time series observed at a certain sensor position is affected by a nano-object adhesion. As discussed in Section 5.2.3, the result of applying background elimination to a time series is called the response of background elimination to that input time series. For a step function, as arising around attaching nano-objects, the background elimination response in $T \cdot A$ is a ramp-like

function. Three classes of response time series are distinguished, the names and semantics of which are determined as follows:

Terminology 5.1. *A background elimination response obtained for an upward step function, as arising in the light center of a nano-object adhesion, belongs to the* ***up signal*** *class, cf. red time series in Figure 5.6. Analogously, a response obtained for a downward step function, as arising in the dark concentric rings around a nano-object adhesion, belongs to the* ***down signal*** *class, cf. blue time series in Figure 5.6. Any response obtained for a time series from an area not affected by a nano-object adhesion belongs to the* ***background*** *class, cf. green time series in Figure 5.6.*

The background class is not to be confused with the temporally constant, high amplitude background component B from the signal model in Equation (4.1) that was removed by background elimination. The intuition behind also calling this class of time series 'background' originates in conventional photography, where those parts of an image that do not show the objects of interest are called background.

Note that the ramp functions defining the up and down signal classes have finite support, so they can be detected using a sliding window of finite length in a template matching approach. Given the temporal coordinate of the starting point of a ramp function in the data, the temporal onset of the nano-object adhesion can be inferred as according to Section 5.2.3.

The time series classification method to be presented in this section is a real-time-capable streaming method, running on the GPU [LST+13a; LST+13b]. It consists of three parts:

1. A preselection of time series is computed, based on applying hard thresholds to intensity magnitudes. It decides which time series undergo the computationally more expensive subsequent processing. Any time series not selected here is immediately classified as background. This part is the topic of Section 5.4.1.
2. Matching scores between the previously selected observed time series and an ideal template time series are computed. This part is described in Section 5.4.2.
3. Fuzzy rules are applied to the previously computed matching scores, classifying the selected time series as either up/down signals or background, producing the binary class mask $\Gamma(x, y, z)$. A distinction between the up and down signal class is not made because both are related to nano-object adhesions. This part is detailed in Section 5.4.3.

5.4.1 Time Series Preselection

Classifying the observed time series into the classes from Terminology 5.1 is conducted in a sliding window fashion. In the following, time series are regarded individually per image pixel, so to avoid cluttered notation, the x, y coordinates will be dropped. In addition, as only excerpts from a fixed-length sliding window are considered, these excerpts are in the following written as vectors, facilitating notation of mathematical operations applied to them. Hence $\mathbf{v} \in \mathbb{R}^T$ denotes the time series $(T \cdot A)(x_c, y_c, t_a \dots t_b)$ observed at a certain image pixel (x_c, y_c), within a sliding window from t_a to t_b. The T in the dimension of $\mathbf{v} \in \mathbb{R}^T$ is the length of the sliding window and is not to be confused with the target patterns signal in $(T \cdot A)(x, y, t)$. Vector component v_1 contains the value of $(T \cdot A)(x_c, y_c, t_a)$, while v_T contains the value of $(T \cdot A)(x_c, y_c, t_b)$. The sliding window length T is subject to optimization and simultaneously determines the size of the ideal template time series used in computing the matching score, cf. Section 5.4.2.

Given these prerequisites of notation, time series preselection is a hard thresholding method, applied to each time series $\mathbf{v}$ observed in the input. Global thresholding can be applied to this input because background elimination removed the multiplicative influence of B, hence making the time series in the $T \cdot A$ estimate comparable in terms of their intensity. A lower threshold $h_1 \in \mathbb{R}_{\geq 0}$ and an upper threshold $h_2 \in \mathbb{R}_{\geq 0}$ are applied to a measure of the intensity difference occurring in $\mathbf{v}$. This intensity difference d is computed as

$$d = \max(\{v_i \mid i \in \{1, \ldots, T\}\}) - \min(\{v_i \mid i \in \{1, \ldots, T\}\}). \tag{5.9}$$

If for a given time series $\mathbf{v}$ it holds that $h_1 \leq d \leq h_2$, it is selected for further processing, and if not, it is classified as a background time series. An intensity difference residing between the two thresholds is a necessary but not a sufficient condition for a time series to belong to the up or down signal class. If this condition is violated because d is too low, there can be no up or down signal in the time series due to low contrast. If it is violated because d is too high, there might be an up or down signal in the time series but from its excessively large magnitude, it can be inferred that it is not related to a nano-object but to a different cause, e.g. an air bubble, or sensor jitter from a concussion. Therefore, the time series is classified as background. If the necessary condition is not violated, no classification can be made because time series structure has not been considered yet. Structure gives a sufficient condition for time series belonging to the up or down signal class and is considered in the subsequent template matching in Section 5.4.2.

Summing up, the purpose of time series preselection is two-fold:

1. Only time series with a certain contrast pertaining to nano-object adhesions are kept. In this process time series *magnitude* is decisive, imposing a necessary condition for up or down signals. In the subsequent template matching, time series *structure* is decisive, constituting a sufficient condition for up or down signals.
2. Early removal of most time series belonging to the background class saves computational effort because these time series do not undergo the subsequent processing. This pays off because large parts of the sensor surface belong to the background class, as the patterns caused by nano-objects only have small spatial and temporal extension.

5.4.2 Matching Score

Time series preselection classified part of the observed time series as belonging to the background class and ensured that the remaining time series exhibit magnitudes that are characteristic for nano-object adhesions. These remaining time series are examined further by computing a matching score, based solely on temporal structure, measuring similarity of the observed time series to an up or down signal. As both, up and down signals are indicative of a nano-object adhesion, these two classes are no longer distinguished, and a matching score is devised that attains high values in both cases. An ideal template time series $\mathbf{t} \in \mathbb{R}^T$ is used to match them to. Such a template is determined by applying background elimination to an ideal upward step function, using the same parameters b^{bg}, w^{ρ}, w^{ϕ} as for the observed input time series. This yields the noise-free, ideal response of background elimination to an upward step function and thus an ideal time series belonging to the up signal class from Terminology 5.1. Section 5.2.3 gives a more detailed discussion of this topic and an illustration in Figure 5.2.

The matching score m for an individual observed time series $\mathbf{v}$ is computed as its absolute cosine similarity to the ideal template $\mathbf{t}$:

$$m = \left| \frac{\langle \mathbf{v}, \mathbf{t} \rangle}{\|\mathbf{v}\| \, \|\mathbf{t}\|} \right|. \tag{5.10}$$

Here $\langle \circ, \circ \rangle$ denotes the dot product of vectors, $\|\circ\|$ is the Euclidean norm of a vector, and $|\circ|$ is the absolute value of a scalar. The argument of the absolute value function corresponds to the cosine between the vectors $\mathbf{v}$ and $\mathbf{t}$, hence the name 'cosine similarity'. If $\mathbf{v}$ is an ideal up signal like $\mathbf{t}$, then $m = 1$ because the vectors are parallel. This property does not change if $\mathbf{v}$ is scaled, so m is independent of the magnitude of the values in $\mathbf{v}$. This is desirable because time series differing solely in magnitude thus attain the same matching score, and time series preselection already ensured that the time series for which m is computed exhibit magnitudes from the interval that is indicative of nano-object adhesions. If $\mathbf{v}$ is an ideal down signal, the vectors are again parallel, but point in opposite directions. Consequently their cosine becomes -1, making $m = 1$. Time series vectors $\mathbf{v}$ with different orientation attain lower matching scores and m decreases with increasing difference in orientation. The time series belonging to the background class that have passed preselection did so because their magnitude is from the interval indicative of a nano-object adhesion. However, their structure, respectively their orientation if written as a vector $\mathbf{v}$, is different from the template $\mathbf{t}$, cf. Figure 5.6. Hence they attain lower values in m than time series belonging to the up or down signal class. The lowest value of an absolute cosine and thus the lowest possible m is zero, which is attained for $\mathbf{v}$ that are orthogonal to $\mathbf{t}$. Consequently, the matching score m is in $[0, 1]$, invariant to time series magnitudes, and attains high values for time series $\mathbf{v}$ that are similar to either up or down signals. Invariance to time series magnitude is a desirable property because down signals usually exhibit smaller magnitudes than up signals. Making m depend solely on time series structure and not on magnitude enables treating both, up and down signals, the same way during further processing like the hard respectively soft thresholding on m to be presented in Section 5.4.3.

Computing the matching score m is done within the same sliding window of length T used in time series preselection, and the results of both define the spatiotemporal volume $M(x, y, t)$ of *all* matching scores m. Ignoring the cases where the sliding window crosses the temporal boundaries[4] of the volume $I(x, y, t)$ of sensor images, there is a one-to-one correspondence between the coordinates (x, y, t) in I and M. Therefore the overall process of preselection and matching score computation is as follows: The sliding window is placed at each (x_c, y_c, t_c) in the domain of the denoising result. If preselection classifies the time series observed at (x_c, y_c, t_c) as belonging to the background class, $M(x_c, y_c, t_c)$ is set to zero and the next coordinate is processed. Otherwise the matching score m is computed and assigned to $M(x_c, y_c, t_c)$, and then the next coordinate is processed. This process defines the spatiotemporal volume M of matching scores, measuring similarity of time series after background elimination and denoising to an ideal template function indicative of nano-object adhesions. High values in M are attained for both, the up and down signal class from Terminology 5.1, while the background class attains low values.

[4] For the boundary cases, M is set to zero.

5.4.3 Fuzzy Time Series Classification

Fuzzy[5] time series classification [LST+13a; LST+13b] is a variant of the noisy versus noise-free pixel classification used in the fuzzy denoising procedure from Section 5.3.2. It uses the same GPU implementation but defines different fuzzy rules, in order to distinguish the two classes of interest: These two classes are the background class from Terminology 5.1 on the one hand, and the union of the up and down signal class on the other. Up and down signals are not distinguished because they are both indicative of nano-objects attaching to the sensor surface.

Fuzzy classification takes the continuous matching scores $M(x, y, t)$ from the previous section as input. For each coordinate (x, y, t), matching scores from a local neighborhood around (x, y, t) are considered via fuzzy rules that are custom-tailored for the *PAMONO* application case. Hence the spatiotemporal information in *PAMONO* data is exploited. After converting fuzzy set memberships to crisp decisions, the output is a robust, binary per-pixel classification $\Gamma(x, y, t) \in \{0, 1\}$, assigning up and down signals to the positive class and background pixels to the negative class. This classification is then passed to the segmentation module described in Section 5.6.

Matching scores $M(x, y, t)$ are from the interval $[0, 1]$ and hence need no extra fuzzification. However, analogously to the definition of $\zeta_{\text{isLarge}}^{l_1,l_2}$ in fuzzy denoising (cf. Equation (5.8)), they are assigned an initial fuzzy set membership by applying two soft thresholds. All other fuzzy set memberships are derived from this initial membership. For shorter notation, again m is used to denote the matching score $m = M(x_c, y_c, t_c)$ at a certain spatiotemporal coordinate (x_c, y_c, t_c). Given these prerequisites, the initial fuzzy set membership function is defined as

$$\zeta_{\text{isPos}}^{s_1,s_2}(m) = \begin{cases} 1 & \text{for } m \geq s_2 \\ \frac{m-s_1}{s_2-s_1} & \text{for } s_1 < m < s_2 \\ 0 & \text{else,} \end{cases} \tag{5.11}$$

with soft thresholds $s_1, s_2 \in \mathbb{R}_{\geq 0}$. This set fuzzily characterizes the degree by which matching score m is regarded as belonging to the positive class, consisting of up/down signals indicative of a nano-object. The graph of $\zeta_{\text{isPos}}^{s_1,s_2}$ looks like that of $\zeta_{\text{isLarge}}^{l_1,l_2}$ in Figure 5.5, but with the soft denoising thresholds l_1 and l_2 replaced with the soft classification thresholds s_1 and s_2. Set membership increases with increasing m. The two soft thresholds s_1 and s_2 are subject to optimization.

Further application-specific fuzzy rules that build on the $\zeta_{\text{isPos}}^{s_1,s_2}$ set from Equation (5.11), evaluated for each coordinate (x, y, t) and a local neighborhood around it, will be presented in the following [LST+13a; LST+13b]. These rules formalize domain knowledge, obtained from examining the spatiotemporal structure of nano-object adhesions. Their formal statement relies on a number of prerequisites which will be given now.

Formal Prerequisites

- In the following statement of fuzzy rules, the fuzzy operators AND_f, OR_f and NOT_f from Definition 5.2 are used.

[5] Terminology associated with fuzzy set theory was introduced as part of fuzzy denoising, cf. the beginning of Section 5.3.2.

- For a given spatiotemporal coordinate (x_c, y_c, t_c), the local neighborhood set $L^{X,Y,T}$ is defined as the multiset of all matching score values $M(x, y, t)$ that occur in a local cuboid with side-lengths X, Y, T in the horizontal, vertical and temporal directions, placed in M with its *center* at (x_c, y_c, t_c).
- Furthermore, $\Lambda(k, S)$ denotes the k-th largest element in a multiset S.

Fuzzy Rules

Using these prerequisites and the fuzzy set $\zeta^{s_1,s_2}_{\text{isPos}}$ from Equation (5.11), a further fuzzy rule characterizing the positive class can be stated. Its goal is to make the detection of pixels at the fringe of a nano-object adhesions more robust:

$$\zeta^{s_1,s_2}_{\text{fringe}}(m) = m > s_1 \quad AND_f \quad \zeta^{s_1,s_2}_{\text{isPos}}(\Lambda(15, L^{7,7,3})). \tag{5.12}$$

Note that the same implicit coordinates (x_c, y_c, t_c) are used for $m = M(x_c, y_c, t_c)$ and for the center of the local neighborhood $L^{7,7,3}$. This rule accounts for the fact that the magnitude of the Surface Plasmon Resonance (SPR) effect causing the up/down signals around nano-object adhesions decreases with distance from the point of adhesion. Hence, time series observed at the fringe of the effect exhibit up/down signals with lower magnitude (cf. center part of Figure 5.6 and compare pixel brightness of inner versus outer part of nano-object adhesion) but corrupted by the same level of noise as all other time series. Therefore, their SNR is lower as e.g. in time series observed in the center of the effect. Lower SNR means a lower matching score m. The rule in Equation (5.12) allows such time series to "rehabilitate", based on their local neighborhood: If $m > s_1$, i.e. if at least the lower fuzzy threshold is met, the time series membership in $\zeta^{s_1,s_2}_{\text{fringe}}$ is assigned the fifteenth-highest membership in $\zeta^{s_1,s_2}_{\text{isPos}}$ observed in a $7 \times 7 \times 3$ local neighborhood. This remedies mediocre matching scores for time series that reside in a neighborhood with a sufficient number of positive pixels. This is the case for time series observed at the fringe of the SPR effect, but not for time series in background areas that attained a matching score $m > s_1$ by chance. Therefore, Equation (5.12) makes detection more robust by promoting fringe time series. The same effect could be achieved in global hard thresholding by lowering the threshold. However, in contrast to that, the fuzzy fringe rule reduces the unwanted "by-catch" of background time series that would be incurred by lowering a global hard threshold without considering local neighborhoods.

The next fuzzy rule for the positive class synchronizes matching scores for multiple detections of the same nano-object over time:

$$\zeta^{s_1,s_2}_{\text{time}}(m) = m > s_1 \quad AND_f \quad \zeta^{s_1,s_2}_{\text{isPos}}(\Lambda(1, L^{1,1,5})). \tag{5.13}$$

It does so by assigning any time series with $m > s_1$ a membership in $\zeta^{s_1,s_2}_{\text{time}}$ that is the highest membership in $\zeta^{s_1,s_2}_{\text{isPos}}$, observed over five consecutive sensor images. This realizes a temporal dilation[6] of the matching scores in M, filling temporal "holes" and thus increasing robustness to noise.

With these three rules for the positive class stated in Equations (5.11) to (5.13), the overall membership to the fuzzy set of up $\cup$ down signals is determined as their fuzzy disjunction:

$$\zeta^{s_1,s_2}_{\text{up}\cup\text{down}}(m) = \zeta^{s_1,s_2}_{\text{isPos}}(m) \quad OR_f \quad \zeta^{s_1,s_2}_{\text{fringe}}(m) \quad OR_f \quad \zeta^{s_1,s_2}_{\text{time}}(m). \tag{5.14}$$

[6]The term 'dilation' and other morphological operators are discussed in more detail in Section 5.6.1.

A final rule is required for the background class, then all classes in Figure 5.6 are captured by the fuzzy classifier. This rule is defined as

$$\zeta^{s_1,s_2}_{\text{background}}(m) = m < s_2 \;\; AND_f \;\; NOT_f \;\; \zeta^{s_1,s_2}_{\text{isPos}}(\Lambda(10, L^{7,7,3})). \tag{5.15}$$

It states that background time series have matching scores lower than s_2 and that their degree of membership to the background increases, depending on their neighborhood: The lower the $\zeta^{s_1,s_2}_{\text{isPos}}$ set membership of the tenth highest neighboring matching score, the higher the membership to the background set. Time series with mediocre matching scores below s_2 are thus assigned high membership to the background set if their score is an outlier in a neighborhood of low matching scores. This removes isolated islands that are smaller than the SPR effect caused by an attaching nano-object.

Like in the fuzzy denoising method from Section 5.3.2, the set memberships $\zeta^{s_1,s_2}_{\text{up}\cup\text{down}}$ and $\zeta^{s_1,s_2}_{\text{background}}$ from Equations (5.14) and (5.15) are defuzzified by assigning to each time series the class with higher degree of set membership, yielding the binary spatiotemporal class mask $\Gamma(x, y, z) \in \{0, 1\}$. In Γ, time series belonging to the positive (up $\cup$ down signal) class are indicated by value 1, while those belonging to the negative (background) class are indicated by value 0. This class mask Γ is input to the segmentation module described in Section 5.6.

Summing up, fuzzy time series classification avoids using a single global hard threshold, but classifies pixels using two soft thresholds s_1, s_2, while integrating information from local neighborhoods in all three dimensions of the input data. It aims at making the positive areas in the class mask Γ more robust against the low SNR in the underlying sensor images. This in turn makes the shape of polygons to be created by the segmentation module more robust [LST+13a] and therefore more consistent for polygons delineating nano-objects. More robust polygon shapes are beneficial in computing the per-polygon features to be described in Section 6.2: These per-polygon features are computed e.g. as polygon shape descriptors or as statistics of intensities observed in the area covered by the polygons and are used to separate true from false positive detector responses.

The parameters arising in fuzzy time series classification are the two soft thresholds s_1, s_2 defining the $\zeta^{s_1,s_2}_{\text{isPos}}$ set from Equation (5.11). A third parameter $b^{\text{classify}}_{\text{fuzzy}} \in \{0, 1\}$ controls whether fuzzy time series classification is enabled (value 1) or disabled (value 0). In case fuzzy time series classification is disabled, the global hard thresholding described in the following is used instead.

Alternative to Fuzzy Rules: Global Hard Thresholding

If $b^{\text{classify}}_{\text{fuzzy}}$ is set to zero, the binary class mask Γ is determined by applying a global hard threshold to the matching scores M. In order to avoid introducing a further parameter to be optimized, the fuzzy upper threshold s_2 assumes the role of this hard threshold: For $M(x, y, t) \geq s_2$, $\Gamma(x, y, t)$ is set to one, otherwise it is set to zero.

5.4.4 Parameters

The parameters that are optimized for the fuzzy time series classification module, along with the examined ranges of values are:

Time Series Preselection

$T \in \{8, \ldots, 32\}$
: an integer determining the length of the temporal sliding window within which time series are classified (which also determines the length of the matched template and affects temporal resolution),

$h_1 \in [0.0005, 0.1]$
: a float defining the lower boundary of the interval of time series magnitudes that are accepted by hard thresholding,

$h_2 \in [0.01, 0.2]$
: a float defining the upper boundary of the interval of time series magnitudes that are accepted by hard thresholding,

Fuzzy Time Series Classification

$b_{\text{fuzzy}}^{\text{classify}} \in \{0, 1\}$
: a boolean that switches between fuzzy time series classification (value 1) and hard thresholding of matching scores (value 0),

$s_1 \in [0.002, 0.8]$
: a float defining the lower soft threshold of fuzzy time series classification,

$s_2 \in [0.05, 1]$
: a float defining either the upper soft threshold of fuzzy time series classification (in case $b_{\text{fuzzy}}^{\text{classify}} = 1$) or the single global hard threshold applied to matching scores (in case $b_{\text{fuzzy}}^{\text{classify}} = 0$).

5.5 Time Series Classification via Translation-Invariant (TI) Wavelet Features

Time series classification via translation-invariant (TI) wavelet features is an alternative to time series classification via fuzzy template matching as presented in the previous section. Input, output and the classification task are the same: Time series in the denoised $T \cdot A$ estimate are to be classified, separating those that are affected by a nano-object adhesion (up or down signals) from those that are not (background), cf. Terminology 5.1 and Figure 5.6. Classification results can again be phrased in terms of a spatiotemporal class mask Γ, serving as input to the segmentation module described in Section 5.6.

The methodological approach, however, differs strongly from fuzzy template matching. It was first proposed in [SFL+14], upon which this section is based. Instead of a classifying time series by defuzzifying fuzzy set memberships that are based on the single feature 'matching score', the approach to be presented now uses a condensed k-Nearest Neighbors (k-NN) classifier [ZLX09] that operates on multiple features, computed from wavelet coefficients [CD95]. Training the classifier works as follows: Signal synthesis as presented in Chapter 4 generates a large set of ground truth class-labeled time series. From these, class-labeled wavelet feature vectors are extracted for training. Data reduction is carried out by coreset-based k-Means clustering [FGS+13] of the feature vectors, condensing the large synthetic training set, while maintaining its neighborhood structure in feature space. Coresets [HM04] enable handling bigger datasets with clustering, while clustering enables handling bigger datasets

with the lazily learning k-NN classifier. To restate this more specifically for the *PAMONO* scenario, coresets allow clustering big training sets, synthesized from multiple *PAMONO* datasets, such that a general k-NN classifier can be trained. Clustering serves to accelerate the k-NN classifier, which operates on the cluster centers only. This classifier can store knowledge from multiple datasets, and it is able to generalize to further unseen *PAMONO* datasets as demonstrated in Section 5.5.4. The output of the training procedure is a supervised classifier (k-NN), created from synthetic ground truth that underwent unsupervised data reduction (k-Means).

The data processed by the classifier are feature vectors computed from wavelet coefficients of the time series to be classified. Using wavelet-based features follows the goal of achieving better classification quality and increased robustness to noise. These goals are pursued in the presented feature extraction by exploiting the following idea: Wavelets separate a signal into multiple scales, each capturing another level of signal detail. This multiscale nature of the wavelet transform is utilized to separate noisy scales from signal scales, and features are computed for each scale, cf. Section 5.5.1. Feature selection can then be applied to consider only features classifying the data well, i.e. features originating from signal scales, cf. Section 5.5.2.

In feature extraction for classification, it is desirable that features are invariant to any information that does not relate to the class label. In *PAMONO* time series classification, the point in time when a nano-object attaches to the sensor surface does not relate to the class of the time series: If an up or down ramp function occurs in the time series, it belongs to the up or down signal class, independent of the location of the ramp on the temporal axis, cf. Figure 5.6. Wavelet decomposition enables extraction of features that are invariant to this information. Different temporal locations of the ramp-like part in an up or down signal can be regarded as different translations of the ramp across the temporal axis. Translation-invariant (TI) wavelets are, in the sense defined in Section 5.5.1, invariant to such translations, and thus provide a foundation for computing TI features. A feature extraction based on such features has the desirable property that time series affected by nano-objects map to similar points in feature space, independent of the time the nano-objects appear. This property furthermore bears the advantage that it is no longer necessary to evaluate the operations applied in the sliding window for *every* temporal coordinate, as was done in fuzzy time series classification. Instead, since not every possible location of the ramp-like part has to be covered, it is possible to advance the sliding window by its length, conducting the classification procedure only for each such block of sensor images. This decreases temporal resolution but also computational burden.

Combining the idea of condensed k-NN with TI feature extraction yields a method for *PAMONO* time series classification exhibiting parameters that do not require optimization, as long as their values are chosen 'large enough', cf. Section 5.5.4. This is a considerable advantage over the fuzzy method from Section 5.4. Furthermore, the method does not require an ideal template as input, but learns the class concepts from classified training data. As a last point, the classifier can be trained with data from multiple *PAMONO* measurements, which can improve its generalization performance to other unseen *PAMONO* datasets.

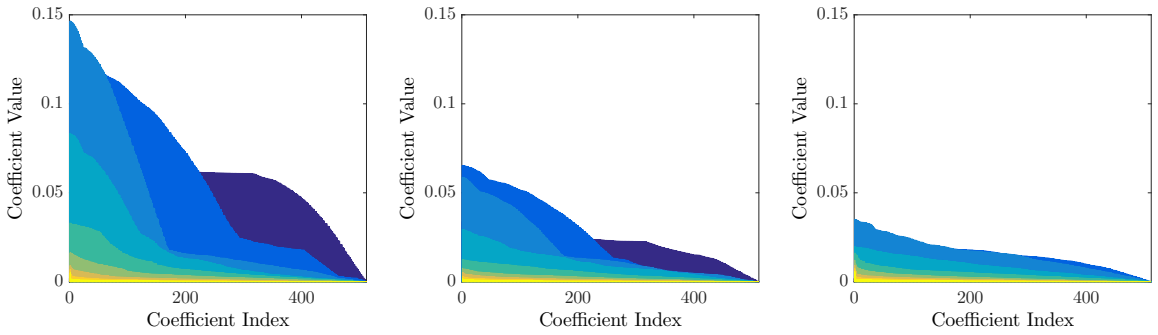
Related Work

Condensing a training set by removing inherent redundancies and similarities is a commonly applied technique to speed up classification for lazy learners like k-NN [GK79; Alp97; Ang05]. Clustering is one approach for data reduction, and it has been previously applied for condensing k-NN training data in text classification [ZLX09]. The approach presented in the following combines this idea with the coreset method, introducing a further stage of data reduction that is applied before clustering: The BIRCH Meets Coresets (BICO) algorithm [FGS+13] is used to reduce the input of clustering to a smaller coreset, to which weighted clustering is applied. The result of weighted clustering of the small coreset is an approximation to the clustering result for the entire dataset. Coresets thus enable clustering of big datasets that are too large for exact clustering algorithms. The impact of the approximate nature of the coreset method on *PAMONO* classification quality versus training time is analyzed in Section 5.5.4.

Related work concerning TI wavelets as a foundation for feature extraction is rather scarce [MT01]. One reason for this might be that they were not originally developed for feature extraction, but for improving wavelet denoising [CD95]. The intention of introducing the TI property to wavelet denoising was to obtain the same result in the following two cases: The first case is applying wavelet denoising to a given signal. The second case is applying a circular shift to that same signal, followed by wavelet denoising, followed by unshifting the result. For orthogonal (non-redundant) wavelets, the two denoising results differ due to energy transfer between coefficients on different scales as a side-effect of shifting. For TI denoising, the results do not differ because energy transfer only occurs between coefficients on the same scale (which is what encodes the shift). Generally speaking, invariance to translation means invariance to the location of an entity, which is an advantage in computing features for classifying that entity, if entity location is not correlated with the class label, as is the case for the ramp-like functions in *PAMONO* time series. This advantage has been exploited e.g. in face recognition, where a non-redundant Discrete Wavelet Transform (DWT) was used as the basis for approximately TI features [MT01]. These features were computed as estimates of the power spectrum in a local window. Even though the employed DWT is non-redundant and thus not translation-invariant, the estimated power spectra approximately are. In contrast, [LLS08] use raw wavelet coefficient values as features and ensure the TI property by redundant DWT and signal registration. The approach proposed in the following combines the use of TI wavelet coefficients with feature extraction. This serves the purposes of dimensionality reduction and increases robustness to noise.

Section Overview

The remainder of this section is organized bottom-up as follows: Section 5.5.1 presents the extraction of TI features from wavelet coefficients. Section 5.5.2 describes a feature ranking technique and, building on that, a feature selection strategy. Section 5.5.3 covers the training procedure for accelerated k-NN by means of a fast coreset-based clustering that condenses the training set. Section 5.5.4 demonstrates performance of these methods in classifying *PAMONO* time series. Section 5.5.5 compares these performances to fuzzy time series classification, and finally, Section 5.5.6 draws conclusions from this comparison.



(a) Up Signal **(b)** Down Signal **(c)** Background

Figure 5.7: Sorted Wavelet Coefficient Matrices. Examples of $\mathbf{W}$ for time series from the three classes. Scales (i.e. rows of $\mathbf{W}$) are ordered coarsest (back) to finest (front). Positions on the horizontal axis are column indices in $\mathbf{W}$. Classes differ in coefficient magnitude and slope, especially for coarser scales, which is exploited by the features in Equations (5.18) to (5.25). Figures adapted from [SFL+14].

5.5.1 Translation-Invariant Feature Extraction

Translation-invariant (TI) features are desirable in *PAMONO* time series classification because they make the feature space representation of a time series independent of the point in time when a nano-object attaches to the sensor surface. That means feature vectors are independent of the location of the upward or downward ramp-like part on the temporal axis, cf. Figure 5.6. After background elimination, *PAMONO* time series approximately fulfill the circularity condition assumed for the TI wavelets proposed by Coifman and Donoho [CD95] which are utilized here: The ramp-like, characteristic part of the signal has finite support[7] at an irrelevant location, preceded and followed by uniform noise. Hence time series with different locations of the ramp can be approximately regarded as circular shifts of each other if the uniform noise is regarded as 'basically the same' everywhere in the series. Neglecting boundary cases, non-circularity solely affects the noise portion of the series: Uniform sensor noise is shifted out on one side, and different uniform sensor noise is shifted in on the other side. In a strict sense, this difference in noise violates circularity. For practical purposes, however, it does not adversely affect feature computation. The statement that the circularity condition assumed for the TI wavelets holds *approximately* denotes this specific circumstance.

For extracting TI features from a time series, the following preprocessing is applied: To remove constant intensity offsets, the series mean is subtracted from each value. Let $\mathbf{v} \in \mathbb{R}^T$ denote the values of a single time series after this subtraction. Let $\widehat{\mathbf{W}} \in \mathbb{R}^{S \times T}$ denote the coefficient table of the TI wavelet transform from [CD95], using the maximum number $S = \log(T)$ of scales and the Haar wavelet basis. The Haar basis is used throughout the entire section because it proved superior in terms of classification Accuracy and computation time among 27 wavelet bases that were examined, cf. the comparison in Appendix B for details. The $S \times T$ matrix $\widehat{\mathbf{W}}$ is a non-orthogonal decomposition of the time series $\mathbf{v}$ into S scales

[7] Background elimination is required for the TI property to hold because background elimination turns a step function into a ramp-like function with finite support. This finite support makes the time series approximately fulfill the circularity condition required for the TI property [CD95]. Background elimination parameters applied to the set of training time series must be the same as for the input time series to be classified.

with T coefficients per scale. The scales with smaller index represent coarser structures of $\mathbf{v}$, while those with larger index represent finer details. The notion of translation-invariance that applies to $\widehat{\mathbf{W}}$ is that a circular shift of the series $\mathbf{v}$ manifests solely as per-scale permutations in $\widehat{\mathbf{W}}$, i.e. for each of the S rows of $\widehat{\mathbf{W}}$, the T entries are permuted. Note that there are neither permutations between coefficients on *different* scales, nor is there energy-transfer between *any* coefficients. The per-scale permutations are the way in which circular shifts of the input time series are encoded by the coefficient table. Since the goal is computing features that are invariant to shifts of the input series, this locational information must be eliminated. This is done by taking the absolute values of all coefficients in $\widehat{\mathbf{W}}$, followed by sorting each scale separately by descending absolute coefficient value. The resulting matrix of sorted absolute values is called $\mathbf{W}$. Figure 5.7 shows examples of $\mathbf{W}$ for series $\mathbf{v}$ from different classes. For any permutation of coefficients in $\widehat{\mathbf{W}}$, the same $\mathbf{W}$ will result, and hence the locational information has been removed. Feature extraction is carried out with respect to $\mathbf{W}$ as to be detailed now.

Features of Translation-Invariant Wavelet Coefficients

Let $\mathbf{W} = [w_{s,t}]_{s=1\ldots S, t=1\ldots T}$ be the coefficient table defined above, where s is the scale index and t is the coefficient index. Furthermore, let

$$\mu(w_{s,\circ}) = \frac{1}{T}\sum_{t=1}^{T} w_{s,t} \tag{5.16}$$

yield the mean value over the variable indicated by the wildcard symbol $\circ$, for a fixed value of the symbol s. The according standard deviation is defined analogously as

$$\sigma(w_{s,\circ}) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(w_{s,t} - \mu(w_{s,\circ}))^2}. \tag{5.17}$$

In the following equations, superscript indices are feature names, while the subscript s indicates the scale on which a feature is computed. Given these conventions, the TI features to be computed from the sorted wavelet coefficient matrix $\mathbf{W}$ are defined as follows: Feature f_s^1 is the mean value of the coefficients on scale s:

$$f_s^1 = \mu(w_{s,\circ}) \ . \tag{5.18}$$

Feature f_s^2 normalizes f_s^1 by the mean coefficient value over *all* scales:

$$f_s^2 = \frac{f_s^1}{\mu(w_{\circ,\circ})} \ . \tag{5.19}$$

Feature f_s^3 is the standard deviation of the coefficients on scale s,

$$f_s^3 = \sigma(w_{s,\circ}) \ , \tag{5.20}$$

while feature f_s^4 normalizes f_s^3 by the accumulated coefficient standard deviation over all scales:

$$f_s^4 = \frac{f_s^3}{\sum_r \sigma(w_{r,\circ})} \ . \tag{5.21}$$

For computing the remaining four features, a linear approximation of the sorted wavelet coefficients $w_{s,\circ}$ on each scale s is computed. This is motivated by the fact that coefficients on certain scales show a linear behavior for the background class, while being less linear and exhibiting varying slope for the up and down signal class, cf. Figure 5.7. Therefore, the remaining four features rely on approximation criteria and slope. Let $\mathbf{r}_s = [r_{s,t}]_{t=1...T}$ denote T discrete samples of a regression line approximating the sorted wavelet coefficients $w_{s,t}$ for a fixed scale s. The line is defined as $a_s x + b_s$, where a_s and b_s are computed as the minimizers of accumulated squared approximation error: $\text{argmin}_{a_s,b_s} = \sum_t (w_{s,t} - (a_s x_t + b_s))^2$. The sampling points x_t are chosen at the locations of the coefficients $w_{s,t}$. Hence $r_{s,t} = a_s x_t + b_s$ is the linear approximation of coefficient $w_{s,t}$. Given these prerequisites, f_s^5 is the slope of the regression line,

$$f_s^5 = a_s \ , \tag{5.22}$$

and f_s^6 is the linear approximation of the largest coefficient $w_{s,1}$:

$$f_s^6 = a_s x_1 + b_s \ . \tag{5.23}$$

Feature f_s^7 is the accumulated absolute deviation between coefficients $w_{s,t}$ and their linear approximations $r_{s,t}$, normalized by the accumulated coefficient set:

$$f_s^7 = \frac{\sum_t |w_{s,t} - r_{s,t}|}{\sum_t w_{s,t}} \ . \tag{5.24}$$

Feature f_s^8 is defined analogously to f_s^7 but with sums replaced by standard deviations:

$$f_s^8 = \frac{\sigma\left(w_{s,\circ} - r_{s,\circ}\right)}{\sigma\left(w_{s,\circ}\right)} \ . \tag{5.25}$$

Note that the differences in the numerator of f_s^8 are taken only between coefficients $w_{s,t}$ and approximations $r_{s,t}$ with the same index t. To prevent numerical issues, a small constant ϵ on the order of computational working precision is added to each denominator.

In order to balance the impact of the different features during the distance computations occurring in subsequent processing stages, the features need to be normalized accordingly. The employed normalization method is shifting and scaling the range of each feature to the unit interval $[0, 1]$. Shifts and scale factors are determined from the training dataset and also applied to any other dataset, the resulting classifier is applied to.

5.5.2 Feature Ranking and Selection

The employed feature ranking and selection method is based on computing a figure of merit for each feature in a supervised fashion. Let $g = f_s^j, s \in \{1, \ldots, S\}, j \in \{1, \ldots, 8\}$ be an abbreviation for the single scalar feature under consideration. Let $g_k^{c_i}, k \in \{1, \ldots, N^{c_i}\}, i \in \{1, \ldots, C\}$ denote the values that feature g attains over the N^{c_i} examples of class c_i in the training dataset, where C is the total number of classes.

The basic measure in computing the figure of merit of a feature g is the mean absolute distance d^{c_a,c_b} between the feature values of class c_a and the mean feature value of class c_b:

$$d^{c_a,c_b} = \mu\left(|g_\circ^{c_a} - \mu\left(g_\circ^{c_b}\right)|\right) \ . \tag{5.26}$$

Using Equation (5.26), the figure of merit m_g of feature g is computed as the following ratio:

$$m_g = \frac{\sum_{c_b} \sum_{c_a \neq c_b} d^{c_a, c_b}}{\sum_{c_i} d^{c_i, c_i}} \,. \tag{5.27}$$

The numerator of m_g accumulates over all classes c_b, in how far the mean feature value for class c_b differs from the feature values for all other classes $c_a \neq c_b$. For features that separate the classes well, this value is large. The denominator accumulates over all classes, in how far feature values vary within a class, hence smaller is better. The features g are then sorted in the order of descending merit m_g, giving a feature ranking assigning rank one to the best feature, rank two to the second-best and so forth. For determining the final sequence of features, the ranking idea by Kuncheva [Kun07] is applied within a ten-fold cross-validation [Koh95]: The ranks attained by each feature are accumulated over the different folds, and the features are sorted in the order of ascending accumulated ranks. Kuncheva indices [Kun07] are computed over this cross-validation in order to evaluate feature selection stability. Finally, the first F of these features are selected to be used for classification, where F is chosen to maximize the mean classification performance in a second cross-validation using the rank-sorted features and incrementally increasing the candidate for F. In all experiments conducted within this section, Accuracy (cf. Appendix A) over class-balanced datasets is the employed measure of classification performance.

5.5.3 Condensed k-NN Using Fast Coreset Clustering

This section describes how the k-Nearest Neighbors (k-NN) classifier [HTF09] is accelerated by computing it from cluster centers of the training data, thus defining a 'condensed k-NN' classifier [ZLX09]. Clustering serves as a preprocessing step in this procedure and will be covered in the first part of this section. It is accelerated for large input sets by using the coreset-based BICO approach [FGS+13]. The second part of this section then depicts the training procedure and the application of the obtained condensed k-NN classifier, making use of the clustering results.

Fast Clustering with Coresets

Clustering is usually defined as partitioning a set of objects into groups, such that objects in the same group are similar and objects in different groups are dissimilar. The k-Means problem is a well-studied clustering problem defining similarity via Euclidean distance. For two vectors $\mathbf{p} = (p_1, \ldots, p_F), \mathbf{q} = (q_1, \ldots, q_F) \in \mathbb{R}^F$, let $\|\mathbf{p} - \mathbf{q}\| \coloneqq \sqrt{\sum_{i=1}^{F} (p_i - q_i)^2}$ denote their Euclidean distance. Given an $N \times F$ matrix $\mathbf{P} = \left[\mathbf{p}^1; \ldots; \mathbf{p}^N\right]$ where each $\mathbf{p}^i \in \mathbb{R}^F$ is a row vector, the k-Means problem asks for a $K_m \times F$ matrix $\mathbf{Q} = \left[\mathbf{q}^1; \ldots; \mathbf{q}^{K_m}\right]$ of K_m cluster centers that minimize the sum of squared distances of all vectors in $\mathbf{P}$ to their nearest cluster center in $\mathbf{Q}$. Typically the number K_m of desired cluster centers in $\mathbf{Q}$ is chosen considerably smaller than the number N of input vectors in $\mathbf{P}$. Formally, the k-Means problem can be stated as the following minimization with result matrix $\mathbf{Q}$:

$$\underset{\mathbf{Q} \in \mathbb{R}^{K_m \times F}}{\operatorname{argmin}} \operatorname{cost}(\mathbf{P}, \mathbf{Q}), \tag{5.28}$$

with

$$\operatorname{cost}(\mathbf{P}, \mathbf{Q}) \coloneqq \sum_{\mathbf{p}^i \in \mathbf{P}} \min_{\mathbf{q}^j \in \mathbf{Q}} \left\|\mathbf{p}^i - \mathbf{q}^j\right\|^2 . \tag{5.29}$$

Note that the min-function in Equation (5.29) finds the distance of the nearest cluster center in a given $\mathbf{Q}$, while the argmin in Equation (5.28) searches the space of all possible $K_m \times F$ cluster center matrices $\mathbf{Q}$.

The cost function in Equation (5.29) is a special case of the weighted k-Means cost function, in which each vector can be weighted by a function $w\colon \mathbb{R}^F \to \mathbb{R}_{\geq 0}$, i.e.

$$\mathrm{cost}_w(\mathbf{P}, \mathbf{Q}) \coloneqq \sum_{\mathbf{p}^i \in \mathbf{P}} w(\mathbf{p}^i) \min_{\mathbf{q}^j \in \mathbf{Q}} \left\| \mathbf{p}^i - \mathbf{q}^j \right\|^2 \tag{5.30}$$

is minimized. Using a k-Means cost function as the objective for condensing the training set is a natural choice when k-NN serves as the classifier because both algorithms decide a vector's membership to a cluster, respectively class, via Euclidean distance.

In practice, Lloyd's algorithm [Llo82] is frequently used to minimize Equation (5.29), respectively its weighted variant Equation (5.30). It is an iterative algorithm converging to a local optimum after a potentially exponential number of steps. The k-means++ algorithm by Arthur and Vassilvitskii [AV07] is an improvement of Lloyd's algorithm, yielding an $\mathcal{O}(\log K_m)$ approximation guarantee. Its runtime is similar to that of Lloyd's algorithm. Both algorithms do not scale well and are hence time-consuming for large sets of input data.

One way of addressing this problem is to construct a small summary of the input set of vectors first and to cluster that summary instead. Such a summary can be computed in terms of a **coreset**: A (K_m, ϵ)-coreset for K_m desired cluster centers and approximation ratio ϵ is a small weighted set of vectors $\mathbf{S} \in \mathbb{R}^{\widehat{N} \times F}$ that ensures that the weighted clustering cost of $\mathbf{S}$ for any set of K_m cluster centers $\mathbf{Q} \in \mathbb{R}^{K_m \times F}$ is a $(1+\epsilon)$-approximation of the cost of the original input $\mathbf{P} \in \mathbb{R}^{N \times F}$ [HM04]:

$$|\,\mathrm{cost}_w(\mathbf{S}, \mathbf{Q}) - \mathrm{cost}(\mathbf{P}, \mathbf{Q})| \leq \epsilon\, \mathrm{cost}(\mathbf{P}, \mathbf{Q})\ . \tag{5.31}$$

Here $w\colon \mathbb{R}^F \to \mathbb{R}_{\geq 0}$ is the weight function, and the number $\widehat{N}$ of vectors in the coreset $\mathbf{S}$ may be considerably smaller than the number N of vectors in the original dataset $\mathbf{P}$. Since large sets of data may not fit into main memory, and short construction times are crucial, coresets are often computed in a streaming setting. The BICO (BIRCH Meets Coresets) algorithm detailed in [FGS+13] is a streaming-capable algorithm for computing coresets. It combines the data structure underlying the BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies) algorithm [ZRL97] with the idea of coresets as defined above. It is used in the following, to reduce the time taken for clustering large input datasets. The subsequent text describes how BICO is integrated into the training procedure of k-NN, yielding a condensed k-NN.

Training and Application of Condensed k-NN

The input of the training procedure for condensed k-NN is an $N \times F$ matrix $\mathbf{G}$, where rows denote training examples and columns denote normalized features, cf. Section 5.5.1. F is the number of features selected as according to Section 5.5.2. N is the sum $N = \sum_{i=1}^{C} N^{c_i}$, giving the total number of examples over the C different classes $\{c_1, \ldots, c_C\}$. In training, class labels are known and can hence be used to partition $\mathbf{G}$ into C per-class matrices $\mathbf{G}^{c_i}$. For each class c_i separately, the BICO approach is used to compute a coreset $\mathbf{S}^{c_i}$ of $\mathbf{G}^{c_i}$, allowing for very large input sets. Subsequently, weighted k-means++ is used to compute a clustering from each coreset $\mathbf{S}^{c_i}$, condensing the examples in $\mathbf{G}^{c_i}$ to $K_m \ll N^{c_i}$ cluster

centers $\mathbf{H}^{c_i}$. The union $\mathbf{H} = [\mathbf{H}^{c_1}; \ldots; \mathbf{H}^{c_C}]$ of the per-class cluster centers is then used as the condensed training set for k-NN. The number K_m of cluster centers per class is chosen as to be tractable in a lazy learning approach like k-NN and can furthermore be exploited to achieve class balancing, i.e. equal prevalence of condensed examples over all classes in $\mathbf{H}$.

Applying the learned classifier to unlabeled input works as follows: The raw input data is preprocessed like the training set, using the same feature normalization and selection. The resulting feature vectors are classified using k-NN with Euclidean distance metric on the condensed training set $\mathbf{H}$. The number K_n of nearest neighbors in k-NN is not to be confused with the number K_m of cluster centers in k-Means. The output of this k-NN classifier consists of predicted labels for the unlabeled input.

5.5.4 Performance

Three variants of the *PAMONO* time series classification task were examined to validate the proposed methods: Task 1 is the three-class separation of time series from the up signal, down signal and background classes (cf. Figures 5.6 and 5.7). Task 2a is the separation of the union of the up and down signal classes, denoted as up $\cup$ down, from the background class, i.e. task 2a is the same classification task as solved by fuzzy template matching in Section 5.4. Task 2b is the separation of the up signal from the background class only. It can be regarded as a variant of task 2a: If time series from the down signal class were present in the input, as is the case for real data, their classification as up signals or background is not penalized in task 2b, hence this corresponds to a scenario where classes assigned to down signals do not matter.

The tasks were enumerated in the order of decreasing difficulty and increasing practical importance: For nano-object detection, it is most important to separate the up signals related to nano-object adhesions from the background related to empty regions (task 2b). Correct classification of the down signal class is less important because if the up signals arising in the center of a nano-object adhesion and in the concentric rings around it (cf. Figure 5.6) are classified correctly, the burrs and holes possibly arising at the down signals in between those rings can be fixed by morphological closing, cf. [GW07] and Section 5.6.1. Uniting the up and down signal classes penalizes the classification of either as background, but like fuzzy template matching it makes no distinction between up and down signals (task 2a). The most difficult task is to correctly separate all three classes (task 1), which is of minor practical interest as the goal is only covering the extent of a nano-object adhesion, not determining signal directions. It was considered to examine the capabilities of the proposed method.

Experimental validation was conducted using a total of $N = 300000$ labeled *PAMONO* time series as input. Each of the $C = 3$ classes was represented with 100000 examples, making the dataset balanced for task 1. For task 2a, every other example from the up and down class was omitted, such that there were 100000 examples from the union class of up and down signals, and 100000 examples from the background class. For task 2b, all examples from the down signal class were omitted. Hence all tasks were examined for class-balanced data, i.e. the number of examples in each class to be distinguished is the same. The labeled examples were obtained using the signal model from Chapter 4 [SLW+14], i.e. real sensor images and nano-object templates were used to create synthetic time series. Three real datasets were used as the basis for this synthesis: 200 nm nano-objects on two differently severe levels of noise ("200 nm HQ" and "200 nm LQ" in Table 7.1) and one dataset with 100 nm nano-objects

Table 5.1: Accuracy for Varying $\widehat{N}$ and K_m. The table shows the impact on Accuracy of varying the coreset size $\widehat{N}$ and the number K_m of cluster centers computed per class.

Configuration	Task 1	Task 2a	Task 2b
$\widehat{N} = 7500, K_m = 1500$	0.86980	0.95407	0.99889
$\widehat{N} = 7500, K_m = 150$	0.86532	0.95156	0.99872
$\widehat{N} = 7500, K_m = 3000$	0.87096	0.95339	0.99898
$\widehat{N} = 75000, K_m = 1500$	0.87234	0.95477	0.99896

("100 nm HQ" in Table 7.1). All datasets are represented in equal proportions within the considered input. The temporal length of the considered time series is $T = 512$, resulting in 72 features available for selection (8 features on $S = \log(T) = 9$ scales).

The input examples (300000 or 200000, depending on the task) were partitioned into a training and test set, balancing them with respect to source datasets and class labels, i.e. each dataset and each class label is represented in equal proportions in both:

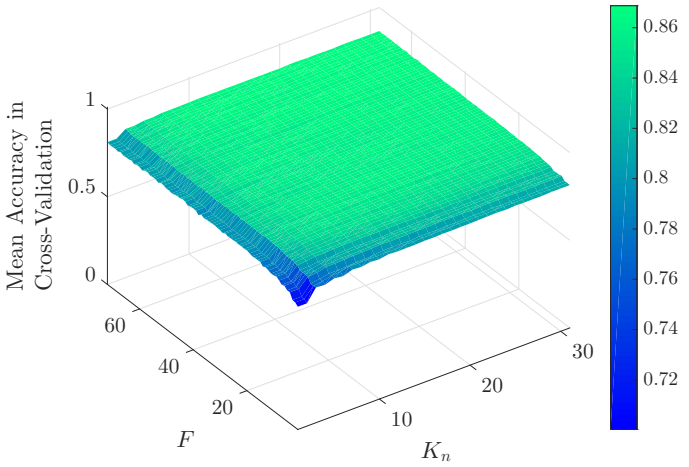
- The **training set** contains 2/3 of these examples and is used to train the condensed k-NN classifier. Furthermore, it is used for feature and parameter selection: A ten-fold cross-validation [Koh95], stratified with respect to class labels, is conducted, within which Kuncheva ranks [Kun07] as according to Section 5.5.2 are computed, along with mean Accuracy over all folds.
- The **test set** contains the remaining 1/3 of examples and is used solely for performance assessment.

Accuracy was selected as the performance metric, due to its capability to assess quality in two- and three-class classification tasks, and furthermore for its property of considering all entries in the confusion matrix of the classification results. Accuracy is defined in Appendix A. It measures the ratio of correctly classified examples among all classified examples. The class-balanced nature of the input datasets equalizes the misclassification penalty over the different classes.

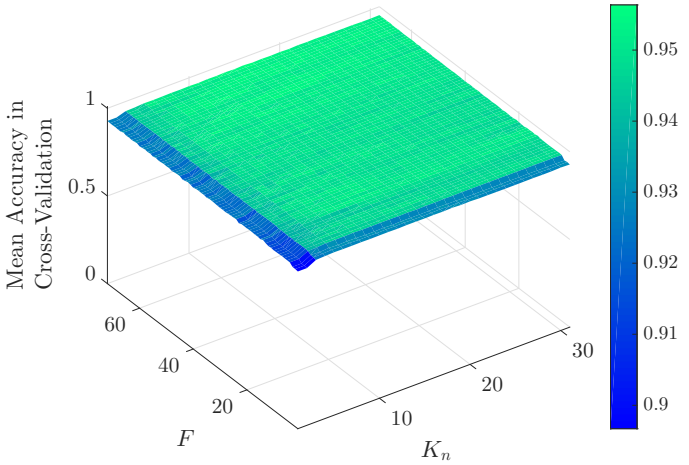
Accuracy and Parameters

The number of cluster centers per class was fixed at $K_m = 1500$ (1.5% of the number of examples per class in task 1) and the coreset size was fixed at $\widehat{N} = 7500$, i.e. at $5K_m$. For determining the number F of best features to be selected and the number K_n of neighbors in k-NN, a grid search was conducted over a ten-fold cross-validation [Koh95] on the training set. F and K_n were chosen to maximize mean Accuracy over the folds. Figure 5.8 plots the achieved mean Accuracy over parameter values for each task. The bars on the right show that for each task, Accuracy is in a different range, with task 2b exhibiting values close to 1. The spread of Accuracy over parameter space decreases with decreasing task difficulty. Accuracy saturates with increasing parameter values, meaning that F and K_n just need to be 'large enough'. In practice, this means that F and K_n need not be optimized to obtain a well-performing *PAMONO* time series classifier.

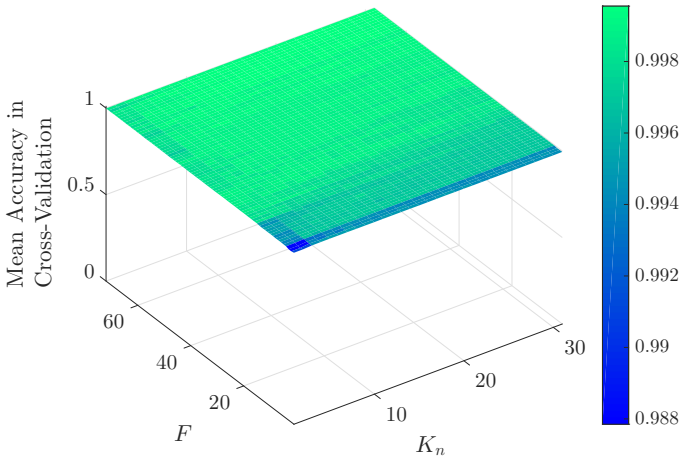
After learning such a classifier from the 2/3 training set, the following Accuracy values were attained on the unseen 1/3 test set: task 1: 0.86980, task 2a: 0.95407, task 2b: 0.99889 (first line of Table 5.1). For a more detailed view on classification performance, confusion



(a) Classification Task 1: Up vs. Down vs. Background



(b) Classification Task 2a: (Up ∪ Down) vs. Background



(c) Classification Task 2b: Up vs. Background

Figure 5.8: Impact of Parameter Choice. Mean Accuracy in ten-fold cross-validation over the number F of best features and the number K_n of neighbors to be used in k-NN is plotted for the three classification tasks. Only the choice of very small parameter values has an (adverse) effect on mean Accuracy. Optimization of these parameters is not necessary as long as they are chosen 'large enough'. Figures adapted from [SFL+14].

Table 5.2: Confusion Matrices for Test Set. Confusion matrices for the three classification tasks are shown, as attained on the test set that was balanced with respect to the source datasets and class labels. Parameters were $K_m = 1500$ cluster centers per class, computed from coresets of size $\widehat{N} = 7500$. The numbers of features F and neighbors K_n were chosen by grid search, cf. Figure 5.8, and they differ by classification task. Matrix entries consist of absolute example counts, as well as rounded relative ratios in brackets. The number of examples for task 1 is 100000 because a 1/3 test set of 300000 examples in total is considered. Tasks 2a and 2b consider fewer examples due to class balancing.

(a) Classification Task 1: Up vs. Down vs. Background

		Ground Truth			
		Up	Down	Background	$\sum$
Prediction	Up	30848 (0.31)	5852 (0.06)	0 (0.00)	36700 (0.37)
	Down	2484 (0.02)	24530 (0.25)	1731 (0.02)	28745 (0.29)
	Background	1 (0.00)	2952 (0.03)	31602 (0.32)	34555 (0.35)
	$\sum$	33333 (0.33)	33334 (0.33)	33333 (0.33)	100000 (1.00)

(b) Classification Task 2a: (Up $\cup$ Down) vs. Background

		Ground Truth		
		Up $\cup$ Down	Background	$\sum$
Prediction	Up $\cup$ Down	31276 (0.47)	1005 (0.02)	32281 (0.48)
	Background	2057 (0.03)	32328 (0.48)	34385 (0.52)
	$\sum$	33333 (0.50)	33333 (0.50)	66666 (1.00)

(c) Classification Task 2b: Up vs. Background

		Ground Truth		
		Up	Background	$\sum$
Prediction	Up	33281 (0.50)	22 (0.00)	33303 (0.50)
	Background	52 (0.00)	33311 (0.50)	33363 (0.50)
	$\sum$	33333 (0.50)	33333 (0.50)	66666 (1.00)

matrices for all three tasks are provided in Table 5.2. The confusion matrix for task 1 shows that the largest part of class confusion arises from the down signal class: Here, false classifications of down signals are responsible for approximately nine percent loss in Accuracy, while the two other classes taken together cause only four percent. The largest portion of this error is due to down class examples that are assigned to the up class. Confusion between the up and background class is particularly rare: One up example was erroneously classified as background, and no background examples were assigned to the up class. Regarding task 2a, more up $\cup$ down examples were confused with the background class than the other way around, a property which holds analogously for task 2b with the up and background class. In task 2b, however, the number of confused examples is considerably smaller: 74 out of 66666 examples are classified incorrectly, again confirming that the primary source of misclassification is the down signal class, which is not considered in task 2b.

Runtime

Runtimes, as observed with an Intel® Core™ i7-2600 at 3.4 GHz (cf. also System Specification 7.1 in Section 7.3.6), were as follows: Clustering the 2/3 training set takes approximately 77 s, averaging over the tasks. This involves applying BICO and k-means++ for each class using coreset size $\widehat{N} = 7500$. Applying the thus learned k-NN classifier based on the trained $K_m = 1500$ cluster centers per class to the 1/3 test set takes approximately 17 s, averaging over the tasks. The bottleneck is feature computation, which is not included in the numbers given above. Feature computation consists of computing the TI wavelet transforms of all time series and extracting the features in Equations (5.18)–(5.25) from them over all scales of the transforms. It must be carried out before both, classifier training via clustering and its application via k-NN. In a nonparallel MATLAB implementation, TI wavelet computation and feature extraction take 3258 s in total for 100000 time series of length $T = 512$. This time is dominated by the computation of the TI wavelet tables $\widehat{\mathbf{W}}$, requiring approximately 3/4 (2480 s) of the overall computation time. To attain real-time capability for a *PAMONO* dataset with 100000 spatial pixel coordinates (i.e. 100000 time series to be processed every T images), classification must be finished within the time taken to record the next T sensor images. At a recording rate of 20 images per second and temporal block size $T = 512$, at most 25.6 s are available for feature computation and k-NN application, thus ruling out the approach for real-time application in its current implementation.

The k-NN classifier can be accelerated by reducing the number K_m of cluster centers per class in its training data. Reducing K_m from 1500 to 150 decreases k-NN application time from 17 s to 2 s, with minor decreases in Accuracy, compare lines one and two in Table 5.1. Alternatively, a non-lazy learner like those described in Section 6.6 can be used. With k-NN time reduced to 2 s, 23.6 s are left for feature computation. As it takes 3258 s in the given nonparallel MATLAB implementation, a speedup of at least 140 is required to achieve real-time capability. This hints at a GPU implementation of TI wavelet transformation and feature extraction. Both algorithms process time series independently, and time series length (e.g. $T = 512$) does not preclude them from being processed in the local memory of the GPU's multiprocessors. Hence, a massively parallel implementation on a GPU is conceivable, albeit not being subject of this thesis.

Doubling the number K_m of computed cluster centers from 1500 to 3000 yields a minor increase in Accuracy, compare lines one and three[8] in Table 5.1. Runtime of k-NN approximately doubles from 17 s to 29 s, averaging over the tasks.

Due to the fact that for classifier application the runtime of clustering during training is not important, another experiment was conducted that invested more runtime on this end: The coreset size $\widehat{N}$ was increased from 7500 to 75000. Clustering time increased considerably from 77 s to 2427 s, while the increase in Accuracy was slightly larger than for increasing K_m to 3000, except for task 2b, where Accuracy remains nearly unchanged, indicating saturation, cf. last line of Table 5.1.

Feature Ranking and Selection

Feature ranking was carried out as described in Section 5.5.2. Figure 5.9 shows the Kuncheva ranks (lower means 'more relevant') as attained by the different features on all scales,

[8] For task 2a, Accuracy with $K_m = 3000$ decreases but is higher than for $K_m = 150$.

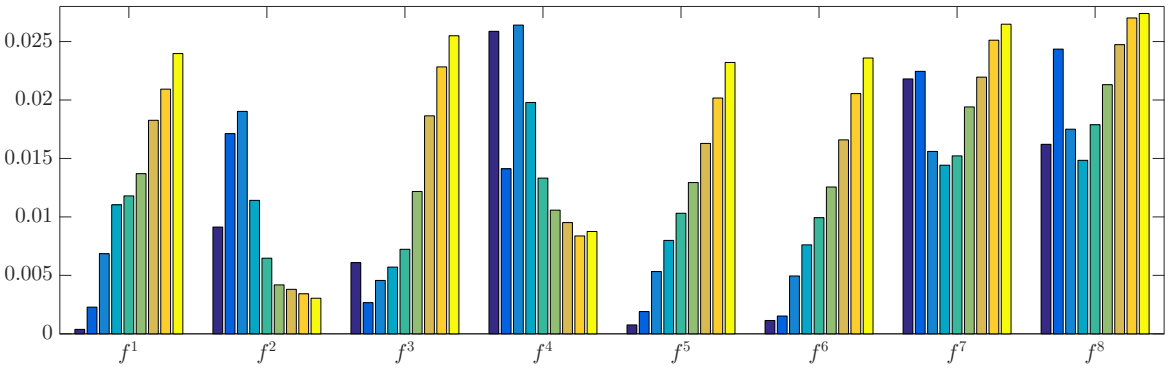


Figure 5.9: Kuncheva Ranks of Features. Kuncheva ranks (lower is more relevant) as attained by all features on all scales are displayed. Scales are ordered coarse (left) to fine (right) for each feature f^i. Figure adapted from [SFL+14].

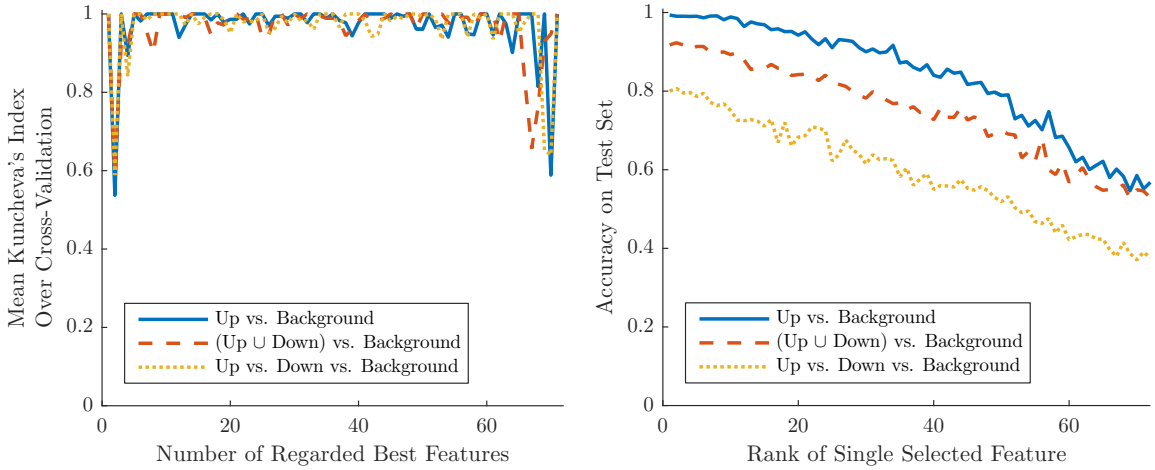
normalized by the sum of all ranks over all folds (vertical axis). The three most important features are located on the coarsest scale and exhibit strictly decreasing relevance for finer scales. Two of them (f_s^5 and f_s^6) are based on the regression line. The normalized versions f_s^2 and f_s^4 of f_s^1 and f_s^3 favor finer scales. The approximation-error-based features f_s^7 and f_s^8 are comparatively irrelevant on all scales. Note that Figure 5.9 shows results for classification task 1; for the other tasks, the results are qualitatively the same.

Figure 5.10a plots feature selection stability in terms of the mean Kuncheva indices attained over the ten-fold cross-validation. Kuncheva indices are above 0.93 for selecting between 10 and 60 features for all classification tasks (task 1: *short dashes*, task 2a: *long dashes*, task 2b: *solid line*). This indicates stability of the selected feature sets.

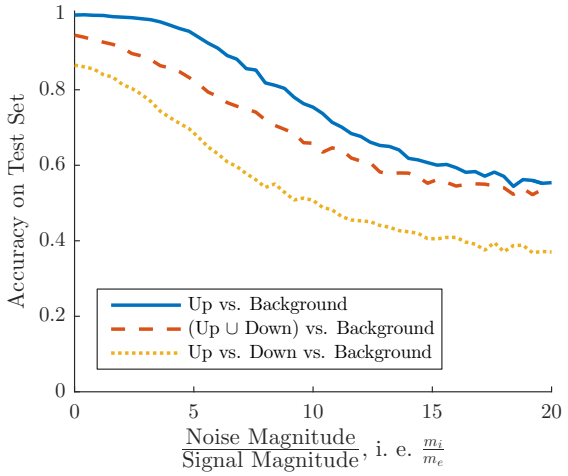
Figure 5.10b illustrates that the underlying feature ranking is meaningful in terms of Accuracy on the unseen test set: Each feature was used in isolation to classify the test set, using the proposed method. The attained Accuracy was plotted over the index that feature had in Kuncheva's ranking. Accuracy decreases approximately linearly with increasing Kuncheva rank. Note that seemingly high Accuracy can already be attained using only the single best feature. This particularly applies to task 2b. However, comparing Accuracy attained by that feature to the first line of Table 5.1 which shows results for using all selected features, it can be seen that single feature Accuracy falls short of the possibilities: Accuracy 0.80018 is attained for task 1 (all selected features: 0.86980), 0.91764 for task 2a (all selected features: 0.95407) and 0.99370 for task 2b (all selected features: 0.99889). Comparing these values to the ranges displayed in Figure 5.8, they rank around the midfield.

Robustness to Noise

In order to assess robustness of the features to noise in the input, experiments with artificial noise were conducted. The standard deviation of each input time series was computed, and the mean of these values was used as an estimate m_e of average signal magnitude. Then the interval $[0, 20m_e]$ was equidistantly sampled, and for each sample m_i, zero-mean, unit variance Gaussian noise was scaled by m_i/m_e and added to the already noisy original input dataset, yielding a noisier dataset. For each of the resulting noisier datasets, a classifier was trained using the proposed method, and test set Accuracy was measured for each classification task. Parameters were chosen as above: K_m = 1500 cluster centers per class



(a) Kuncheva Indices **(b)** Single-feature Accuracy Values



(c) Robustness to Noise

Figure 5.10: Kuncheva Indices, Single-Feature Accuracy Values and Robustness to Noise. The plots in this figure show results for all three considered time series classification tasks. The three-class task 'up vs. down vs. background' is indicated by short yellow dashes. The two-class task '(up ∪ down) vs. background' is indicated by long red dashes, while the 'up vs. background' task is indicated by a solid blue line. (a) shows Kuncheva indices attained for increasing the number of best features to select. Values closer to one indicate a higher stability of the selected set of features. (b) shows Accuracy values attained for using a single feature for classification. For increasing Kuncheva rank of the single feature, Accuracy decreases, thus validating the feature selection strategy. (c) illustrates robustness of the overall classification method to noise. The horizontal axis is the factor by which the magnitude of artificially added noise exceeds the magnitude of the already noisy input time series. Figures adapted from [SFL+14].

Table 5.3: Confusion Matrices for 100 nm Test Set. Confusion matrices for the three classification tasks are shown, as attained for training on two 200 nm datasets and using a 100 nm dataset as the test set, to which the tabulated results refer. Like with Table 5.2, parameters were $K_m = 1500$ cluster centers per class, computed from coresets of size $\widehat{N} = 7500$. The numbers of features F and neighbors K_n were chosen by grid search, cf. Figure 5.8, and differ by classification task. Matrix entries consist of absolute example counts, as well as rounded relative ratios in brackets. The number of examples for task 1 is 100000 because 100000 time series from the 100 nm dataset were considered. Tasks 2a and 2b consider fewer examples due to class balancing.

(a) Classification Task 1: Up vs. Down vs. Background

		Ground Truth			
		Up	Down	Background	$\sum$
Prediction	Up	31069 (0.31)	349 (0.00)	0 (0.00)	31418 (0.31)
	Down	2264 (0.02)	27528 (0.28)	2091 (0.02)	31883 (0.32)
	Background	0 (0.00)	5457 (0.05)	31242 (0.31)	36699 (0.37)
	$\sum$	33333 (0.33)	33334 (0.33)	33333 (0.33)	100000 (1.00)

(b) Classification Task 2a: (Up ∪ Down) vs. Background

		Ground Truth		
		Up ∪ Down	Background	$\sum$
Prediction	Up ∪ Down	29492 (0.44)	1307 (0.02)	30799 (0.46)
	Background	3842 (0.06)	32026 (0.48)	35868 (0.54)
	$\sum$	33334 (0.50)	33333 (0.50)	66667 (1.00)

(c) Classification Task 2b: Up vs. Background

		Ground Truth		
		Up	Background	$\sum$
Prediction	Up	33315 (0.50)	172 (0.00)	33487 (0.50)
	Background	18 (0.00)	33161 (0.50)	33179 (0.50)
	$\sum$	33333 (0.50)	33333 (0.50)	66666 (1.00)

were computed, each from a coreset of size $\widehat{N} = 7500 = 5K_m$, and the numbers F of features and K_n of neighbors were chosen to maximize mean Accuracy in cross-validation, as plotted in Figure 5.8. Figure 5.10c plots the obtained Accuracy values over m_i/m_e. The values for zero noise provide a baseline for each classification task. Results deteriorate slowly with increasing noise, which is especially true for task 2b (*solid line*), where noise with up to four times the magnitude of m_e causes only minor loss. From there on, the slope of Accuracy loss increases, indicating the tolerance limit of the method. It decreases again, starting at approximately twelve times more noise than signal, and for each task converges to the limit of random guessing for the respective task (1/2 for the two-class tasks and 1/3 for the three-class task; balanced class distributions).

Generalization Across Datasets

All previous evaluations assessed the generalization performance of the proposed method on a 1/3 test sample, balanced with respect to three different *PAMONO* measurements used as the data source for synthesizing the labeled time series. That means the training as well as the test set contain information from all three measurements/datasets in equal proportions. In order to assess generalization performance *across* datasets and thus across measurement conditions, a dataset with 100 nm nano-objects was used as the test set, while the classifier was trained solely on two 200 nm datasets. In this scenario, the following Accuracy values were attained (values in brackets repeat line one of Table 5.1 for comparison): task 1: 0.89839 (0.86980), task 2a: 0.92277 (0.95407), task 2b: 0.99715 (0.99889). Comparing the confusion matrices for the 100 nm test set in Table 5.3 to the previous one in Table 5.2 explains why in task 1, Accuracy is better for the 100 nm case: Fewer down signals are erroneously classified as up signals, pointing out that 100 nm down signal feature vectors are located further away from 200 nm up signal cluster centers than the mixed down signals are separated from the mixed up signal cluster centers in the dataset from Table 5.2. They are, however, located closer to background cluster centers, as can be seen from the increased confusion of down signals as background. Both effects can be explained by the lower amplitudes in 100 nm signals due to the size-dependence of the SPR effect (cf. Section 2.2): 100 nm down signals are less similar to 200 nm up signals and more similar to background because their amplitudes are smaller, pulling them away from the high amplitudes in 200 nm up signals and pushing them closer to the low amplitudes in background signals that depend solely on SNR. This also explains the increased confusion between the up $\cup$ down and the background class in task 2a: Down signals are more likely to be classified as background. On the other hand, background time series are not affected by nano-object size and look more similar across datasets, explaining the nearly unchanged confusion of background as up $\cup$ down signals. When comparing task 2b to task 1 in Table 5.3, it is noticeable that there is zero confusion in both directions between the up and background class in task 1, but a minor increase in confusion in task 2b. This indicates that the examples that were confused in task 2b end up in the down class in task 1.

5.5.5 Comparison to Fuzzy Template Matching

A comparison between the TI wavelet feature-based time series classification from Section 5.5 and the fuzzy template matching-based time series classification from Section 5.4 was conducted in order to make a choice between the two. To this end, the fuzzy template matching method was applied to the same dataset as was used in Section 5.5.4 to evaluate the TI wavelet feature-based method.

The parameters of fuzzy template matching, as listed in Section 5.4.4, were optimized by conducting a 5000 trials random search, and runtime as well as Accuracy were measured. Parameters of the modules that precede time series classification in Figure 5.1 were not optimized, but the same parameters as for the TI wavelet feature-based method were used. Hence the two methods are compared with respect to their ability to handle the data produced by a fixed background elimination and denoising strategy.

Accuracy was measured for task 2a, i.e. for the distinction of up $\cup$ down signals from background because, as discussed in Section 5.4, this is the two-class classification task solved by fuzzy template matching. Like with the wavelet method, classes are balanced, i.e. one half

Table 5.4: Performance of Fuzzy Template Matching. The 'Union'-row shows performance measures as attained on the composite dataset that was also used in evaluating the wavelet-based approach. The other rows subdivide these results into the three datasets from which the union dataset was composed. Dataset names refer to Table 7.1. The columns to the right of the 'Accuracy'-column display ratios of correctly classified examples in the respective class. The two rightmost columns refer to the correct classification of up, respectively down signals into the up ∪ down class. The primary source of misclassifications is the confusion of down signals as belonging to the background class, cf. rightmost column. This particularly affects datasets with low Signal-to-Noise Ratio (SNR), either due to a low quality sensor surface (200 nm LQ) or low amplitudes because of small nano-object size (100 nm HQ).

Dataset	Accuracy	Background	Up ∪ Down	Up	Down
Union	0.82980	0.93124	0.72835	0.95506	0.50164
200 nm HQ	0.92933	0.94818	0.91033	0.96745	0.85321
200 nm LQ	0.79691	0.95669	0.63591	0.93648	0.33533
100 nm HQ	0.76439	0.88916	0.64144	0.96106	0.32182

of the data belongs to the up ∪ down class and the other one to the background class, making Accuracy a suitable indicator of classification quality over all classes, cf. Appendix A. As a reminder, the dataset is composed of time series synthesized from three sensor measurements with different quality, respectively nano-object size (datasets "200 nm HQ", "200 nm LQ" and "100 nm HQ" in Table 7.1).

Accuracy as attained by fuzzy template matching for the union of the three datasets is 0.82980 (cf. first line of the 'Accuracy'-column in Table 5.4). The wavelet method achieved Accuracy values between 0.95156 and 0.95477 in this scenario (cf. 'Task 2a'-column in Table 5.1). Fuzzy template matching has lower Accuracy due to bad performance on the 200 nm LQ and 100 nm HQ datasets, i.e. on datasets with low SNR (cf. last two lines of the 'Accuracy'-column). In 200 nm LQ the SNR is low due to a low quality sensor surface, while in 100 nm HQ it is low due to the linear relationship between particle size and signal amplitude, cf. Section 2.2 and [ZKG+10]. The primary source of misclassifications is not the background class: A ratio of 0.93124 of all background time series is classified correctly, and only on the 100 nm HQ dataset it performs worse, with a ratio of 0.88916 (cf. 'Background'-column in Table 5.4). Instead, most errors occur in the up ∪ down class, especially on the low SNR datasets. As ground truth for the up ∪ down class is available that differentiates up from down signals, the last two columns in Table 5.4 investigate the primary source of errors further: A ratio of 0.95506 of up signals in the overall dataset is classified correctly as belonging to the up ∪ down class. The lowest ratio in an individual dataset is 0.93648, obtained for 200 nm LQ. That means the primary source of the errors incurred by fuzzy template matching on the given composite dataset lies in the down class: Over all datasets, only a ratio of 0.50164 of down signals is classified correctly into the up ∪ down class. In the given balanced two-class scenario, this is close to randomly guessing the class of down signals. However, this issue affects only the down signal subset of the up ∪ down class. Practical conclusions from this are drawn in Section 5.5.6. While with a ratio of 0.85321 of correctly classified down signals, performance on the 200 nm HQ dataset is above random guessing, the ratios are 0.33533 and 0.32182 for the 200 nm LQ and 100 nm HQ datasets, respectively. As a summary, most of the errors incurred in fuzzy template matching arise from down signals that are not assigned to the up ∪ down signal class, but are confused with the background class. This is consistent

with the observations made in the wavelet-based approach, where the down signals were most evasive to correct classification as well.

Computation time required by the GPU-implementation of fuzzy template matching [LST+13a; LST+13b] are considerably lower than for the wavelet-based approach running on the Central Processing Unit (CPU): Evaluating the 5000 randomly drawn parameter sets took 42 265 s (≈12 h), resulting in an average of 8.45 s for one analysis with parameters known (NVIDIA® GeForce® GTX 980, cf. System Specification 7.1 in Section 7.3.6). In particular, fuzzy template matching is real-time capable: Being able to process the dataset consisting of $T = 512$ images within 8.45 s means that 60 images can be processed per second, while the recording rate of the sensor is 20 images per second. Computation times arising in the wavelet-based method are summarized here for convenience and contrasted with these values: A parameter optimization like in fuzzy template matching is not necessary for the wavelet-based method, as long as parameter values are chosen large enough. However, a training phase is required, which for the given dataset takes 77 s (coreset size $\widehat{N} = 7500$, $K_m = 1500$ centers per class to be computed) plus the time taken for feature computation, which is 3258 s per 100000 time series of length $T = 512$ and thus the bottleneck of the overall method. Applying the classifier to 100000 time series takes 17 s ($K_m = 1500$ centers per class), respectively 2 s ($K_m = 150$ centers per class), which by itself is real-time capable. However, the 3258 s for feature computation that also arise during application to 100000 input time series spoil this property.

A summary of comparing time series classification via fuzzy template matching to the TI wavelet-based method is given in Table 5.5, listing the strengths and weaknesses of both methods. An interpretation of these results is given in the conclusions in the subsequent section. More results, covering not only time series classification but all modules constituting the detector from Figure 5.1, as well as their interactions, are provided in Chapter 7.

5.5.6 Conclusion

A novel set of translation-invariant wavelet-based features for time series classification was presented and used in a condensed k-NN classifier that was built from a fast coreset-based clustering of a big training dataset. The efficacy of the approach in *PAMONO* time series classification was demonstrated: For the most important classification task of distinguishing up signals in the central parts of nano-object adhesions from noisy background, Accuracy close to 1 was achieved.

It was demonstrated that the method is robust to increasing noise in the input signal and insensitive to its main parameters (number of regarded features and number of nearest neighbors in k-NN), as long they are chosen large enough. Optimization of these parameters is not necessary. Feature selection was shown to be stable, and clustering time was considerably reduced by using the coreset-based BICO approach. Furthermore, the capability of generalizing across datasets was empirically demonstrated by training on two 200 nm datasets and applying the resulting classifier to a 100 nm dataset. Only a minor decrease in Accuracy was observed, as compared to learning from all three datasets.

Choice of Method

As the major drawback of the TI wavelet feature-based approach, the high computation times arising in wavelet transformation and feature extraction were identified, ruling out a

Table 5.5: Strengths and Weaknesses of the Presented Methods for Time Series Classification.

	Fuzzy Template Matching	TI Wavelet Features
Requirements	Template time series to match against	Labeled Training Data
Optimization	Parameters have large impact on results quality and need to be optimized for each dataset	Few parameters, each with minor impact on results quality; no optimization necessary as long as parameter values are chosen large enough
Training Speed	No training required	Fast coreset-based training but preceded by feature computation which is the main bottleneck
Application Speed	Real-time-capable on GPU	Real-time-capable classifier application but preceded by feature computation that requires a speedup of at least 140 to attain real-time performance
Quality	Accuracy 0.82980 on examined dataset	Accuracy 0.95407 on the same dataset

real-time application of the otherwise real-time-capable classifier. In face of the portable sensor scenario with on-site real-time analyses, this is an important disadvantage that can be addressed by a GPU implementation of the method, which is not the subject of this thesis.

Fuzzy template matching-based time series classification achieves real-time-capable performance on the GPU and attains ratios of correct predictions between 0.88916 and 0.96745 for the up signal and background class over all datasets, cf. Table 5.4. Like for the wavelet-based approach, but to a larger extent, the main weakness of fuzzy template matching lies in misclassification of the down signal class. In real data, down signals arise in the periphery of nano-object adhesions and constitute the least prevalent class, followed by up signals arising in and around the center of nano-object adhesions, and dominated by the considerably larger number of background time series: The synthetic data used in the preceding evaluations was balanced not for the sake of realism but in order to give equal importance to all classes and avoid the Accuracy paradox [Bru07] in developing the classifier. However, for real sensor data, bad performance in classifying down signals firstly affects only few time series and secondly can be remedied by exploiting the spatial structure of nano-object adhesions: The center part of Figure 5.6 illustrates that down signals occur in between up signals and that the union of up and down signals covers the area affected by a nano-object adhesion. Down signals that are misclassified as up signals hence cause no problem as they contribute to that area being detected. Down signals that are misclassified as background cause burrs or holes in that area but these can easily be fixed using morphological operators, as discussed in Section 5.6.1.

As a conclusion, in the remainder of this thesis fuzzy template matching is used as the method for time series classification. It is selected firstly for its real-time capability and secondly because the misclassifications it incurs can easily be remedied and affect the class that is least important in practice.

Future Work

For TI wavelet feature-based time series classification, one crucial perspective is a GPU implementation of the underlying TI wavelet transform as well as the subsequent feature extraction. Such a GPU port can enable real-time-capable application of the method, and thus exploitation of its superior Accuracy within the real-time scenario. Until then it can be used for offline analysis and aid e.g. in cases of datasets with very low SNR, like 50 nm nano-objects. Corresponding *PAMONO* measurements are yet to be conducted. Additionally, the cross-dataset generalization performance of the classifier can be investigated further, using datasets with larger difference in nano-object size and/or sensor surface quality.

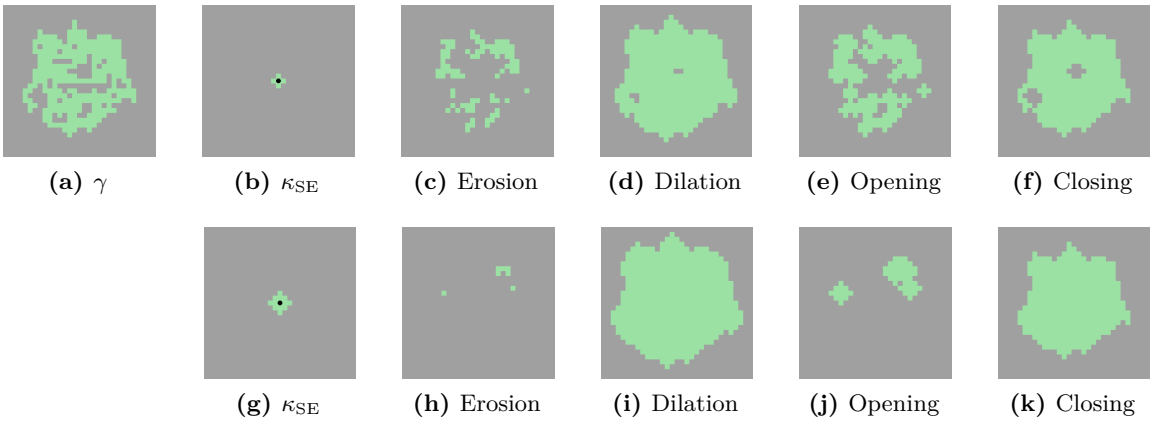
In a first phase, these experiments can be carried out using data obtained by synthesis, as described in Chapter 4. In a second phase, the synthetic test data used for assessing classification quality can be replaced with labeled time series obtained from real *PAMONO* sensor measurements. This involves manually labeling a large amount of time series, which can be conducted e.g. via crowdsourcing platforms like Amazon's *Mechanical Turk* [BKG11; DK13]. In order to reduce costs, training data can still be generated via synthesis, thus evaluating how a classifier trained from synthetic data performs on real data. An evaluation using real data furthermore means that performance measures are computed with respect to the real unbalanced class distributions. To capture class-specific performances in this case, quality measures like per-class Precision and Recall [SL09] can be regarded, instead of using Accuracy as a single omnibus index. In addition to that, classifiers other than k-NN can be examined.

A further experiment being a good follow-up to a GPU implementation is running the entire *SynOpSis* approach from Figure 3.2 using the wavelet-method for time series classification in the pattern detector from Figure 5.1. This evaluates the impact of the higher quality class mask it provides, on overall detection and classification quality. Such an experiment assesses the method's net benefit for *overall* analysis results. It evades the need for manually labeling data on the time series level, in favor of requiring a segmentation on the level of polygons delineating nano-objects, as output by the overall analysis.

5.6 Segmentation

Time series classification, as carried out by either one of the methods presented in the two preceding sections, yields a spatiotemporal class mask $\Gamma(x, y, t)$ as its output. Subsuming the up and down signal classes under the up $\cup$ down class makes this class mask binary, and spatiotemporal regions classified as up $\cup$ down approximately correspond to times *when* and areas *on* the sensor surface that are affected by nano-object adhesions. Figure 5.12a shows an amplified $T \cdot A$ estimate for such an area, with the nano-object manifesting as upward and downward deviations in measured intensities. Figure 5.12b overlays the corresponding class mask Γ, as computed by time series classification. It is composed of the pixels affected by up or down signals and approximately covers the area affected by the nano-object.

The segmentation module in Figure 5.1, which is to be detailed in this section, receives such a binary spatiotemporal class mask $\Gamma(x, y, t)$ as input, streams it through the GPU and aggregates spatiotemporally adjacent coordinates (x, y, t) with $\Gamma(x, y, t) = 1$ to contiguous candidate objects, represented as 2-D polygons. This process is real-time-capable and is divided into three steps:



(a) γ **(b)** κ_{SE} **(c)** Erosion **(d)** Dilation **(e)** Opening **(f)** Closing

(g) κ_{SE} **(h)** Erosion **(i)** Dilation **(j)** Opening **(k)** Closing

Figure 5.11: Morphology – Examples. Four morphological operators were applied to the binary input class mask γ shown in (a), using the structuring elements κ_{SE} in (b) and (g), respectively. The origins of the structuring elements are indicated by black dots. Each row of images shows the results of applying either erosion, dilation, opening or closing to (a), using the respective structuring element. Closing γ with the circular structuring element of radius 2 displayed in (g) remedies burrs and holes in the class mask, cf. (k).

1. A preprocessing on the pixel-level is applied to the class mask Γ to remove its possible burrs and holes, cf. Section 5.6.1.
2. The resulting enhanced pixel mask $\widehat{\Gamma}$ is converted to 2-D polygons by applying the marching squares algorithm [BL03] to each image $\widehat{\Gamma}(\circ, \circ, t)$, cf. Section 5.6.2.
3. A postprocessing on the polygon-level clusters polygons in close spatiotemporal proximity and represents each such cluster by a single polygon with largest area. This aims at removing repeated detections of the same nano-object as well as spurious/shattered responses, cf. Section 5.6.3.

5.6.1 Preprocessing on the Pixel-Level

Preprocessing on the pixel-level is carried via standard morphological operators [GW07] which are briefly summarized here, explaining their purpose for *PAMONO* and their parameters to be optimized. As these operators are applied separately for each temporal coordinate t, the function $\gamma(x, y)$ is used as an abbreviation to denote one 2-D spatial image from the class mask Γ at a certain point in time t, i.e. $\gamma(\circ, \circ) = \Gamma(\circ, \circ, t)$ for the current t under consideration.

Morphological operators on binary input images are defined based on a so-called structuring element $\kappa_{\mathrm{SE}}(x, y) \in \{0, 1\}$, determining the kernel of the morphological operation. Structuring elements, in turn, are defined by their coordinates x, y of support and their origin. Figures 5.11b and (g) show two structuring elements κ_{SE}. In (b), κ_{SE} is a circle with radius one and its origin in the center (indicated by a black dot), while in (g) the radius of the circle is two. Morphological operations are implemented by using the respective structuring element as a nonlinear window operator: The origin of κ_{SE} is translated to every coordinate x, y in the domain of γ (zero-padding γ to handle boundary conditions), and the value in γ below the origin of κ_{SE} is replaced with the operator result. Four of the most basic among such operators, namely erosion, dilation, opening and closing, are explained in the following.

In an **erosion**, the value in γ at the origin of κ_{SE} is set to one, if *all* values in γ that are covered by the one-values in κ_{SE} equal one. Erosion thus shrinks the class mask by removing details in γ that are smaller than κ_{SE}, e.g. burrs at the fringes of nano-objects. Figure 5.11a displays an example class mask γ around a nano-object adhesion, while (c) and (h) show the results of applying erosion with the structuring elements in (b) and (g) respectively.

In a **dilation**, the value in γ at the origin of κ_{SE} is set to one, if *any* of the values in γ that are covered by the one-values in κ_{SE} equals one. Dilation thus grows the class mask by bridging gaps and holes in γ that are smaller than the structuring element. It therefore eliminates burrs at the fringes of nano-objects by growing the class mask to incorporate the burrs. Figures 5.11d and (i) show the results of dilating (a) with the structuring elements in (b) and (g), respectively.

The preprocessing applied to the class mask in the *PAMONO* pattern detector aims at removing burrs and closing gaps without overly shrinking or growing the mask because that would result in either missing parts of the nano-object areas or including too much of the background area. This goal can be achieved by combining erosion and dilation, giving two further morphological operators, called opening and closing.

Opening γ with a structuring element κ_{SE} means that first, γ is eroded with κ_{SE} and then the result is dilated with κ_{SE}. Figures 5.11e and (j) show the result of opening (a) with the structuring elements in (b) and (g), respectively.

Closing γ with a structuring element κ_{SE} means that first, γ is dilated with κ_{SE} and then the result is eroded with κ_{SE}. Figures 5.11f and (k) show the result of closing (a) with the structuring elements in (b) and (g), respectively.

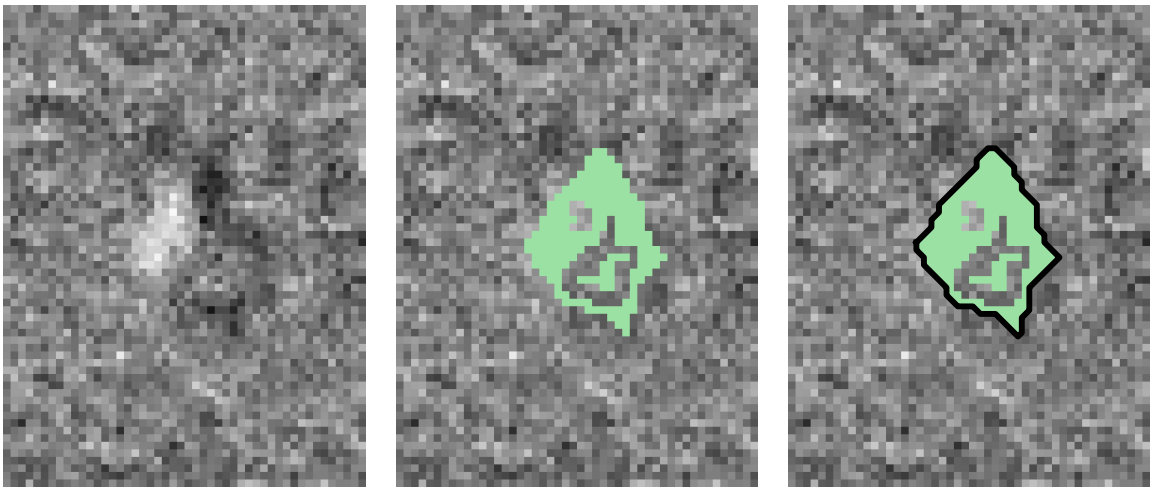
Application in Segmentation and Arising Parameters

In the segmentation module of the *PAMONO* pattern detector from Figure 5.1, morphological closing and opening can be applied to the class mask Γ provided by time series classification. Closing is enabled by the parameter $b^{\mathrm{closing}} \in \{0,1\}$, which is subject to optimization, along with the radius $K^{\mathrm{closing}}_{\mathrm{radius}} \in \mathbb{N}_{>0}$ of the circular structuring element used in closing. The same holds for $b^{\mathrm{opening}} \in \{0,1\}$ enabling opening and the opening radius $K^{\mathrm{opening}}_{\mathrm{radius}} \in \mathbb{N}_{>0}$. If both, opening and closing are enabled, closing is applied before opening because applying opening first would remove too much of the original class mask, especially for larger structuring elements, cf. Figures 5.11e and (j). Concatenating the results of these 2-D spatial morphological operations along the temporal dimension yields the enhanced spatiotemporal class mask $\widehat{\Gamma}$ that is the input of the polygon aggregation algorithm described in the next section.

5.6.2 Aggregating Pixels to Polygons

After preprocessing, the enhanced class mask $\widehat{\Gamma} \in \{0,1\}$ is converted into a per-image polygonal representation, where each polygon delineates a spatially contiguous set of pixels with value one in $\widehat{\Gamma}$. Thus each obtained polygon marks an area that is a candidate for being affected by a nano-object adhesion, cf. Figure 5.12. For turning $\widehat{\Gamma}$ into polygons, the marching squares algorithm[9] [BL03] is applied separately to each 2-D spatial class mask $\widehat{\Gamma}(\circ, \circ, t)$ for any given t.

[9]Marching squares is used because then every 2-D polygon has a single temporal coordinate. There is a 3-D generalization of marching squares, called marching cubes [BL03], that can process 3-D class masks directly. Marching cubes yields 3-D polygonal volumes, thus the polygonal representation of a single adhesion event would span multiple points in time, aggravating its temporal localization. In contrast, marching squares, in



(a) $(T \cdot A)(\circ, \circ, t)$ Estimate **(b)** $\widehat{\Gamma}(\circ, \circ, t)$ as an Overlay **(c)** Marching Squares Result

Figure 5.12: Marching Squares – Example. (a) shows the $(T \cdot A)(\circ, \circ, t)$ estimate from which the class mask $\widehat{\Gamma}(\circ, \circ, t)$ in (b) was computed. The shown spatial detail contains one nano-object adhesion, manifesting as upward and downward deviations from the mean intensity value in (a). The corresponding coordinates are marked by ones in $\widehat{\Gamma}(\circ, \circ, t)$, indicated by the green pixels in (b). The black polygon in (c) is the result of applying the marching squares algorithm [BL03] to $\widehat{\Gamma}(\circ, \circ, t)$.

The obtained polygons are placed in a spatiotemporal coordinate system at the temporal coordinate t for which they were created. The result is a set of polygons which are again located in the spatiotemporal domain (x, y, t) of the input data, constituting its segmentation. These polygons are then clustered in a postprocessing, to minimize repeated detections of the same nano-object candidate, cf. Section 5.6.3.

Marching Squares Algorithm

In its most basic variant, the marching squares algorithm [BL03] can be regarded as a window operator applied to a 2-D binary class mask: A 2×2 window is shifted over all coordinate pairs x, y in the class mask, and the four binary values within the window are regarded. This results in $2^4 = 16$ possible configurations of values within the window. Figure 5.13 shows a lookup table of tiles, associating configurations of class mask values with polygons. Class mask values are indicated by dark and light circles, with dark circles representing value one. Polygons are indicated by green areas. Note that there are 16 possible configurations in the class mask but 18 tiles. The two extra tiles arise due to the two ambiguities in row four of the figure: In this row, the polygons can be placed in two different ways. These ambiguities can be resolved by deciding to use either the polygons to the left or to the right of the slash in the figure.

Applying the thus-defined window operator to a class mask yields a set of partial polygons that yet need to be combined to larger polygons. This is done by selecting any tile in the result with at least one and at most three one values in the underlying class mask. Hence

combination with the heuristic described in Section 5.6.3, ideally yields one polygon per nano-object adhesion, and this polygon is located on the image showing the strongest evidence for that adhesion, which is desirable for computing image-based local features, cf. Section 6.2.

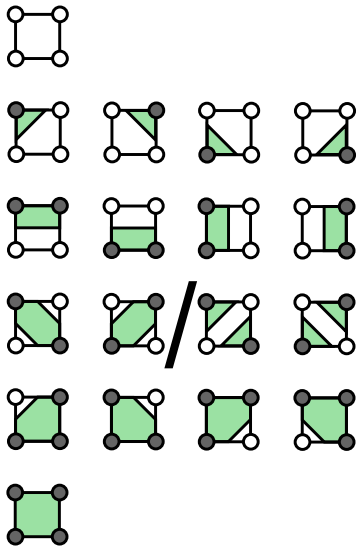


Figure 5.13: Marching Squares – Tiles. A lookup table of tiles associating configurations of class mask values with polygons is shown. Dark and light circles indicate ones and zeros in the class mask, and polygons are displayed as green areas. While there are 16 possible configurations of values in the employed 2×2 window into the class mask, 18 tiles are shown. Two extra tiles are due to the ambiguities of polygon placement in the fourth row of the figure. These ambiguities can be resolved by deciding to use either the polygons to the left or to the right of the slash. Figure adapted from [BL03].

a tile is selected that resides on an edge or a corner in the class mask. Starting from that tile, the adjacent tiles are examined and traced in the direction of the polygon boundary until the starting tile is found again. All polygon points visited during this examination are assembled into a matrix, defining one output polygon, which can be simplified by removing points between subsequent collinear edges. All tiles visited during this process are marked as visited, and the process is iterated until no more unvisited edge or corner tiles exist. Finally, non-compact polygons, as arise e.g. within closed holes of zeros in the class mask, are discarded.

5.6.3 Postprocessing on the Polygon-Level

Applying marching squares for each temporal coordinate t yields a set of polygons in the same spatiotemporal coordinate system as the images recorded by the sensor. In a sliding window approach to time series classification like the fuzzy template matching proposed in Section 5.4, a single nano-object adhesion to the sensor surface may result in high matching scores for several consecutive images, causing positive responses in the class mask $\widehat{\Gamma}$ not only for one temporal coordinate but within a temporal window. The result is that multiple polygons are created within that temporal window which all relate to the same adhesion. A similar phenomenon exists in the spatial domain: The low SNR in the input data may lead to multiple, fragmented polygons being created within a spatial window around a single nano-object adhesion. As a consequence of both phenomena, the polygons output by marching squares need postprocessing, aimed at establishing a one-to-one correspondence between polygons and nano-object adhesions and at choosing those polygons with highest merit for the pattern classifier to be presented in Chapter 6. Such polygons cover the area affected

by the nano-object adhesion to the fullest extent and reside on the $T \cdot A$ image showing the strongest evidence for that adhesion.

In order to find these polygons, the following heuristic is applied: A spatiotemporal clustering of polygons is conducted, using a cylindrical kernel and considering polygon areas. The process works as follows:

1. The polygons are sorted by descending polygon area.
2. For the currently largest polygon, the following is done: A circle with radius $K_{X,Y}^{\text{merge}} \in \mathbb{R}_{\geq 0}$ around the polygon centroid is searched for centroids of other polygons. This is conducted for the temporal coordinate t the polygon resides on, as well as for the $K_T^{\text{merge}} \in \mathbb{N}_{\geq 0}$ preceding and following temporal coordinates, thus making the search region a spatiotemporal cylinder. Choosing this shape for the search kernel respects the cylindrical spatiotemporal structure of nano-object adhesions. Any polygon with its centroid within the cylinder is removed. These polygons are smaller, and thus it is expected that they do not cover the adhesion better than the larger polygon and that they do not reside on an image that shows stronger evidence for the adhesion in terms of image intensity.
3. The second step is repeated with the next smaller remaining polygon, until all remaining polygons have been examined.

As a result, polygons in close spatiotemporal proximity are clustered to a single polygon, chosen as the one with largest area in that cluster. This heuristic aims at eliminating repeated detections of the same nano-object adhesion. In an ideal result, each polygon delineates exactly one candidate object.

The purpose of the polygons output by the overall segmentation module is twofold: Firstly, each polygon indicates (in an ideal case exactly) one nano-object adhesion candidate. Secondly, the polygons delineate the areas over which the features for the pattern classifier are extracted, cf. Section 6.2. These features serve in deciding whether or not such a candidate was caused by an actual nano-object adhesion.

5.6.4 Parameters

The parameters that are optimized for the segmentation module, along with the examined ranges of values are:

Preprocessing on the Pixel-Level

$b^{\text{opening}} \in \{0, 1\}$
: a boolean enabling or disabling morphological opening,

$K_{\text{radius}}^{\text{opening}} \in \{1, \ldots, 5\}$
: an integer determining the radius of the circular structuring element used in morphological opening,

$b^{\text{closing}} \in \{0, 1\}$
: a boolean enabling or disabling morphological closing,

$K_{\text{radius}}^{\text{closing}} \in \{1, \ldots, 5\}$
: an integer determining the radius of the circular structuring element used in morphological closing,

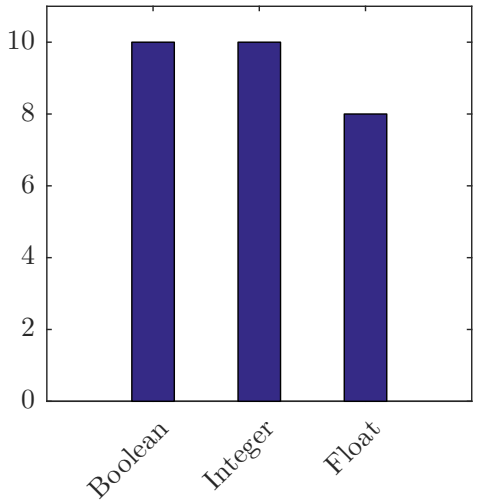


Figure 5.14: Numbers of Detector Parameters by Parameter Type. The pattern detector has ten boolean, ten integer and eight floating point parameters, summing to 28 parameters in total. These parameters are subject to optimization within the *Optimization* stage of *SynOpSis*, cf. top right part of Figure 3.2.

Postprocessing on the Polygon-Level

$K_{X,Y}^{\text{merge}} \in [2, 12]$
: a float determining the spatial radius of the cylinder used in spatiotemporal polygon clustering,

$K_{T}^{\text{merge}} \in \{5, \ldots, 100\}$
: an integer determining the temporal extension of the cylinder used in spatiotemporal polygon clustering.

5.7 Parameters of the Detector

Sections 5.2 to 5.6 each presented methods for realizing one of the modules used in the pattern detector developed for *PAMONO* data analysis, cf. Figure 5.1. The union of these methods[10] gives a full realization of this pattern detector, exhibiting a number of parameters, the roles and purposes of which were discussed in the respective sections.

Figure 5.14 displays the numbers of these parameters by type: There are ten boolean parameters enabling or disabling certain algorithms in the detector, respectively choosing between alternative methods. Furthermore, ten integers control window and kernel sizes and the length of the matched ideal template. Finally, eight floating point parameters control hard and soft thresholds, as well as non-integer kernel properties in filtering and polygon merging, summing to 28 parameters in total.

These 28 parameters of the pattern detector are subject to optimization within the *Optimization* stage of *SynOpSis*, cf. top right part of Figure 3.2. Table 5.6 lists each parameter along with its type, the range of values that was selected to be examined during optimization, the module name and corresponding section where the parameter is defined, and a brief

[10] Fuzzy template matching was chosen over the wavelet-based approach to time series classification, as argued for in Section 5.5.5.

Table 5.6: Parameters Optimized for the Pattern Detector.

Name	Type	Range	Pattern Detector Module	Section	Brief Description
b^{bg}	bool	$\{0, 1\}$	Background Elimination	5.2	Temporal averaging or median
w^{ρ}	int	$\{1, \ldots, 40\}$	Background Elimination	5.2	Past window length
w^{ϕ}	int	$\{1, \ldots, 40\}$	Background Elimination	5.2	Present window length
$b^{\text{denoise}}_{\text{avg}}$	bool	$\{0, 1\}$	Denoising	5.3.1	Spatial averaging filter on/off
$K^{\text{w}}_{\text{avg}}$	int	$\{1, \ldots, 7\}$	Denoising	5.3.1	Averaging filter kernel width
$K^{\text{h}}_{\text{avg}}$	int	$\{1, \ldots, 7\}$	Denoising	5.3.1	Averaging filter kernel height
$b^{\text{denoise}}_{\text{Gauß}}$	bool	$\{0, 1\}$	Denoising	5.3.1	Spatial Gauß filter on/off
$\sigma_{\text{Gauß}}$	float	$[1.5, 3.5]$	Denoising	5.3.1	Gauß filter standard deviation
$b^{\text{denoise}}_{\text{med}}$	bool	$\{0, 1\}$	Denoising	5.3.1	Spatial median filter on/off
$K^{\text{w}}_{\text{med}}$	int	$\{1, \ldots, 7\}$	Denoising	5.3.1	Median filter kernel width
$K^{\text{h}}_{\text{med}}$	int	$\{1, \ldots, 7\}$	Denoising	5.3.1	Median filter kernel height
$b^{\text{denoise}}_{\text{fuzzy}}$	bool	$\{0, 1\}$	Denoising	5.3.2	Fuzzy denoising on/off
l_1	float	$[0.02, 0.1]$	Denoising	5.3.2	Lower soft threshold (denoising)
l_2	float	$[0.1, 0.2]$	Denoising	5.3.2	Upper soft threshold (denoising)
$b^{\text{denoise}}_{\text{bright}}$	bool	$\{0, 1\}$	Denoising	5.3.3	Brightness correction on/off
$b^{\text{denoise}}_{\text{spill}}$	bool	$\{0, 1\}$	Denoising	5.3.3	Overspill compensation on/off
T	int	$\{8, \ldots, 32\}$	Time Series Classification	5.4.1	Length of matched template
h_1	float	$[0.0005, 0.1]$	Time Series Classification	5.4.1	Lower hard threshold (magnitude)
h_2	float	$[0.01, 0.2]$	Time Series Classification	5.4.1	Upper hard threshold (magnitude)
$b^{\text{classify}}_{\text{fuzzy}}$	bool	$\{0, 1\}$	Time Series Classification	5.4.3	Fuzzy or hard thresholding classification
s_1	float	$[0.002, 0.8]$	Time Series Classification	5.4.3	Lower soft threshold (matching score)
s_2	float	$[0.05, 1]$	Time Series Classification	5.4.3	Upper soft/hard threshold (matching score)
b^{opening}	bool	$\{0, 1\}$	Segmentation	5.6.1	Morphological opening on/off
$K^{\text{opening}}_{\text{radius}}$	int	$\{1, \ldots, 5\}$	Segmentation	5.6.1	Opening circle radius
b^{closing}	bool	$\{0, 1\}$	Segmentation	5.6.1	Morphological closing on/off
$K^{\text{closing}}_{\text{radius}}$	int	$\{1, \ldots, 5\}$	Segmentation	5.6.1	Closing circle radius
$K^{\text{merge}}_{X,Y}$	float	$[2, 12]$	Segmentation	5.6.3	Spatial merging distance
K^{merge}_{T}	int	$\{5, \ldots, 100\}$	Segmentation	5.6.3	Temporal merging distance

description of its purpose. Note that for the integer and float parameters, the examined ranges are smaller than the ranges that are feasible in principle: They were chosen to be as small as possible while still providing good objective function values in practice, over a range of different *PAMONO* experiments. Note that by optimizing objective functions that quantify measures of detection quality, the parameters of the image processing algorithms in the detector are optimized to give the best detection-specific image enhancement. They do not aim at image restoration.

5.8 Matching and Labeling

Automatic optimization of algorithmic parameters with respect to the objective functions employed in *SynOpSis* requires that the confusion matrices and other quantities from which these objectives are derived can be automatically evaluated. In the *SynOpSis* approach depicted in Figure 3.2, this takes place in the "Evaluate Objectives" module which is part of the *Optimization* stage. This section describes how this module is realized for *PAMONO* data analysis. For the pattern detector, the objectives to be optimized are defined in Section 3.5.2 and rely on the confusion matrix shown in Table 3.1 and illustrated in Figure 3.4. Automatic computation of this confusion matrix is carried out by **matching** the ground truth known from the *Synthesis* stage (cf. Chapter 4) to the results output by the pattern detector. For the pattern classifier to be presented in Chapter 6, these objectives are defined in Section 3.6.2 and rely on the confusion matrix shown in Table 3.2 and illustrated in Figure 3.6. Automatic computation of this confusion matrix is enabled by **labeling** the matching results, i.e. by assigning ground truth class labels to the output of the pattern detector. Given these ground truth-labeled detector results, the confusion matrix of the classifier can be computed by comparing ground truth labels to those predicted by the classifier. In addition to that, ground truth-labeled detector results serve as training data in supervised learning of that classifier. Performance estimation techniques as presented in Section 3.9 are applied to avoid evaluating objectives on the same data that was used to train the classifier.

Matching

Matching the synthetic *PAMONO* ground truth, created as according to Figure 4.2, to the output of the *PAMONO* pattern detector from Figure 5.1 is carried as follows: Both, ground truth and detector results are represented as polygons, with ground truth polygons generously delineating the area affected by a nano-object adhesion and detector polygon properties depending on the algorithmic parameters of the detector. The matching criterion is point-to-area incidence within a temporal window: Each ground truth polygon area is checked for incident detector polygon centroids within a temporal window of size $\pm T$. Here T is the length of the considered ideal template pattern as defined in Section 5.4.2. It is a parameter of the pattern detector and is subject to optimization, so the temporal size of the matching window automatically adapts to changing processing parameters.

Among the detector polygon centroids that are incident to a ground truth polygon, the detector polygon with largest area becomes the primary match, while the others are listed as repeated detections of the same nano-object. The overall number of repeated detections is required for computing $\widehat{\mathrm{TP}}$, which is the number of true positive detector responses, excluding repeated detections. $\widehat{\mathrm{TP}}$ is used in the definition of the objective M-Rate, cf.

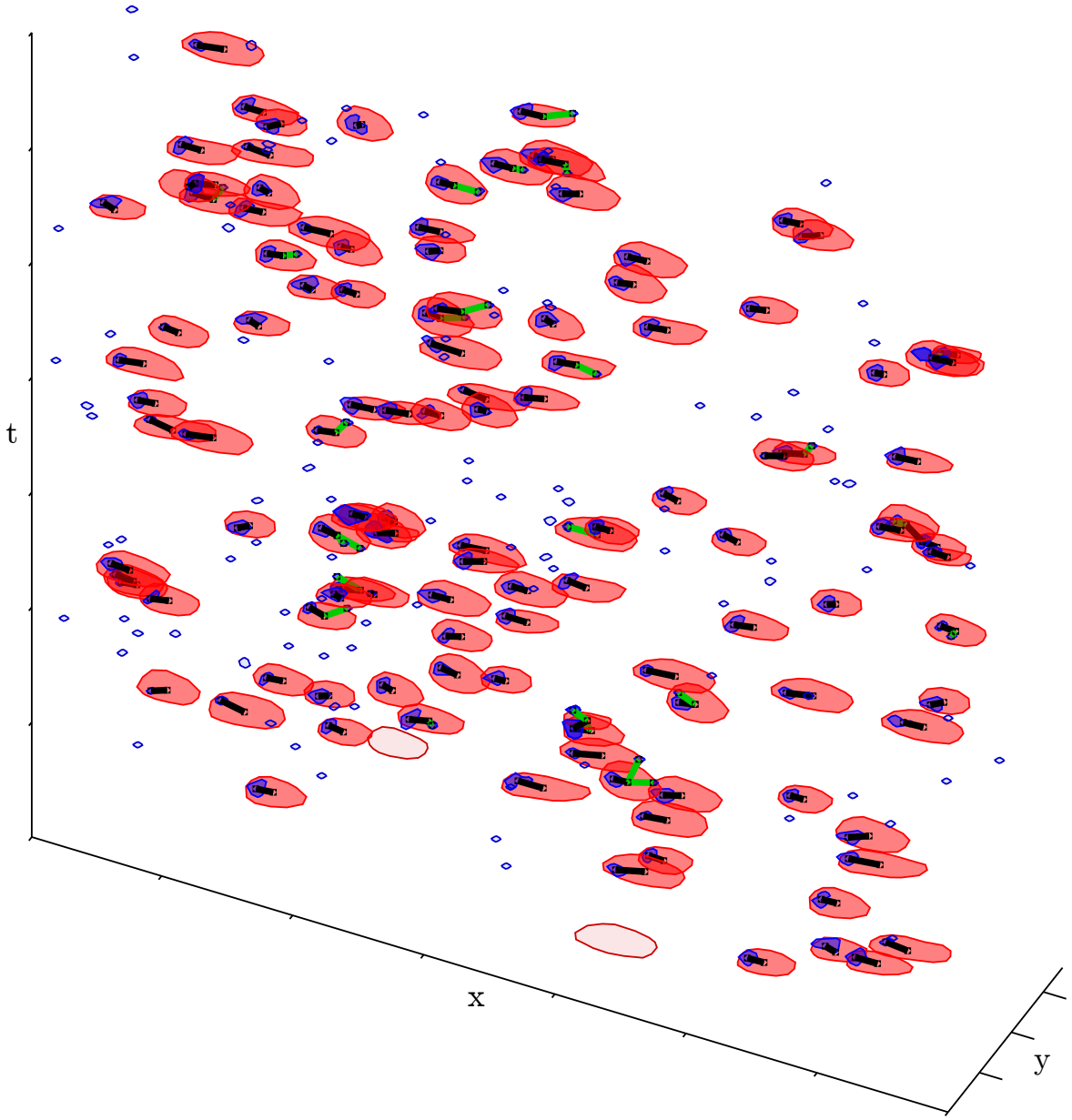


Figure 5.15: Matching – Illustration. A result of matching is shown in a perspective view on the underlying spatiotemporal coordinate system. The x, y dimensions are located at the bottom, while t extends upward. Ground truth polygons are colored red, and detector polygons are blue. A pale color indicates unmatched polygons, while matched polygons are drawn more saturated. There are two False Negatives (FNs), visible as pale red ground truth polygons in the lower part, and a larger number of False Positives (FPs), visible as pale blue detector polygons all over the spatiotemporal volume. Black lines indicate primary matches, thus connecting pairs of True Positives (TPs), while green lines connect ground truth polygons to their repeated detections.

Equation (3.2), as well as in defining a Recall objective that is cleansed from repeated detections, cf. Equation (3.3). As a consequence, repeated detections do not contribute to the objective Recall but are penalized by the objective M-Rate. This is explained in more detail in Section 3.5.2.

The opposite case of a detector polygon centroid that is incident to multiple ground truth polygons is not treated specifically because this condition is rare, and in combination with the previous case, a maximum matching problem on a bipartite graph arises: Ground truth polygons and detected polygons form two disjoint sets of vertices, and the centroid-to-area incidence defines a set of edges connecting only *between* the two sets of vertices, not *within*. This fulfills the definition of a bipartite graph. The task is to find a maximum subset of these edges, matching ground truth with detected polygons, where maximality ensures that in the case a detector polygon matches multiple ground truth polygons, and at least one of those ground truth polygons also matches with another detected polygon, an optimal assignment of detected polygons to ground truth polygons is found and vice versa. In order to solve a maximum matching problem on a bipartite graph efficiently, the algorithm by Hopcroft and Karp [HK73] can be used. As this special case was rarely observed in practice, this is not done.

With these two special cases of multiple matches in both directions discussed, there are three cases remaining, defining the three non-zero entries of the confusion matrix of the detector in Table 3.1:

- A detector polygon with a matching ground truth polygon is counted as a True Positive (TP) detector response.
- A detector polygon without a matching ground truth polygon is counted as a False Positive (FP) detector response.
- A ground truth polygon without a matching detector polygon is counted as a False Negative (FN) detector response (so actually, it is a non-response). FNs can not be corrected for by the pattern classifier because they evaded the detector and are hence not represented in its output. They are counted to evaluate the quality of the detection result, e.g. in terms of detector Recall.

The notion of a True Negative (TN) is not defined in this context of detection because it corresponds to an entity that has neither been detected, nor marked in the ground truth, and hence can not be counted [WHS+12; SLN+09]. In order to accelerate the matching procedure during optimization, an early cancellation criterion is used: If the number of detected polygons is larger than a times the number of ground truth polygons, the matching process cancels and the objectives are set to the worst possible values to indicate this condition and to mark the individual as unattractive for the optimization process. This avoids over-sensitive parameters covering the images in polygons and saves time during optimization because the matching need not be computed for the largest inputs. A value of $a = 5$ was chosen empirically. Outside of optimization the matching is always computed.

Figure 5.15 illustrates the results of matching with an example: A spatiotemporal coordinate system is displayed in a 3-D perspective view. The temporal dimension extends upward, and the two spatial dimensions are at the bottom. Ground truth polygons are drawn in red, while detector polygons are colored blue. Unmatched polygons are indicated by a pale color, while the color of matched polygons is more saturated. As an example, in the lower temporal coordinates, there are two FNs, visible as pale red ground truth polygons, and all

over the spatiotemporal volume there are pale blue detector polygons constituting FPs. All polygons connected by black lines form pairs of TPs, as black lines indicate primary matches. Green lines connect ground truth polygons to their repeated detections.

Labeling

Each polygon output by the detector can belong to one of two classes, as stated in Terminology 3.1 and restated here for the concrete context of *PAMONO*:

- **Target pattern** polygons are caused by actual nano-object adhesions, i.e. the detector response is due to the T component of the $T \cdot A$ signal estimate. This class is equivalent to the TP detector responses discussed in the context of matching.
- **Non-target pattern** polygons are caused by something other than an actual nano-object adhesion. The set of other causes was subsumed under the term 'artifact' and modeled by the artifacts component A in the $T \cdot A$ signal estimate. Thus, these polygons are responses of the detector to signals in the A component, which was not separated from $T \cdot A$ on the pixel level. Instead, these polygons are sorted out by the classifier in Chapter 6. The class of polygons due to A (or due to residual noise) is equivalent to the FP detector responses discussed in the context of matching.

This defines the two-class classification problem tackled via supervised learning in Chapter 6: TP detector polygons are labeled as target patterns, constituting the positive class. FP detector polygons are labeled as non-target patterns, constituting the negative class. Labeled detector polygons are used later in evaluating the objective functions of the classifier in the *Optimization* stage, and in evaluating overall analysis quality. Furthermore they serve as training data for the supervised classifier.

Transferring labels via matching from ground truth polygons to detected polygons is necessary because the ground truth segmentation practice[11] differs from the detector output. The matching procedure translates between ground truth segmentation practice and the detector-generated segmentation. In a fully automatic analysis of unsegmented data, only the latter is available, which is why the classifier must be learned from labeled detector polygons, not ground truth polygons.

5.9 Conclusion

A pattern detector, custom-tailored for the *PAMONO* sensor scenario, was presented, cf. Figure 5.1. Its in- and outputs interface the *SynOpSis* approach presented in Chapter 3, cf. Figures 3.2 and 3.3. The detector consists of four consecutive modules that were presented in Sections 5.2 to 5.6. Section 5.5 proposed an alternative method for realizing the time series classification module, using a condensed k-NN classifier on translation-invariant, wavelet-based features. This approach was ruled out due to runtime considerations, unless accelerated by computation on the GPU. Time series classification via fuzzy template matching was chosen

[11] Ground truth polygons are typically larger than the detected ones: Nano-objects are just delineated generously in the ground truth. An exact reconstruction of the ground truth polygons is not the goal of the detector because the task is *counting* adhesions and covering *the most salient* part of the nano-object signals, in order to extract features for the classifier.

instead, for its real-time capability and because its lower results quality pertains primarily to the type of signal that is the least common and the least important in practice.

The four modules constituting the pattern detector exhibit 28 degrees of freedom in terms of parameter choice, as summarized in Section 5.7. In order to ensure the best-possible analysis quality for each *PAMONO* dataset, these parameters can be automatically optimized via *SynOpSis*. To this end, Section 5.8 presented an application-specific heuristic for matching ground truth to detector results, enabling automatic evaluation of objective functions measuring detection quality and thus the quality of parameter sets.

The output of the detector consists of candidate regions for nano-object adhesions, represented as polygons. Furthermore, the spatiotemporal volume of intensities after background elimination and denoising is part of the output because intensity-based local features used for classifying the polygons are extracted from it later, cf. Section 6.2. If ground truth polygons are present, as e.g. in synthetic data, the detected polygons are annotated with ground truth class labels, which can be used by the supervised learning procedure described in Chapter 6.

An extensive evaluation of the pattern detector and a validation of the overall *SynOpSis* approach will be given in Chapter 7. One part of this focuses on parameter optimization, including choices between the competing algorithms within the modules, cf. Section 7.6. Furthermore, the conjunction of the pattern detector with the pattern classifier will be evaluated on real *PAMONO* data, thus measuring the quality of the results obtained from the overall *SynOpSis* approach, cf. Section 7.5.

Chapter 6

Pattern Classifier for PAMONO

Contents

In the previous chapter, the detection part of the abstract task description in Figure 3.1 was covered. Stated in terms of *PAMONO* data analysis, the output of the pattern detector are spatiotemporal locations that are candidates for being related to target patterns, i.e. nano-objects in the input time series of images. The pattern classifier to be presented in this chapter serves to separate these candidates into actual target patterns and non-target detector responses. It is a concrete realization of the abstract pattern classifier displayed in Figure 3.5.

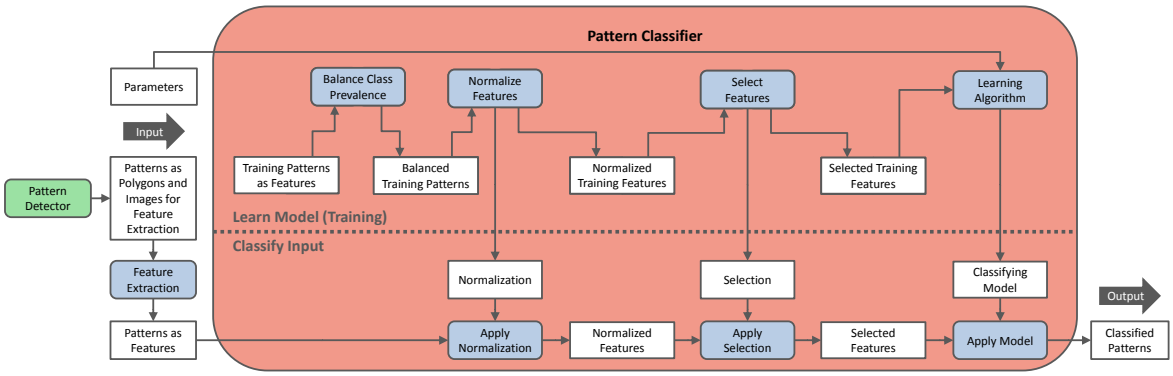


Figure 6.1: Classifier Component for PAMONO. The abstract pattern classifier from Figure 3.5 is realized for *PAMONO* as shown above. Its inputs are the output of the pattern detector from Figure 5.1, along with algorithmic parameters for a learning algorithm used to train a classifying model. Feature extraction transforms the images and polygons provided by the detector to the feature vectors employed by the classifier. Labeled synthetic training data is used in a supervised learning procedure depicted in the upper part of the pattern classifier. This procedure consists of three optional preprocessing steps and a learning algorithm. As depicted in the lower part of the pattern classifier, input patterns to be classified undergo the same preprocessing, before the learned model is applied to assign a class label to each input pattern.

The features used for classification specifically aim at the *PAMONO* application scenario, while the overall machine learning and classification process is more general. The concrete pattern classifier described here has the same interface as the abstract one in Figure 3.5 and can therefore implement the two instances of the pattern classifier in *SynOpSis*, cf. Figure 3.2.

Section 3.6 identified Precision and Recall as suitable objectives to be optimized for the pattern classifier within the *Optimization* stage of *SynOpSis*. The optimization itself will not be covered further within this chapter, as the focus lies on the internals of the pattern classifier and on feature extraction. However, the algorithmic parameters that are to be optimized with respect to these objectives will be introduced in detail.

After Section 6.1 gave an overview of the concrete pattern classifier, Section 6.2 presents the set of features that are extracted for *PAMONO* data analysis. Sections 6.3 to 6.6 cover the internal components of the pattern classifier, including supervised learning algorithms to compute classifying models. Section 6.7 provides results comparing different variants of realizing the pattern classifier, and based on this evaluation, three design decisions are taken. The parameters remaining after these choices are summarized in Section 6.8, and finally, Section 6.9 gives conclusions and an outlook on future work concerning the pattern classifier.

6.1 Introduction

The pattern classifier to be presented in this chapter is a heavily extended version of the work in [SWL+11]. Major additions include new features for classification, balancing of class prevalence and a Random Forest [Bre01] learning algorithm, which will all be presented throughout this chapter. Furthermore, the resulting evolved version of [SWL+11] was embedded into the *SynOpSis* approach, enabling a multi-objective optimization of its algorithmic parameters.

Pattern classification in the context of *PAMONO* data analysis addresses the separation of the $T \cdot A$ term in Equation (4.1). Separating the target patterns in T from the non-target

patterns in A *by division on the pixel level* is an ill-posed inverse problem because only an estimate of the product signal $T \cdot A$ is known for real data: Given only the their product, the individual components can not be separated uniquely on the pixel level. The pattern classifier hence addresses this separation *by classification on the pattern level*: Each detector response constitutes a candidate pattern to be classified. These candidates are represented in terms of application-specific features to be described in Section 6.2, which are designed to separate responses induced by T from those induced by A. To this end, the employed features collect information about local spatial and spatiotemporal intensities, and shape of the response region for each candidate pattern. The target patterns, i.e. the patterns due to nano-objects, exhibit a characteristic geometrical and visual appearance, allowing their discrimination from spurious non-target detector responses. This discrimination is realized by the classification process depicted in Figure 6.1. Before a guided tour through this figure is conducted, the required terminology will be defined.

Terminology 6.1. *The classification problem addressed by the pattern classifier distinguishes two classes, called* ***target*** *and* ***non-target****. The target class is the* ***positive class*** *and corresponds to the True Positive (TP) detector responses. The non-target class is the* ***negative class*** *and corresponds to the False Positive (FP) detector responses.* ***Classifying*** *a candidate pattern means assigning it to one of the two classes. The name of the class assigned to a pattern is also denoted as its* ***(class) label****. Two types of labels are distinguished:* ***Ground truth labels*** *are obtained by matching detected patterns either to synthetic ground truth or to ground truth obtained by letting a human expert manually annotate real* PAMONO *sensor data, cf. Section 5.8.* ***Predicted labels*** *are the labels predicted by a classifying model which is applied to feature vectors extracted from the candidate patterns provided by the detector. In the context of classification, detected patterns are often referred to as examples. An* ***example*** *consists at least of a feature vector and may optionally be annotated with a class label, making the tuple consisting of the feature vector and the label a* ***labeled example****. Note that examples can be either* ***ground truth-labeled*** *or* ***prediction-labeled****, depending on the source of the label.*

Given the prerequisites from Terminology 6.1, an overview of the classification process in Figure 6.1 can be given. The organization of this chapter follows the order in this process.

1. **Feature extraction** (cf. Section 6.2) constitutes a connector between the pattern detector and the pattern classifier. Unlike the detector, the classifier does not operate on per-pixel intensity data but on per-polygon feature vectors, and the feature extraction module is responsible for the corresponding transformation. Its input is the output of the pattern detector, consisting of detected patterns represented as polygons and an estimate of the $T \cdot A$ component. Each polygon is located on one of the images in $T \cdot A$, which are used for extracting intensity-based features. In the output of feature extraction, each pattern is represented in terms of a vector of feature values.
   Labeled and unlabeled feature vectors are used by the classifier, aiming at two different purposes:
   (a) Labeled feature vectors are used in the supervised learning procedure depicted in the upper part of Figure 6.1. In the figure, these vectors are denoted as "Training Patterns as Features". These labeled examples are created by matching synthetic ground truth to detected patterns. The ground truth labels provided by matching

enable utilization of supervised learning algorithms. Training examples must be represented in the same feature space as the input to be classified, and they must be generated using the same parameter set for the pattern detector. These labeled feature vectors undergo the preprocessing described in points 2. to 4. and are used to learn the classifying model as discussed in point 5.

(b) Unlabeled feature vectors are the representation of the input candidate patterns to be classified, as depicted in the lower part of Figure 6.1. Feature vectors are classified by applying the classifying model learned in the training phase, after they underwent the same preprocessing in terms of normalization and feature selection. Hence the actual classification of the input patterns consists of applying the outputs generated in the training phase.

2. **Balancing class prevalence** (cf. Section 6.3) is the first optional preprocessing module that can be employed in the training phase of the pattern classifier, cf. upper part of Figure 6.1. It is applied solely to the training data and serves to balance the frequencies with which the two class labels occur in the training data. Doing so is important for some learning algorithms to prevent class label prevalence from becoming an implicit weight on class importance [HG09].
3. **Feature scale normalization** (cf. Section 6.4) is the second optional preprocessing module. It serves to put the values observed over different feature dimensions on the same scale, thus enabling comparison of feature values across dimensions and preventing large-valued feature co-domains from acting as an implicit weighting factor on feature importance, e.g. in Euclidean distance computations. Feature scale normalization is only required for learning algorithms involving feature comparisons across dimensions.
4. **Feature selection** (cf. Section 6.5) is the third and last optional preprocessing module. It serves to determine those dimensions of feature space that are particularly important with regard to the class label. This can be exploited e.g. for faster model application by restricting the learning algorithm to use only the important features, or it can be utilized to assess the merit of the extracted features.
5. **Learning algorithms** (cf. Section 6.6) constitute the only non-optional module in the training phase. A learning algorithm receives the (optionally preprocessed) training data as input, along with algorithmic parameters configuring that learning algorithm. Via supervised learning it creates an abstraction of the labeled training data and encodes this abstraction in a classifying model which is used to predict class labels for the unlabeled input examples.

Note that during the *Optimization* stage of *SynOpSis*, i.e. in the upper pattern classifier depicted in Figure 3.2, the classification is conducted within a five-fold cross-validation [Koh95], cf. Section 3.9. The reason is that in the *Optimization* stage, the task of the pattern classifier is not computing a classifying model but measuring the quality of the current parameter set of the learning algorithm in terms of the objective functions to be optimized. The cross-validation serves to avoid undue optimism in evaluating these objectives, and the output objectives are the averages achieved over its folds.

Feature extraction and applying the classifying model are real-time capable and run on the Graphics Processing Unit (GPU) [Lib15a; Lib15b], thus attaining real-time capability of the *Application* stage of *SynOpSis*.

6.2 Feature Extraction

Feature Extraction is the first module from Figure 6.1 to be presented. It precedes the actual pattern classifier and receives the outputs of the pattern detector from Figure 5.1 as inputs. Consequently, the input of feature extraction are the polygons from segmentation and the processed images estimating the $T \cdot A$ component of Equation (4.1) which are provided by the denoising module of the detector. Now the features to be extracted aim at separating T from A on the polygon level by aggregating information from all three dimensions of the input data over a local region defined by each polygon.

Since the $T \cdot A$ estimate is used as the data source from which the intensity-based features are extracted, neighboring pixel intensities are comparable: The high-amplitude, high-frequency background signal B dominating the $T \cdot A$ components in the original input has been removed. Furthermore, the noise component N has been attenuated, thus improving feature quality.

Like for the pattern detector, all computations involved in feature extraction are carried out in a streaming setting on the GPU [Lib15a], attaining real-time capability of the *Application* stage of *SynOpSis* and accelerating global optimization, where features have to be extracted for each examined detector parameter set in order to compute classification performance. The inputs of feature extraction already reside in GPU memory because they are computed there, within the detector. Hence, the speedup of parallel GPU computing can be exploited without incurring costs in terms of additional memory transfer time. Furthermore, the intensity-based features involve computing per-pixel feature maps as an intermediate step, with only local operations to be executed on each pixel. This lends itself to an efficient GPU implementation [Lib15a].

The output of feature extraction consists of one feature vector per input pattern, where each input pattern is represented as a polygon in the spatiotemporal coordinate system underlying $T \cdot A$. Feature vectors aim at separating classes by characterizing polygon shape and the underlying spatial and spatiotemporal intensity distributions. A total number of 67 real-valued features is computed, thus producing one feature vector $\mathbf{f} \in \mathbb{R}^{67}$ for each polygon in the input.

The extracted features can be divided into three different categories: Features of polygon shape are presented in Section 6.2.1. Section 6.2.2 is concerned with features of local intensity distribution in the spatial dimensions, while Section 6.2.3 presents features of local intensity distribution in the spatiotemporal domain, i.e. over all dimensions of $T \cdot A$.

6.2.1 Features of Polygon Shape

Features of polygon shape measure geometric properties of the vector of points defining each polygon that is output by the pattern detector in Figure 5.1. Temporal coordinates of polygons are not considered. Most of the computed shape features are based on the work by Landini [Lan06] and have been examined in the context of *PAMONO* in [SWL+11]. Furthermore, intensity-weighted central moments [Hu62] and an intensity-weighted measure of polygon elongation [JT81] have been added, resulting in the overall feature list presented in the following.

List of Features of Polygon Shape

f^{area}

The first feature f^{area} [Lan06] measures the area covered by the polygon.

f^{perim}

The perimeter feature f^{perim} [Lan06] is the sum of the lengths of all line segments delineating the polygon.

$f^{\text{width}}_{\text{AABB}}$

Feature $f^{\text{width}}_{\text{AABB}}$ [SWL+11] is defined with respect to the Axis-Aligned Bounding Box (AABB) of the polygon. The AABB is the smallest rectangle that encompasses the polygon and consists solely of lines that are parallel to one of the axes of the image coordinate system. Feature $f^{\text{width}}_{\text{AABB}}$ is the extension of the AABB in the x-direction.

$f^{\text{height}}_{\text{AABB}}$

Analogously, $f^{\text{height}}_{\text{AABB}}$ [SWL+11] measures the extension of the AABB in the y-direction.

$f^{\text{axis1}}_{\text{OBB}}$

The following features are defined with respect to the Oriented Bounding Box (OBB). The OBB is defined as the smallest rectangle that encompasses the polygon, but in contrast to the AABB, the sides of the rectangle may be rotated in the image plane. Then $f^{\text{axis1}}_{\text{OBB}}$ [Lan06] is the length of the longer side of that rectangle.

$f^{\text{axis2}}_{\text{OBB}}$

Analogously to $f^{\text{axis1}}_{\text{OBB}}$, $f^{\text{axis2}}_{\text{OBB}}$ [Lan06] is the length of the shorter, perpendicular side of the rectangle.

$f^{\text{area}}_{\text{OBB}}$

The area $f^{\text{area}}_{\text{OBB}}$ [Lan06] of the OBB is computed as the product of $f^{\text{axis1}}_{\text{OBB}}$ and $f^{\text{axis2}}_{\text{OBB}}$.

$f^{\text{aspect}}_{\text{OBB}}$

A further feature related to the OBB is its aspect ratio $f^{\text{aspect}}_{\text{OBB}}$ [Lan06], i.e. $f^{\text{axis1}}_{\text{OBB}}$ divided by $f^{\text{axis2}}_{\text{OBB}}$.

$f^{\text{axis1}}_{\text{ori}}$

The orientation feature $f^{\text{axis1}}_{\text{ori}}$ [Lan06] is defined as the smaller angle enclosed by the longer side of the OBB and the x-axis.

f^{rect}

Rectangularity f^{rect} [Lan06] is measured as the ratio between the polygon area f^{area} in the numerator and the area $f^{\text{area}}_{\text{OBB}}$ of the OBB in the denominator. Since $f^{\text{area}}_{\text{OBB}} \geq f^{\text{area}}$, the maximum value one is achieved if the polygon is perfectly rectangular.

f^{circul}

Circularity f^{circul} [Lan06] is defined as

$$f^{\text{circul}} = \frac{4\pi f^{\text{area}}}{f^{\text{perim}} f^{\text{perim}}} \tag{6.1}$$

and measures the similarity of the polygon to a circle: The maximum value one is attained if the polygon is a circle, and it decreases with increasing deviation of the edges of the polygon from approximating a circle.

f^{compact}

Compactness f^{compact} [Lan06], which is defined as

$$f^{\text{compact}} = \frac{\sqrt{\frac{4}{\pi} f^{\text{area}}}}{f^{\text{axis1}}_{\text{OBB}}}, \tag{6.2}$$

also attains its maximum value one for circles but decreases as the polygon becomes *lengthier*.

f^{C_2}

The central moment feature f^{C_2} [JT81] is an intensity-weighted feature of polygon shape, relying on the concept of 'moments of intensity', originally introduced by Hu [Hu62]: The coordinates (x, y) of all pixels residing inside the polygon are weighted with their relative contribution to the total sum of intensities inside the polygon. In the *PAMONO* case, these intensities are the $T \cdot A$ estimate output by the detector, cf. Figure 5.1. The intensity-weighted polygon centroid $(\bar{x}, \bar{y})$ is computed as the respective mean value over all intensity-weighted (x, y) coordinates inside the polygon. Given $(\bar{x}, \bar{y})$, the higher central moments $m_{i,j} \in \mathbb{R}$ for $i, j \in \mathbb{N}_{\geq 0}, i + j \geq 2$ are defined as

$$m_{i,j} = \sum_{(x,y) \text{ in Polygon}} (x - \bar{x})^i (y - \bar{y})^j (T \cdot A)(x, y). \tag{6.3}$$

Relying on this equation, the central moment feature f^{C_2} is defined as

$$f^{C_2} = \frac{m_{2,0} + m_{0,2}}{m_{0,0}} \tag{6.4}$$

with

$$m_{0,0} = \sum_{(x,y) \text{ in Polygon}} (T \cdot A)(x, y). \tag{6.5}$$

f^{C_4}

Similarly, the central moment feature f^{C_4} [JT81] is defined as a fourth-order analogon of the second-order feature f^{C_2}:

$$f^{C_4} = \frac{m_{4,0} + 2m_{2,2} + m_{0,4}}{m_{0,0}}. \tag{6.6}$$

f^{elong}

The last considered feature based on moments is the elongation feature f^{elong} [JT81], which is similar to f^{compact} but in contrast to that incorporates intensity information. It is defined as

$$f^{\text{elong}} = \frac{\sqrt{(m_{2,0} - m_{0,2})^2 + 4m_{1,1}^2}}{m_{2,0} + m_{0,2}}. \tag{6.7}$$

As a summary on the intensity-moments-based features f^{elong}, f^{C_2} and f^{C_4}, the following can be said: The elongation feature f^{elong} specifically targets elongated objects. The central moment feature f^{C_2} collects intensity-weighted information about the variance of the coordinate distribution in the polygon, while f^{C_4} uses information about its kurtosis (an intuition behind the notion of kurtosis will be given in the discussion of feature $f^{\text{kurt}}_{\text{time}}$ in Section 6.2.3). Jarvis and Tyson state that f^{C_2} and f^{C_4} are strongly correlated but that providing information about kurtosis along with variance proved beneficial in their subsequent classification [JT81].

6.2.2 Features of Spatial Intensities

In extracting the features of spatial intensities, the employed intensities are again the $T \cdot A$ estimate as output by the detector in Figure 5.1. Since the features are spatial, their extraction is conducted per image, enabling a more convenient notation by dropping the temporal index and designating $T \cdot A$ by a single letter: Let $\eta(x, y)$ denote the image $(T \cdot A)(x, y, t_c)$ under consideration at a certain point of time t_c. The examined features are computed over the entire domain of such an image. In order to obtain per-polygon features from these per-pixel features, each polygon on the image is assigned the maximum feature value observed within the polygon boundary.

The first four features of spatial intensities to be presented rely on the Hessian of $\eta(x, y)$, i.e. on the matrix containing all second-order partial derivatives of $\eta(x, y)$ [MSB+13]. This concept of the Hessian is extended into a scale-space, which usually means that several convolutions of $\eta(x, y)$ with Gaussian kernels of increasing standard deviation σ are considered [Lin94], cf. Equation (5.7). Note, however, that the Hessian matrix is concerned solely with derivatives, so the computation of the Gaussian-filtered images in the usual space-space is unnecessary. Instead, the Gaussian-based derivative operators from [FA91] are used, and the scale-space parameter σ is incorporated into their analytic representation from which the discrete second-order partial Gaussian derivative operators $\partial^\sigma_{xx}, \partial^\sigma_{xy}, \partial^\sigma_{yx}, \partial^\sigma_{yy}$ are sampled: These operators each derive a 2-D Gaussian function with standard deviation σ (Equation (5.7)) into the indicated directions and are applied as derivative operators by convolving them with $\eta(x, y)$. Increasing σ in this process yields derivatives on coarser levels of scale space. Given these operators, the per-pixel Hessian matrix $\mathbf{H}^\sigma(x, y)$ of $\eta(x, y)$ on scale σ is defined as

$$\mathbf{H}^\sigma(x, y) = \begin{bmatrix} \partial^\sigma_{xx}\eta(x, y) & \partial^\sigma_{xy}\eta(x, y) \\ \partial^\sigma_{yx}\eta(x, y) & \partial^\sigma_{yy}\eta(x, y) \end{bmatrix}. \tag{6.8}$$

It captures information about local intensity curvature. For increasing scales σ in the derivative operators, this curvature information considers larger local regions of the image $\eta(x, y)$. Using this scale-space extension ensures that target patterns can be characterized independent of their scale, and since every examined σ defines new features, information from all these scales can be considered in the classification. For *PAMONO*, ten linearly-spaced scales were considered, starting with $\sigma = 0.7$ and ending with $\sigma = 7$. Therefore, each of the features presented in the following that is subscripted with σ, actually denotes ten features of the same type, evaluated on different scales.

List of Features of Spatial Intensities

f^{trace}_σ

The first feature f^{trace}_σ to be computed from the scale-space extension of the Hessian is defined as its trace:

$$f^{\text{trace}}_\sigma(x, y) = \text{tr}(\mathbf{H}^\sigma(x, y)) = \partial^\sigma_{xx}\eta(x, y) + \partial^\sigma_{yy}\eta(x, y). \tag{6.9}$$

This definition is equivalent to the image Laplacian [GW07]. It measures total curvature in the direction of both image axes. The per-pixel values $f^{\text{trace}}_\sigma(x, y)$ are aggregated to a single per-polygon value by assigning each polygon the maximum feature value observed within the polygon. Unless stated otherwise, the same holds for all features presented

in this and the next section. The rationale behind using the maximum is as follows: Within the area covered by a polygon, feature values typically vary. For example, with a curvature-based feature like the Laplacian in f_σ^{trace}, there is typically a single point of maximum curvature within a polygon. Using the maximum for aggregation yields the feature value in this point. In contrast, using e.g. the mean makes the feature value depend on polygon size because the feature value decreases, the more pixels the polygon covers that exhibit lower curvature. This behavior is undesired, hence the maximum is used as the final feature value. Its susceptibility to outliers is counteracted by the fact that the underlying intensities already underwent a denoising procedure and by the smoothing incorporated in the Gaussian-based derivative operators.

f_σ^{det}

The determinant feature f_σ^{det} is the scale-space extension of a feature used by Thomann et al. [TRS+02]. As the name suggests, it is defined as the determinant of the Hessian:

$$f_\sigma^{\text{det}}(x,y) = \det(\mathbf{H}^\sigma(x,y)) = \partial_{xx}^\sigma \eta(x,y)\partial_{yy}^\sigma \eta(x,y) - \partial_{xy}^\sigma \eta(x,y)\partial_{yx}^\sigma \eta(x,y). \tag{6.10}$$

It quantifies the local intensity curvature in image coordinate (x,y). Regarding the two rows of $\mathbf{H}^\sigma(x,y)$ as vectors spanning a parallelogram, the absolute determinant gives the area of that parallelogram, hence f_σ^{det} is invariant to the relative orientation of intensity curvature and responds particularly to areas where curvature is large in both directions, as e.g. in circular structures like the target patterns in Figure 3.1. In contrast to that, artifacts like the waves in Figure 3.1 are edge-like, exhibiting high intensity curvature only in one direction. Detector responses due to noise have low curvature in all directions. Hence f_σ^{det} is suitable to separate spurious detector responses from target patterns.

f_σ^{spot}

The fact that intensities around target patterns are expected to be higher than those caused by artifacts (cf. Figure 3.1) leads to the spot feature f_σ^{spot}, which weights curvature $f_\sigma^{\text{det}}(x,y)$ with intensity $\eta(x,y)$. It is as well a scale-space extension of a feature from [TRS+02] and is defined as:

$$f_\sigma^{\text{spot}}(x,y) = f_\sigma^{\text{det}}(x,y)\eta(x,y). \tag{6.11}$$

This product feature responds particularly to spots of high local curvature that are *brighter* than others because curvature and intensity are regarded in conjunction.

f_σ^{blob}

A further feature f_σ^{blob}, responding to blob-like structures, can be deduced from the eigenvalues of the Hessian $\mathbf{H}^\sigma(x,y)$: Let $|\lambda_1(x,y)| \le |\lambda_2(x,y)|$ denote the two eigenvalues of $\mathbf{H}^\sigma(x,y)$, sorted by increasing absolute value. The scale σ is omitted in the following equations to avoid cluttered notation. Then the blob feature $f_\sigma^{\text{blob}}(x,y)$ [MSB+13] is composed of the following two quantities:

$$\alpha(x,y) = \sqrt{\lambda_1^2(x,y) + \lambda_2^2(x,y)}, \quad \beta(x,y) = \frac{|\lambda_1(x,y)|}{|\lambda_2(x,y)|}. \tag{6.12}$$

Values of $\beta(x,y)$ that are close to the maximum of one vote for blob-like intensities around (x,y) because the amount of curvature $\lambda_2(x,y)$ in the direction of maximum local curvature is not much larger than the amount of curvature $\lambda_1(x,y)$ in the orthogonal

direction. This happens at circular, blob-like structures. Lower eigenvalue ratios suggest more elongated structures in $\eta(x, y)$. However, $\beta(x, y)$ responds to blobs of arbitrarily low overall curvature because eigenvalue *ratios* are considered. Complementing that, $\alpha(x, y)$ measures eigenvalue magnitudes, such that highly curved blobs are characterized by high values in both, $\alpha(x, y)$ and $\beta(x, y)$. One way of formalizing this in a single scalar is the blob feature by Moon et al. [MSB+13], which is defined as

$$f_\sigma^{\text{blob}}(x,y) = \begin{cases} \left(1 - \exp\left(-\frac{\alpha(x,y)}{2a^2}\right)\right)\left(1 - \exp\left(-\frac{\beta(x,y)}{2b^2}\right)\right) & \text{if } \lambda_1(x,y), \lambda_2(x,y) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6.13}$$

Negativity of both eigenvalues indicates a blob associated with a local *maximum* of intensity and is thus enforced by the condition. If at least one of the eigenvalues is not negative, f_σ^{blob} is defined to be zero because the target patterns associated with nano-objects do not cause blobs of locally minimal intensity or intensity saddle points. The weights a and b allow to control whether f_σ^{blob} is more sensitive to curvature magnitude or roundness. Both weights were set to 0.5 as recommended in [MSB+13].

$f_{\text{space}}^{\max}$

As intensities in target patterns are expected to be larger than residual noise intensities (cf. Figure 4.1b), the feature $f_{\text{space}}^{\max}$ is computed as the maximum intensity in η, observed within the area covered by each detected polygon.

$f_{\text{space}}^{\text{mean}}$

Analogously, $f_{\text{space}}^{\text{mean}}$ is the corresponding average intensity.

$f_{\text{space}}^{\text{std}}$

Furthermore, $f_{\text{space}}^{\text{std}}$ is the corresponding standard deviation of the intensities underlying the polygon area, which is also expected to assume larger values for target patterns than for residual noise.

6.2.3 Features of Spatiotemporal Intensities

Features of spatiotemporal intensities integrate intensity information from all three dimensions of the data provided by the *PAMONO* sensor. Like the purely spatial features, they are computed from the denoised $T \cdot A$ signal. All features to be presented here are computed from time series of intensities, measured in the same sliding window along the temporal axis that was used in background elimination, cf. Section 5.2. Therefore, in the following, $\mathbf{v} \in \mathbb{R}^W$ denotes a single such time series $(T \cdot A)(x_c, y_c, t_c - w^\phi + 1 \ldots t_c + w^\rho)$ observed at a certain image pixel (x_c, y_c), within a temporal sliding window of length W, located around the temporal coordinate t_c of the polygon. Here $W = w^\phi + w^\rho$ is the total length of the foreground and background window, cf. Figure 5.2a for an illustration.

For all eligible coordinates (x, y, t), such a time series $\mathbf{v} = (v_1, \ldots, v_W)$ is regarded, and the time series features presented in the following list are computed. Indices (x, y, t) are dropped for a less cluttered notation. While this covers the temporal dimension, information from the spatial dimensions is considered by aggregating time series features over each polygon: Again the maximum feature value observed in the area covered by a polygon is used as the aggregated feature value assigned to that polygon. The list of time series features computed in this process was used by Janidarmian, Radecka, and Zilic in the context of classifying knee pathology [JRZ14] from Electromyogram (EMG) and goniometer data.

$f_{\text{time}}^{\text{mean}}$

Feature $f_{\text{time}}^{\text{mean}}$ is the temporal analogon of the spatial $f_{\text{space}}^{\text{mean}}$ and is computed as the mean intensity over all values in $\mathbf{v}$, cf. also Equation (5.16).

$f_{\text{time}}^{\text{cross}}$

Feature $f_{\text{time}}^{\text{cross}}$ counts the number of times that $\mathbf{v}$ crosses its mean value $f_{\text{time}}^{\text{mean}}$. It is analogous to the number of zero-crossings after the mean has been subtracted.

$f_{\text{time}}^{\text{std}}$

Feature $f_{\text{time}}^{\text{std}}$ is the temporal analogon of the spatial $f_{\text{space}}^{\text{std}}$ and is computed as the standard deviation of the intensities in $\mathbf{v}$, cf. also Equation (5.17).

$f_{\text{time}}^{\text{CV}}$

The coefficient of variation $f_{\text{time}}^{\text{CV}}$ is the standard deviation $f_{\text{time}}^{\text{std}}$ divided by the mean $f_{\text{time}}^{\text{mean}}$. As units are eliminated by normalizing with $f_{\text{time}}^{\text{mean}}$, the coefficient of variation is better suitable than $f_{\text{time}}^{\text{std}}$ for comparing time series with different ranges of intensity.

$f_{\text{time}}^{\text{skew}}$

The skewness feature $f_{\text{time}}^{\text{skew}}$ examines the distribution of intensities in $\mathbf{v}$ and, as an illustration, measures whether the histogram of intensities is 'more heavy' on the left side of the mean ($f_{\text{time}}^{\text{skew}} < 0$) or on the right side of the mean ($f_{\text{time}}^{\text{skew}} > 0$). It is computed as

$$f_{\text{time}}^{\text{skew}} = \frac{1}{\left(f_{\text{time}}^{\text{std}}\right)^3 W} \sum_{i=1}^{W} \left(v_i - f_{\text{time}}^{\text{mean}}\right)^3 . \tag{6.14}$$

$f_{\text{time}}^{\text{kurt}}$

Kurtosis $f_{\text{time}}^{\text{kurt}}$ is a fourth-order analogon of skewness, defined as

$$f_{\text{time}}^{\text{kurt}} = \frac{1}{\left(f_{\text{time}}^{\text{std}}\right)^4 W} \sum_{i=1}^{W} \left(v_i - f_{\text{time}}^{\text{mean}}\right)^4 . \tag{6.15}$$

Kurtosis attains larger values for time series with an intensity distribution with few large outliers than for one with many small outliers.

$f_{\text{time}}^{\text{L1AC}}$

The lag-one-autocorrelation feature $f_{\text{time}}^{\text{L1AC}}$ measures the cross-correlation of the time series $\mathbf{v}$ with itself, shifted by one temporal coordinate unit:

$$f_{\text{time}}^{\text{L1AC}} = \frac{\sum_{i=1}^{W-1} \left(v_i - f_{\text{time}}^{\text{mean}}\right)\left(v_{i+1} - f_{\text{time}}^{\text{mean}}\right)}{\sum_{i=1}^{W} \left(v_i - f_{\text{time}}^{\text{mean}}\right)^2}. \tag{6.16}$$

$f_{\text{time}}^{\text{RMS}}$

The root mean square feature $f_{\text{time}}^{\text{RMS}}$ measures variability of the time series $\mathbf{v}$ about zero (as opposed to the standard deviation $f_{\text{time}}^{\text{std}}$, measuring variability about the mean $f_{\text{time}}^{\text{mean}}$). It is defined as

$$f_{\text{time}}^{\text{RMS}} = \sqrt{\frac{1}{W} \sum_{i=1}^{W} v_i^2}. \tag{6.17}$$

$f_{\text{time}}^{\text{PPA}}$

The peak-to-peak amplitude $f_{\text{time}}^{\text{PPA}}$ is the difference between the maximum of $\mathbf{v}$ and its minimum. It was already used in time series preselection, cf. Equation (5.9).

Summary

In this section, a total number of 67 features were presented, aimed at separating target patterns from non-target detector responses in a subsequent supervised classification process. These features subdivide into 15 features of polygon shape, presented in Section 6.2.1, 43 features of spatial intensities, presented in Section 6.2.2, and nine features of spatiotemporal intensities presented in Section 6.2.3. The number of 43 spatial features arises from four scale-space features being computed on ten scales each, plus three spatial features being evaluated on the input scale. All features are computed in a real-time-capable streaming setting on the GPU. An evaluation of feature importance and quality will be given in Sections 6.7.3 to 6.7.4.

6.3 Balancing Class Prevalence

After feature extraction, the pattern classifier in Figure 6.1 is used to assign predicted class labels to the obtained feature vectors. In order to do so (lower part of the figure), information from the training phase (upper part of the figure) is required. This and the following three sections describe the four modules used in the training phase, starting with the optional class balancing module.

Algorithms for class balancing receive labeled data as input, possibly exhibiting severe class imbalance, i.e. class prevalence differs considerably between classes. Balancing algorithms serve to even out such differences. Applying a balancing algorithm to the training data before learning the classifying model counteracts the tendency of some learning algorithms to focus on over-represented classes, neglecting the correct classification of the classes for which only few examples occur in the training data. Learning from a dataset with severe between-class imbalance, e.g. with majority class prevalence exceeding minority class prevalence by a factor of 100 or 1000, may result in a large proportion of minority class examples being classified into the majority class. If the minority class is e.g. people bearing a certain pathology, many of these people will be erroneously classified as healthy. Using the *PAMONO* sensor as an example, if viral nano-objects are the minority class, they will be erroneously classified as artifacts, thus preventing the correct diagnosis of an infection and distorting derived estimates of virus concentration [GTÜ+11]. Speaking more generally: If in a given application scenario, the cost of misclassifying a minority example is higher than for misclassifying a majority example, class balancing may serve to reduce costs by reducing the number of misclassified minority examples [HG09].

Besides the issue of between-class imbalance described above, within-class imbalance may pose another problem: It arises if some parts of feature space are better covered with examples of a certain class than others. The sparser parts tend to be smoothed away by learning algorithms. Hence, if the data to be classified contains many examples from such regions, classification quality will be poor. Therefore, some class balancing algorithms focus on sparsely populated regions, thus addressing this issue [HG09]. Note however that distinguishing sparse regions due to lack of examples from sparse regions due to outliers may pose a further issue.

Related Work

He and Garcia give a survey, on the topic of learning from imbalanced data [HG09]. They summarize a number of methods that have been proposed to tackle imbalance in binary

classification problems, hence 'majority' and 'minority' each denote one definite class in the following. As classification in *PAMONO* data analysis is binary as well, the multi-class case is not explicitly covered in this section but a short outlook concerning one approach is given at the end of Section 6.3.2.

The methods surveyed by He and Garcia can, on a coarse level, be divided into the following five categories [HG09]:

- **Random Oversampling and Undersampling:**
  Random over- and undersampling are the simplest methods to modify class balance. Oversampling randomly selects and replicates examples from the minority class, until the desired class ratio has been reached. One drawback is the tendency of overfitting the classifier toward the replicated examples because multiplicity may act as an implicit weight in their favor. Undersampling randomly removes examples from the majority class until the desired class ratio has been reached. Here a major drawback lies in wastage of labeled training data from the majority class.
- **Informed Undersampling:**
  Informed undersampling tries to overcome the drawbacks of random undersampling by compensating for the information loss incurred by leaving out examples. One way of doing so is the EasyEnsemble approach [LWZ09]: Instead of balancing the input for a base learner, an ensemble [HTF09] of multiple instances of the base learner is used, each trained from the full minority set and a different subsample of the majority dataset, attaining a user-definable class ratio. An alternative is BalanceCascade [LWZ09], which constructs an ensemble as a boosted cascade of classifiers [HTF09], where in each iteration, majority class examples are identified as expendable if they are already correctly classified by the classifier from the previous iteration.
- **Synthetic Sampling with Data Generation:**
  Synthetic approaches differ from undersampling because instead of discarding majority examples, minority examples are added. They differ from oversampling because these examples are not replicate copies of minority examples, but they are synthetically created new data points, computed from given minority examples. The Synthetic Minority Over-Sampling Technique (SMOTE) [CBH+02] is an example of such an approach and will be presented in Section 6.3.1.
- **Adaptive Synthetic Sampling:**
  Taking also the majority class into account during synthetic data generation yields *adaptive* synthetic sampling methods: The key component of such techniques is the strategy of how to select minority examples from which to generate synthetic examples. Taking the surrounding majority examples into account within this process adapts the synthesis strategy to the majority set. An example for such a technique is Borderline-SMOTE [HWM05], which creates synthetic examples particularly on the borderline between classes in feature space. A more recent adaptive approach is Adaptive Synthetic Sampling (ADASYN) [HBG+08], which will be presented in Section 6.3.2.
- **Cost-Sensitive Learning:**
  In cost-based approaches to imbalanced learning [PMM+94], the number of examples in the input dataset is not altered, but instead the misclassification costs for minority class examples are chosen larger than for majority class examples. To this end, Domingos proposed a general method for making classifiers cost-sensitive [Dom99].

An alternative applicable in the case of large training datasets like in Section 5.5, is exploiting the clustering part of a condensed k-Nearest Neighbors (k-NN) classifier for class balancing: Letting the clustering compute the same number of cluster centers for each class, balances class prevalence in the training data of the k-NN classifier. This was discussed in detail in Section 5.5.3. For a discussion of further balancing techniques, omitted here for brevity, the reader is referred to [HG09].

The remainder of this section is structured as follows: Section 6.3.1 presents the SMOTE procedure and identifies its major drawback, which is counteracted by the ADASYN algorithm presented in Section 6.3.2. ADASYN uses SMOTE as a subroutine and is the technique employed to realize the class balancing strategy of *SynOpSis*, which is presented in Section 6.3.3.

6.3.1 Synthetic Minority Over-Sampling Technique (SMOTE)

The Synthetic Minority Over-Sampling Technique (SMOTE) [CBH+02] oversamples the minority class without introducing replicates. Instead, new examples are synthesized by linear interpolation between existing examples. The process works as follows: Let $S = S_{\text{maj}} \sqcup S_{\text{min}}$ be the overall unbalanced dataset, disjointly composed of the majority set S_{maj} and the minority set S_{min}. While the desired class ratio has not yet been reached, SMOTE repeats the following procedure: A minority example, represented by its numerical feature vector $\mathbf{f}_a \in \mathbb{R}^F$, is randomly drawn from S_{min}. Then for a given K_s, the K_s nearest neighbors[1] in feature space of $\mathbf{f}_a$ from the set S_{min} are determined. One of these neighbors is randomly drawn as the interpolation target $\mathbf{f}_b \in \mathbb{R}^F$, and a synthetic new example $\widehat{\mathbf{f}} \in \mathbb{R}^F$ is created as

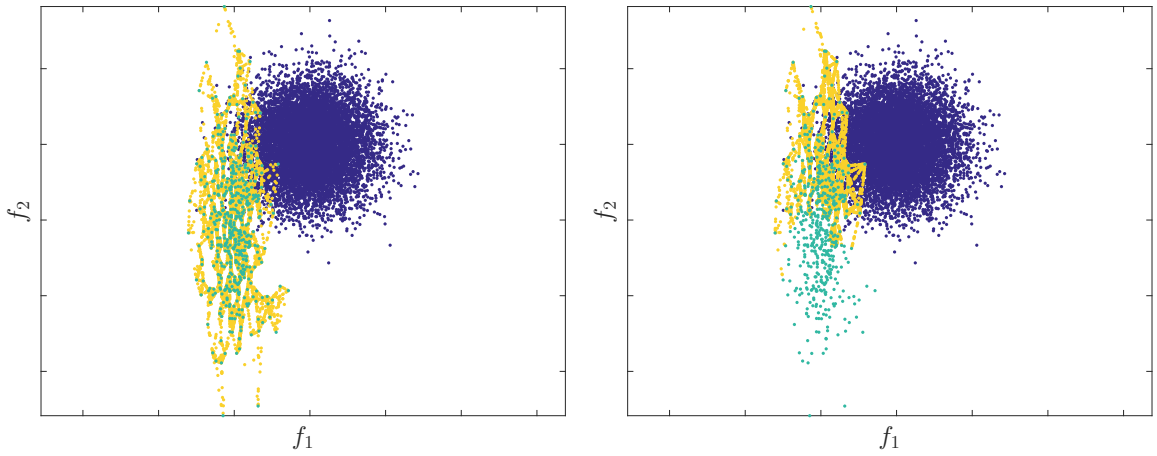
$$\widehat{\mathbf{f}} = \mathbf{f}_a + r(\mathbf{f}_b - \mathbf{f}_a), \tag{6.18}$$

where r is randomly drawn from the open interval $]0, 1[$. Hence the synthetic example $\widehat{\mathbf{f}}$ is convexly interpolated between $\mathbf{f}_a$ and $\mathbf{f}_b$, with higher values of r increasing its similarity to $\mathbf{f}_b$. During this process, all synthetically created examples are stored in the set $\widehat{S}$, and the output dataset with the desired class prevalence is obtained as $B = S_{\text{maj}} \sqcup S_{\text{min}} \sqcup \widehat{S}$.

Figure 6.2a shows the result of applying SMOTE to an imbalanced toy dataset in two-dimensional feature space. The majority class consists of 10000 examples, indicated as blue dots, while the 500 minority examples are colored turquoise. Applying SMOTE up to perfect class balance yields the 9500 synthetic minority examples depicted as yellow dots. They reside on straight lines between minority examples, due to the linear interpolation in Equation (6.18). Such lines exist in the entire convex hull of the minority point set in feature space.

By not introducing replicates into its output, the SMOTE algorithm results in fewer ties in neighborhood computations over examples in feature space, which was demonstrated to

[1] Note that this step assumes normalized feature scales if Euclidean distance is used to measure point distance. For feature scales that are not normalized, the scales of the different dimensions of feature space might be incomparable, with features on larger scales dominating those on smaller scales in Euclidean distance computation. Any of the techniques for normalization to be presented in Section 6.4 can be used for this purpose. It is advisable to apply this normalization only within the context of neighborhood search in SMOTE and not propagate it into further processing steps because the final normalization (if required for the learning algorithm to be employed, e.g. in case of a k-NN or SVM learner, cf. Section 6.6) should also consider the synthetic examples newly created by SMOTE. An alternative to explicit feature scale normalization that was chosen in the *PAMONO* context is to use *standardized* Euclidean distance which implicitly embeds a unit-variance normalization (cf. Section 6.4) into the distance metric.



(a) SMOTE **(b)** ADASYN

Figure 6.2: SMOTE versus ADASYN – Example. (a) shows the result of applying SMOTE in balancing a binary toy problem in two-dimensional feature space. (b) shows the corresponding result of ADASYN. The majority class (blue dots) contains 10000 examples, and the minority class (turquoise dots) has size 500. While SMOTE distributes the creation of synthetic examples (yellow dots) evenly across the original minority examples, ADASYN focuses synthesis in the borderline regions between minority and majority. Examples in these regions are typically harder to learn due to their proximity to class boundaries, and creating more examples of this kind was demonstrated to be beneficial in practice in [HBG+08].

be beneficial for learning algorithms in practice [CBH+02]. As a drawback, the expected same number of synthetic examples is created from each minority example, i.e. regions that are already well-covered with minority examples are treated the same as regions where such examples are rarer. Furthermore, no information about neighboring majority class examples is considered, ignoring that borderline examples, i.e. examples located in between classes in feature space, are the hardest-to-learn examples, typically requiring more data to be correctly represented in the learned classifier than examples in regions where only one class is present [HG09].

6.3.2 Adaptive Synthetic Sampling (ADASYN)

The drawbacks of the SMOTE procedure [CBH+02] mentioned in the previous paragraph are addressed by the Adaptive Synthetic Sampling (ADASYN) approach [HBG+08]. It extends SMOTE with a more advanced procedure for nonuniform selection of seed examples for synthesis: More synthetic examples are created in the vicinity of the boundary between the two classes, than in homogeneous minority class regions. Hence, examples are created particularly in the hard-to-learn borderline regions, cf. Figure 6.2b, which is expected to be beneficial in learning a classifier [HBG+08].

ADASYN works as follows [HBG+08]: Firstly, the number $G \in \mathbb{N}_{\geq 0}$ of examples to be synthesized is computed as

$$G = b(|S_{\text{maj}}| - |S_{\text{min}}|), \tag{6.19}$$

where $b \in [0, 1]$ determines the amount of balancing to be applied. Choosing $b = 1$ means that the ADASYN result will exhibit perfect class balance, while $b = 0$ leaves the input unchanged.

Thus b controls by how much class balance is *improved*, compared to the input dataset. For each minority example $\mathbf{f}_i \in \mathbb{R}^F$ from $S_{\min}$, its K_a nearest neighbors[2] in the full input set $S = S_{\text{maj}} \sqcup S_{\min}$ are determined. As it is known, whether a neighbor is from the majority class, the following ratio $g_i \in [0, 1]$ can be computed for each examined minority example $\mathbf{f}_i$:

$$g_i = \frac{1}{z} \frac{M_i}{K_a}. \tag{6.20}$$

In this ratio, M_i is the number of majority-class neighbors of $\mathbf{f}_i$, hence the right fraction is the relative proportion of majority class neighbors of $\mathbf{f}_i$. The normalization constant z serves to ensure that $\sum_{i=1}^{|S_{\min}|} g_i = 1$, making each g_i the relative proportion of synthetic examples to be created using $\mathbf{f}_i$ as the seed example. This implements the heuristic of creating synthetic examples particularly for minority examples surrounded by a large proportion of majority examples. The relative proportions g_i are converted to absolute per-example values G_i by computing

$$G_i = g_i G \tag{6.21}$$

and rounding the result G_i to the nearest integer. As the g_i were normalized to sum to one, the G_i sum to the total number G of synthetic examples to be created (ignoring rounding errors). As a last step, the SMOTE procedure from Section 6.3.1 is used to create G_i synthetic examples for each original minority example $\mathbf{f}_i$. Hence ADASYN replaces the random selection of minority seed examples in SMOTE with a more systematic approach focusing specifically on examples with a large proportion of majority neighbors, thus residing closer to class boundaries in feature space. This difference can be seen in Figure 6.2 by comparing the SMOTE result in (a) to the ADASYN result in (b): ADASYN focuses the creation of synthetic examples (yellow dots) in the vicinity of minority examples (turquoise dots) that are harder to learn due to their proximity to class boundaries.

Note that while being presented here for binary classification problems, ADASYN can be easily extended to the multi-class case by running it multiple times: The number G of desired synthetic examples for each minority class is determined with respect to the majority class only, while nearest neighbor search can be carried out in a minority-versus-all-other-classes setting, such that class boundaries with respect to all other classes are considered in determining the weights g_i for each minority. The union of all synthetically created minority class examples over all minority sets then balances the overall multi-class dataset.

6.3.3 Balancing in SynOpSis

In the pattern classifier of *SynOpSis* shown in Figure 6.1, class balancing is an optional module that can be enabled and disabled. If enabled, any *training* of a classifying model is carried out with respect to training data that has been balanced using ADASYN, with the goal of preventing the learning algorithm from focusing on the majority class (upper part of the figure). In contrast to that, any data the trained classifying model is *applied* to is not balanced (bottom part of the figure). Hence all measures of classification quality employed during optimization, for model selection, performance estimation and reporting results, are

[2]The same rationale on feature scale normalization as in the previous footnote concerning SMOTE applies, cf. Section 6.3.1. ADASYN uses SMOTE as a subroutine, and the value of K_s chosen for the SMOTE subroutine may be different from the value of K_a used in ADASYN.

measured with respect to the class prevalence actually observed in the detector output. This enables a representative assessment of the analysis outcome. Empirical results concerning the effect of balancing on *PAMONO* data analysis are reported in Section 6.7.2, along with the employed parameter choices.

6.4 Feature Scale Normalization

In the context of learning a feature-based classifier, feature scale normalization means putting the different dimensions of feature space, which are possibly measured on different scales, on one single scale. Normalizing feature scales makes feature value magnitudes comparable between different dimensions of the feature space $\mathbb{R}^F$, i.e. between different types of features. Feature scale normalization operates separately per dimension of $\mathbb{R}^F$ and is not to be confused with normalizing feature vectors $\mathbf{f} \in \mathbb{R}^F$ to a certain length.

Feature scale normalization is a highly recommended data preprocessing step, if a learning algorithm is to be used that is based on distance computations involving more than one feature dimension, such as Euclidean distance between feature vectors. Putting all features on the same scale before Euclidean distance computations prevents features residing on larger scales from dominating features on smaller scales. Popular examples of learning algorithms for which feature scale normalization is recommended are k-NN [HTF09] and SVM [MMR+01], cf. also [HCL03]. For learning algorithms not involving comparisons of feature magnitudes between different features, like Naïve Bayes [RN03] and Random Forest [Bre01], the feature scale normalization module in Figure 6.1 can be deactivated.

As input, feature scale normalization receives a feature matrix $\mathbf{F} \in \mathbb{R}^{E \times F}$, consisting of E examples, represented as row vectors in an F-dimensional feature space, where each column of $\mathbf{F}$ contains another type of feature. Now affine feature scale normalization computes per-feature scales $s_i \in \mathbb{R}$ and offsets $o_i \in \mathbb{R}$, as will be described in Section 6.4.1. Assuming s_i and o_i as given, an entry $f_{e,i}$ of the feature matrix $\mathbf{F}$ is affinely normalized by computing

$$\widehat{f_{e,i}} = \frac{1}{s_i}(f_{e,i} - o_i). \tag{6.22}$$

Here i is the index of the regarded feature, and e is the index of the example in the dataset that is normalized. Scale s_i and offset o_i are computed and applied individually for each dimension i of feature space, using the examples e in the training data. Assembling the $\widehat{f_{e,i}}$ for all $i \in \{1, \ldots, F\}$ into a vector yields the normalized example e in feature space. Normalizing all examples e in an input dataset this way, yields the output matrix $\widehat{\mathbf{F}} \in \mathbb{R}^{E \times F}$ with normalized feature scales.

6.4.1 Methods for Affine Feature Scale Normalization

Affine methods for feature scale normalization primarily differ in how the per-feature scales s_i and offsets o_i applied in Equation (6.22) are computed. Aksoy and Haralick give an overview to be briefly summarized here [AH01].

Unit Range Transformation

The unit range transformation maps all feature values to the interval $[0, 1]$. This is achieved by determining s_i and o_i as

$$s_i = \max_e(f_{e,i}) - \min_e(f_{e,i}), \quad o_i = \min_e(f_{e,i}). \tag{6.23}$$

Note that this transformation is solely defined by the two most extreme values in each dimension and is thus susceptible to outliers.

Zero-Mean, Unit-Variance Transformation

A zero-mean, unit-variance transformation is particularly beneficial if the distributions of the individual features are approximately Gaussian: For input features with perfectly Gaussian distributions, the normalized features follow the standard normal distribution. Scale s_i and offset o_i are determined as

$$s_i = \sigma\left(f_{\circ,i}\right), \quad o_i = \mu\left(f_{\circ,i}\right), \tag{6.24}$$

using the sample standard deviation and mean functions defined in Equations (5.17) and (5.16).

Interquantile Range Transformation

In the presence of extreme outliers, the contribution of *every* feature value to the sample standard deviation and mean can make the the zero-mean, unit-variance transformation a poor choice. An alternative that is less prone to suffer from outliers is an interquantile range transformation: It works similarly to the unit range transformation but instead of using the minimum and maximum, and thus the most extreme outliers, s_i and o_i are computed with respect to more robust rank order statistics defined by the quantiles. As an example, if quartiles are selected, s_i is defined as the interquartile range of the data, i.e. the difference between the upper and the lower quartile. The lower quartile can be defined as the smallest value that is larger than the quarter of smallest values. Analogously the upper quartile can be defined as the largest values that is smaller than the quarter of largest values. Then o_i is set to be the lower quartile. This normalization is not influenced by the most extreme half of values in both directions. Besides these affine methods, nonlinear approaches to feature scale normalization exist, e.g. rank normalization or clipping of extreme outlier values [AH01].

6.4.2 Applying Feature Scale Normalization

Scales and offsets s_i and o_i for each dimension of feature space are computed from the training data, using e.g. one of the methods from the previous section. Subsequently, Equation (6.22) is used to scale every example and feature in the training data and in the input to be classified. It can hardly be overemphasized that scales and offsets are determined *once*, from training data, and that these scales and offsets from *training* are applied to *any* input data to be classified [HCL03], e.g. the real dataset in the *Application* stage of *SynOpSis*, cf. Figure 3.2. They are not computed anew for input datasets. Applying the same normalization to all datasets ensures that a certain feature value $v \in \mathbb{R}$ observed in a normalized training feature has the same meaning as value v appearing in the same normalized feature in any other dataset:

Feature value relations across datasets are preserved as the same transformation is applied to all of them, mapping them into the same normalized feature space from which the classifying model is learned. Separate determination of s_i and o_i for each dataset would destroy this property. Note however, that e.g. in case of a zero-mean, unit-variance transformation, only the normalized training set will have zero-mean and unit-variance, while mean and variance of the normalized input set may slightly deviate.

A further aspect to be considered if feature scale normalization is combined with class balancing (cf. Section 6.3) is to balance the training data *before* normalization. Doing so enables the newly created examples to contribute to the computation of scales and offsets (oversampling), respectively prevents discarded examples from doing so (undersampling).

6.5 Feature Selection

Feature selection receives the output of the optional feature scale normalization module as input, consisting of labeled examples in feature space. The task of feature selection is to identify a subset of the dimensions of feature space that is most important in the classification task. Hence it is a technique for dimensionality reduction. Reducing the number of dimensions in the data may aim at a number of different goals. Guyon and Elisseeff in their 2003 survey on feature selection [GE03], identify three main categories of such goals:

- Feature selection can increase prediction performance, e.g. by counteracting adverse phenomena related to the curse of dimensionality.
- Time and space complexity is improved: Storing a model learned from fewer dimensions requires less space, and applying it for predictions is faster.
- New insights into the data and the underlying application problem may be generated: Finding out which features are important in the application problem to be solved may aid its understanding. Furthermore, reducing the dimensionality of feature space usually facilitates data visualization.

In the context of applying *SynOpSis* for *PAMONO* data analysis, the primary goal followed by feature selection is assessing feature importance: The already small number of 67 features in total does not necessitate feature selection with regard to efficiency, and as will be shown in Section 6.7.3, the benefit in terms of classification quality is negligible. Therefore, in the pattern classifier of *SynOpSis* (cf. Figure 6.1), feature selection is an optional module, serving in the evaluation of the features themselves, cf. Sections 6.7.3 and 6.7.4. The output of feature selection is the subset of features that are useful in classifying the input data, cf. bottom part of Figure 6.1.

6.5.1 Approaches to Feature Selection

Adopting the taxonomy from [GE03], upon which this entire section is based, feature selection approaches divide into three main categories. Note that the summarizing descriptions provided here focus on the task of conducting feature selection for a subsequent binary classification. Regression problems and the multi-class case are covered in [GE03].

Filter Approaches

The main characteristic of filter approaches to feature selection is that features are examined *separately*: For each feature separately, a figure of merit is computed, measuring the amount of information it provides concerning the class label. Examples of such figures of merit are correlation criteria like the Pearson correlation coefficient [GE03] or inter- and intra-class distances as used in Section 5.5.2. Features are sorted by decreasing merit and the top-ranking features are selected. By not considering feature interactions, filter approaches are fast and scale well. They are typically implemented as a preprocessing step to the actual learning algorithm of which they are independent. However, learning algorithms can be employed as single-feature classifiers within filter methods, generating figures of merit in terms of their performance.

Being univariate is a major drawback of filter approaches because features that yield no information concerning the class label if regarded separately, can provide considerable benefit if regarded jointly with other features, cf. Figure 3 in [GE03] for two revealing toy examples. Univariate methods like filters can not consider such relations. Furthermore, they prefer perfectly correlated and thus redundant features with larger individual merit over more complementary features with lower individual merit, even if the combined information from the complementary features exceeds that of the correlated features.

In the light of these issues, the following distinction can be made: Filter approaches select the most **relevant** features in terms of *per-feature* predictive power concerning the label. These features are not necessarily the most **useful** ones in learning a classifier from *all* selected features.

Wrapper Approaches

Wrapper approaches aim at resolving this drawback of filter approaches at the expense of increased computational complexity and decreased scalability. The key idea of wrapper approaches is wrapping a learning algorithm in the feature selection scheme and using the performance values it attains on different subsets of features as a measure of the quality of the *entire subset*. The utilized learning algorithm is regarded as a black box and it may or may not be the learning algorithm to be used in the final classification. Two very common examples of feature selection schemes, i.e. of search strategies with respect to good subsets of features, are forward selection and backward elimination. Forward selection starts with an empty set and in each iteration adds the feature to the current set that improves classification performance the most. It terminates e.g. if a certain number of features has been reached or if improvement drops below a threshold. Forward selection is a greedy strategy because decisions to select a certain feature are never revised. Backward elimination analogously eliminates features from the full set in a greedy fashion.

By considering classification performance as obtained by *subsets* of features, wrapper approaches aim at finding features that are *useful* for classification, in the sense described above. They might however miss relevant features if these features are highly correlated with features already selected because such features do not improve classification performance due to their redundancy. Overfitting the selected subset to the training data is another drawback, which particularly affects applications where the number of features to select from exceeds the number of available training examples by a large extent.

Embedded Approaches

The unifying characteristic of embedded approaches to feature selection is that the feature selection scheme is directly embedded into the training procedure of the final classifier: In each step of e.g. a forward selection scheme, the change in objective function values is *predicted* for adding or removing a feature to or from the selection. The procedures to compute such predictions highly depend on the learning algorithm under consideration, and hence embedded methods differ vastly with respect to the employed learning algorithm. The major advantage of embedded approaches is their computational efficiency because the classifying model need not be retrained for every change made to the set of selected features, like in wrapper approaches. A drawback compared to wrappers is that embedded methods can not be constructed from existing algorithms in a building block manner, but they must be specifically designed for each learning algorithm to be used.

6.5.2 Feature Selection in SynOpSis

While any of the feature selection approaches summarized in the previous section can be used to implement the feature selection module in the pattern classifier (Figure 6.1), the following choice was taken in the context of *PAMONO* data analysis: A wrapper approach with a Naïve Bayes classifier [RN03] was employed for feature selection. The Desirability Index (DI) of the same objectives as used for the pattern classifier, i.e. the geometric mean DI of Precision and Recall, was used to measure the quality of a given subset of features (cf. Section 7.3.3 for concrete desirability settings). Precision and Recall were evaluated in a five-fold cross-validation with stratified sampling. As the search strategy to optimize the DI, a single-objective Genetic Algorithm (GA) was chosen, the configuration of which is detailed as part of the evaluation in Section 6.7.3.

The choice of using this wrapped Naïve Bayes approach for feature selection was taken for the following reasons:

- First of all, for being a wrapper approach, it is multivariate. The focus of feature selection in *SynOpSis* for the $F = 67$ features used in *PAMONO* lies on identifying the most *useful* features for classification, in order to gain knowledge about which features work well in conjunction. Hence a method that can find useful subsets and not only relevant single features is desired.
- In contrast to embedded approaches, choosing a wrapper introduces no assumption about the learning algorithm used to compute the classifying model. Therefore, the pattern classifier remains modular, and learning algorithms can be easily switched.
- The selected Naïve Bayes classifier has no parameters requiring optimization to give good results.
- Besides that, a Naïve Bayes classifier is fast to compute, allowing cross-validation during feature selection at reasonable computational expense.
- Furthermore, no normalization of feature scales is assumed, so the optional normalization module in Figure 6.1 remains optional even with the feature selection module enabled.
- With regard to the *PAMONO* application scenario, the risk of overfitting is comparably low due to the low number of $F = 67$ features to select from, in comparison to the number E of examples typically observed in detecting nano-objects.

A drawback is that feature selection is unspecific to the learner to be finally employed, if that learner is not Naïve Bayes. If the optional feature selection module described in this section is enabled, the selected subset is computed with respect to the labeled training data, and the selection is applied to both, the training data used in learning the classifying model, and any data to be classified by that model.

6.6 Learning Algorithms

The learning algorithm is the only non-optional module in the training phase of the pattern classifier, cf. upper part of Figure 6.1. *SynOpSis* conducts supervised learning, i.e. for each example in the training data not only a feature vector $\mathbf{f} \in \mathbb{R}^F$ is known, but also its associated ground truth class label $t \in \{c_1, \ldots, c_C\}$. Besides this training data, that was preprocessed by a subset or all of the optional preprocessing modules presented in the previous sections, the learning module receives algorithmic parameters as inputs that configure the learner. Its output is a classifying model, represented as a function $\xi(\mathbf{f}) = p$, mapping from feature space to label space. This model assigns a predicted class label $p \in \{c_1, \ldots, c_C\}$ to a feature vector $\mathbf{f} \in \mathbb{R}^F$. The function $\xi(\mathbf{f})$ is evaluable over the entire feature space, not only in the supplied training vectors. Therefore, as an illustration, determining $\xi(\mathbf{f})$ can be regarded as a scattered data approximation from the feature space $\mathbb{R}^F$ to a discretely-valued co-domain of labels.

Supervised learning means computing the classifying model $\xi(\mathbf{f})$ from *labeled* training data supplied as input. In *SynOpSis* such data is available via synthesis (cf. Chapter 4) and matching (cf. Section 5.8). The process of evaluating the function $\xi(\mathbf{f})$ to generate a predicted class label $p \in \{c_1, \ldots, c_C\}$ will in the following also be referred to as **applying the model**. Model application is used to create predictions of class labels for unlabeled input feature vectors, cf. bottom part of Figure 6.1. In real-time data analysis, as conducted by *SynOpSis* in the *PAMONO* case, learning algorithms for which the classifying model $\xi(\mathbf{f})$ can be applied in real-time are of particular interest. Section 6.6.3 presents the Random Forest learner for which real-time capable model application has been implemented on the GPU by Libuschewski [Lib15b]. Note that the modular structure of *SynOpSis* allows any other supervised learner to be used for classification, provided that it accepts the data types of the features arising in the application case at hand.

In the context of *PAMONO*, four learning algorithms were examined in *SynOpSis*, which will be summarized in the following: Section 6.6.1 depicts the k-NN algorithm as a representative of lazy learning. Section 6.6.2 presents the SVM learner, based on convex optimization, while Section 6.6.3 recaps Random Forest, an ensemble learner made from decision trees. Finally, Section 6.6.4 describes how a Naïve Bayes classifier works. Degrees of freedom that can be used to configure each learning algorithm are discussed during the respective presentation, while those algorithmic parameters that are optimized by *SynOpSis* are listed as part of the experiment description in the results, cf. Section 6.7.1.

6.6.1 k-Nearest Neighbors Algorithm (k-NN)

The first learner to be presented here is the k-Nearest Neighbors (k-NN) algorithm [HTF09]. It is an example of lazy learning, i.e. training consists solely of storing the training data. In contrast to non-lazy learners like those presented in the subsequent sections, where the

classifying model is fitted to the training data in a process of *abstracting* from it, the classifying model of a lazy learner is simply a copy of the training data itself, leaving any process of abstraction to be conducted in applying the model. On the upside, training such a classifier is as fast as creating a copy of the training data. On the downside, depending on the type of lazy learner, applying it to classify data can be slow due to the lack of preceding abstraction.

The k-NN learner [HTF09] classifies an input example $\mathbf{f}$, represented as a vector in the feature space $\mathbb{R}^F$, by finding the k nearest neighbors of $\mathbf{f}$ in the labeled training data and assigning it to the class observed in the majority of those neighbors. Choosing an even k can result in ties in this voting scheme, which can be broken, e.g. by randomly assigning the example to one of the competing majority classes or weighting votes with proximity. The larger k is chosen, the smoother the class boundary of the classifier will be. Conversely, decreasing k increases the complexity of the learned model and thus the risk of overfitting the data. In the extreme case of $k = 1$, each training example is classified correctly by fully overfitting the training set, thus providing little abstraction. Generalization performance for such models is typically very low, cf. also Section 3.9.1, especially Figure 3.10.

Besides the choice of k, the behavior of the k-NN learner is determined by the choice of the distance measure used in computing the neighbors of an input vector. A very common choice is Euclidean distance [HTF09], which only works on numerical features and assumes normalized feature scales. Hastie, Tibshirani, and Friedman recommend using a zero-mean, unit-variance transformation for k-NN with Euclidean distance [HTF09], cf. Section 6.4. Further distance measures can be found e.g. in [WNC07] and in Chapter 14 of [HTF09], which also covers measures that are suitable for categorical and mixed types of features. Depending on the selected distance measure, further parameters may arise, and the assumption of normalized feature scales may vanish. For information on how optimization of the k-NN learner was conducted in *SynOpSis*, cf. Section 6.7.1.

6.6.2 Support Vector Machine (SVM)

The Support Vector Machine (SVM) [MMR+01] is a method for supervised classification and regression. The presentation given here will focus on two-class classification because it is the task arising in *PAMONO* data analysis. SVM regression is covered e.g. in [SS04], while an extension to multi-class classification is given in [WW98]. The emphasis of the following depiction lies not on a formal (re-)statement of how SVMs work but on the roles and influences of the parameters arising in optimizing SVMs. Detailed formal descriptions of SVMs can be found e.g. in [MMR+01; Bur98].

The key idea of SVM training can be summarized as follows: Given a labeled training set in feature space, the SVM algorithm finds a hyperplane that best separates the two classes. As the criterion defining what a 'best separation' is, the so-called maximum margin is used: The hyperplane is chosen such that the margin between the hyperplane and the respective closest vectors in both classes is maximized. These vectors that are closest to the hyperplane and thus reside *on* the margin are denoted as the **support vectors**. The further away from the hyperplane they are located, i.e. the larger the size of the margin, the better the classes are separated. As the notion of the margin involves point-to-hyperplane distances, the SVM learner assumes normalized feature scales to avoid dominance of the larger scales in these distance computations [HCL03]. The techniques from Section 6.4 can be used for normalization.

Training an SVM is equivalent to determining the support vectors within the training set, and this task can be formulated as a convex quadratic optimization: The equation to be optimized results from the objective of maximizing the margin, in combination with Lagrange multipliers that serve to transform the conditions of correct classification of the training examples into summands of the objective. A formal derivation of the optimization problem and its dual can be found in [MMR+01]. The decision boundary defined by the hyperplane is influenced *solely* by the support vectors: If any other feature vector is changed, the hyperplane is not influenced as long as that vector is not changed to become a support vector (or to swap its side with respect to the hyperplane as discussed below in the context of slack variables). Applying an SVM model to classify an input example works by determining on which side of the maximum margin hyperplane its feature vector resides. This can be decided considering solely the support vectors. As a consequence, the storage requirements of an SVM classifying model are determined by the number of supports vectors, as only these need to be stored. This number is often small compared to the size of the training dataset, giving the SVM an advantage in terms of storage complexity (and application time) over k-NN.

It is important to note that in the data arising in most practical applications, classes are rarely linearly separable in feature space. In such cases, the optimization problem to be solved for determining which of the training vectors are support vectors has no solution. Two mechanisms, described in the following, are used in SVM learners to alleviate this problem.

Slack Variables and the Regularization Parameter C

One way of making the optimization problem solvable in case of data that is not linearly separable is to introduce so-called **slack variables** into the optimization problem. These slack variables relax the constraints imposed by the margin by allowing some examples to be misclassified, making it a soft margin. Each slack variable corresponds to one training example and assumes value 0 if the corresponding example is classified correctly and lies outside the margin. Values in $]0, 1]$ are assumed if the example is classified correctly but lies inside the margin, and a value larger than 1 indicates that an example is misclassified [MMR+01]. Hence a solution can be found even for data that is not linearly separable, and that solution will contain some slack variables larger than 1. The **regularization parameter** $C \in \mathbb{R}_{>0}$ is the weight with which the slack variables contribute to the optimization problem, and therefore it controls the magnitude of misclassification penalty: The larger C, the stronger misclassified examples are penalized and the more rigid the margin becomes [MMR+01]. C trades off against the size of the margin: The lower C is chosen, the softer the margin, and consequently the wider it will be because allowing some examples to be misclassified means that the support vectors can be further away from the hyperplane. Lower choices of C decrease the risk of overfitting the training data because it decreases model complexity, as discussed in the context of model selection and illustrated in Figure 3.10.

The Kernel Trick

Another way of letting the SVM cope with data that is not linearly separable is the so-called **kernel trick**, which can be combined with the previous approach: It consists of exploiting the fact that data which is not linearly separable in the original feature space might be linearly separable if mapped into a space of higher dimension using a nonlinear kernel function. Even if the data is still not linearly separable in that higher dimension, its separability

typically improves, allowing for a smaller value of C and thus a softer and wider margin. An illustrative figure visualizing how the kernel trick works can be found in [Fle08] (Figure 4). Common choices of nonlinear kernel functions are polynomial, Radial Basis Function (RBF) and sigmoid kernels, formally defined e.g. in [Bur98].

Besides the choice of the kernel function to be used, each kernel function itself may exhibit parameters that require to be configured by the user [Bur98]. Furthermore, the misclassification penalty C of the soft margin must be chosen [HCL03]. Kernel choice, parameters, and examined ranges as used for optimizing the SVM learner in *SynOpSis* are listed in Section 6.7.1.

6.6.3 Random Forest

Random Forest is an ensemble learner first proposed by Breiman in his seminal paper [Bre01], upon which this section is based, unless stated otherwise. The idea of ensemble learning is to combine many weak learners to give one strong learner [HTF09]. In the case of Random Forest, the weak learner is a tree constructed using the Classification and Regression Tree (CART) algorithm [BFS+84], which will be explained later in the context of its usage as a subroutine of the Random Forest algorithm. For now, a tree obtained via CART can be thought of as single classifier. As its name suggests, CART and resultantly Random Forest can also be used for regression. This topic is omitted here for brevity, and the reader is referred to the literature [Bre01; LW02; HTF09].

Random Forest uses bagging (bootstrap aggregation) [Bre96] to create its ensemble of CART trees: Each tree is grown from a bootstrap sample (cf. Section 3.9.1) of the original training data, i.e. examples are selected randomly with replacement from the input. Each bootstrap sample has the same size as the input dataset, and thus an expected ratio of $(1 - 1/n)^n \approx \exp(-1) \approx 0.368$ of examples does not appear in each such sample [Koh95]. Aggregation of the individual CART results is done by majority voting over all trees. Using a randomized ensemble aims at reducing the risk of overfitting the training data, and in Section 2.1 of [Bre01], Breiman has proven that the generalization error incurred in Random Forest converges as the number of trees is increased. Hence increasing the number of trees does not increase model complexity in a way that enables the classifier to memorize the training data, as was the case in Figure 3.10.

The Random Forest learner uses the **CART** algorithm [BFS+84] as a subroutine. CART will be summarized now, following [HTF09] but already introducing Breiman's modifications with regard to its use in the Random Forest Breiman. Given one bootstrap sample as provided by the outer Random Forest ensemble algorithm, the construction of a single CART tree in a Random Forest works as follows: For initialization, a root node is created and all training data is assigned to that root node. Then, starting with the root node, the following procedure is conducted and recursively launched for all child nodes spawned in its course: A feature f_s and a threshold t_s are selected to disjointly split the data in the node into two subsets, where one subset contains all data with $f_s \leq t_s$ and the other contains all data with $f_s > t_s$. The objective driving the search for appropriate f_s and t_s is to minimize the impurity of the class labels in the resulting subsets, i.e. the search for f_s and t_s aims at producing subsets containing many examples of one class and as few as possible examples from other classes. Several measures of impurity exist, the most common of which are summarized now [HTF09]:

- The **misclassification error** is defined as the relative proportion of examples that would be misclassified in a majority vote over all examples in the node.
- **Cross-entropy** quantifies uniformity of the class labels in the node: The more uniformly class labels are distributed in a node, the higher the cross-entropy, and thus the more impure the node. Conversely, cross-entropy is zero for pure nodes.
- The **Gini index** can be interpreted in the following way: The data in the node is considered as the training data of a classifier that classifies each example as class c with a probability equal to the relative prevalence of class c examples in the node. Then the Gini index is the training error of this classifier. Like cross-entropy it grows with increasing similarity of the class label distribution to a uniform distribution.

Formalizations of these measures of impurity can be found e.g. in Equation (9.17) of [HTF09]. Figure 9.3 in the same book illustrates them for two-class classification tasks. Cross-entropy and the Gini index are most commonly used in training CART classifiers because in contrast to misclassification error they are differentiable, such that the search for the splitting feature f_s and threshold t_s can readily be conducted via numerical optimization.

Given f_s and t_s, the data in the current node is split as discussed before, and the two disjoint subsets of data are assigned to two newly created child nodes. Furthermore f_s and t_s are stored in the parent node for applying the model to unseen data. Then the procedure is launched recursively for the child nodes until a stopping criterion is fulfilled, e.g. the nodes are pure, too small for further splits, or a maximum depth D of the tree has been reached. In contrast to the original CART algorithm [BFS+84], the CART trees used in a Random Forest are not pruned after this growing procedure because aggregation is expected to sufficiently counteract the risk of overfitting incurred by unpruned trees [Bre01].

A further and more pivotal change to the original CART algorithm Breiman introduced for Random Forest is that at each node during CART construction, the splitting feature may be chosen only among a subset of K features, which is drawn i.i.d. at each node from the full set of features. This extra randomness (in addition to bootstrap sampling) is injected into CART by the Random Forest algorithm in order to decrease the correlation between the individual trees in the ensemble: Decreasing K makes feature choice more random, so the trees are less correlated, which is beneficial for the diversity of classifiers in the overall ensemble. However, predictive strength of the individual tree decreases because it can choose from fewer candidate features at each node. An ideal Random Forest aims at low correlation between trees on the one hand, and individual trees with high predictive strength on the other hand. Both of these opposing objectives are to a large extent controlled by the choice of K, realizing a trade-off between them.

Repeating this CART procedure for all bootstrap samples, a Random Forest consisting of I trees is constructed. Applying that forest to classify a new example relies on the application of the individual CART trees which works as follows: A new example is classified by using its feature values to traverse the tree. Traversal starts from the root node and ends in a leaf node by recursively tracing the path to the child node corresponding to the infinite interval in which the value observed in the splitting feature resides. Once a leaf of the tree is reached, the example is classified (by that single tree) as belonging to the majority class observed in the leaf. This is repeated for all I trees in the forest, with each tree voting for a certain class. Finally, the classification output of the forest is determined as the majority vote over its constituent trees.

Properties of the Random Forest Learner

The particular way in which the Random Forest algorithm learns a classifying, respectively regression model results in a number of advantageous properties, which will be summarized now. First of all, it requires no normalization of feature scales because comparisons of feature values only occur for equal types of features, not between different types: Each splitting operation divides one individual axis of feature space into two parts, and no other feature dimensions are involved in this split. This makes the Random Forest fast to train compared e.g. to SVM training, where all features are involved simultaneously in the optimization problem. Compared to iterative methods like boosting [FS97; JZK+07; HBR+08], the training procedure is embarrassingly parallel because is uses independent bootstrap samples and applies CART independently for each such sample. Furthermore, the Random Forest algorithm has only few parameters that impact results quality and actually need tuning [LW02]: If model size and application time do not matter, the number I of trees can simply be chosen as a large-enough constant because an I that is larger than necessary does not increase overfitting as proven in [Bre01]. The maximum tree depth D is also only of concern with regard to model size and application time; Breiman e.g. simply uses fully grown trees [Bre01]. In some practical applications, however, the number K of attributes available for splitting, was reported to have a considerable impact on results quality [HTF09]. In *SynOpSis*, all three parameters I, D and K are tuned in the ranges and for the reasons to be discussed in Section 6.7.1. A recommended starting value for K is $\lfloor\sqrt{F}\rfloor$, where F is the number of features in the dataset [Bre01; HTF09]. With parameters fixed and the classifier learned, the Random Forest classifier can be applied in real-time on the GPU [Lib15b].

6.6.4 Naïve Bayes

Naïve Bayes [RN03; Ris01] is a learning algorithm that classifies an unlabeled feature vector $\mathbf{f} \in \mathbb{R}^F$ as belonging to the class c_k exhibiting the highest conditional probability of *being* observed, given that feature vector $\mathbf{f}$ *has been* observed. Stated formally, Naïve Bayes classifies $\mathbf{f}$ into the class c_k that maximizes the posterior probability $P(c_k \mid \mathbf{f})$. This conditional probability is called 'posterior' because it depends on an observation of data, in this case on the feature vector $\mathbf{f}$. Classifying $\mathbf{f}$ as belonging to the class with maximum posterior probability is denoted a Maximum a Posteriori (MAP) decision rule [RN03].

Training a Naïve Bayes classifier means estimating the posterior distribution $P(c_k \mid \mathbf{f})$ for each class c_k on the basis of labeled training data. Applying it for classification means evaluating these distributions for the feature vector to be classified and outputting the class c_k associated with the distribution exhibiting maximum probability. The method for learning a single distribution $P(c_k \mid \mathbf{f})$ from the training data can be derived as follows: Firstly, Bayes' theorem [RN03] is applied:

$$P(c_k \mid \mathbf{f}) = \frac{P(c_k)P(\mathbf{f} \mid c_k)}{P(\mathbf{f})}. \tag{6.25}$$

Doing so, restates the posterior distribution in terms of a prior probability $P(c_k)$ of observing class c_k not conditioned on any observation, multiplied with the likelihood of observing feature vector $\mathbf{f}$ given that the class label actually is c_k, and normalized by dividing by the overall probability of observing $\mathbf{f}$. As this denominator does not depend on the class label and is constant for given $\mathbf{f}$, it is not of interest in computing the Naïve Bayes classifier.

Equation (6.25) can be further simplified by introducing the eponymous naïve assumption of the Naïve Bayes classifier: It is assumed that given the class label, each feature dimension f_i is conditionally independent of every other feature dimension f_j in the vector $\mathbf{f}$. Naïve Bayes classifiers are known to work well, even though this assumption rarely actually holds in practice [RN03]. Applying the independence assumption to Equation (6.25) decomposes the class-conditional probability $P(\mathbf{f} \mid c_k)$ of observing feature *vector* $\mathbf{f}$ into a product of the class-conditional probabilities of its *scalar* components:

$$P(c_k \mid \mathbf{f}) \propto P(c_k) \prod_{i=1}^{F} P(f_i \mid c_k). \tag{6.26}$$

Note that the constant denominator $P(\mathbf{f})$ from Equation (6.25) has been omitted, hence the proportionality instead of equality. Compared to estimating $P(c_k \mid \mathbf{f})$ directly, the factors in Equation (6.26) can be more easily estimated from the training data: The class prior probabilities can be estimated as the class ratios in the training data, and the class-conditional distributions of scalar feature values can be modeled by fitting e.g. Gaussian distributions to the per-class feature values observed in the training data.

Naïve Bayes in *SynOpSis*

In the pattern classifier of *SynOpSis*, the Naïve Bayes learner is used as a baseline method for comparison with the three previously discussed learners. This is due to its simplicity and the fact that it exhibits no algorithmic parameters if the employed model for the class-conditional feature distributions is assumed fixed. In analyzing *PAMONO* data with *SynOpSis*, Gaussian feature distributions are assumed, hence the Naïve Bayes learner used in this context requires no optimization. This is useful in ranking the classification quality achieved by the other learners, given a certain parameterization, and it is furthermore exploited in making the feature selection scheme described in Section 6.5 independent of classifier parameters. Furthermore, Naïve Bayes requires no normalization of feature scales because only probabilities are regarded in the MAP decision rule based on Equation (6.26).

6.7 Results

In the following sections, the pattern classifier from Figure 6.1 is evaluated on *PAMONO* data, and three design decisions are finalized with regard to the obtained results. The first decision is the choice of the learning algorithm to be used (Section 6.7.1), followed by decisions on whether to enable or disable the class balancing module (Section 6.7.2) and the feature selection module respectively (Section 6.7.3). Justifying these design decisions is the primary task of the conducted evaluations because these decisions remain fixed in the overall evaluation conducted in Chapter 7. Besides taking these decisions in Sections 6.7.1 to 6.7.3, Section 6.7.4 provides insight into the usefulness and distribution of some of the features presented in Section 6.2.

The order of presentation in this results section is slightly different from that in Figure 6.1, along which the methods within this chapter were presented. This reordering serves to facilitate taking the design decisions: Choosing a learner first means that class balancing and feature selection need only be evaluated for one learner instead of four, thus a combinatorial explosion of the number of results to report is avoided, and results need not be aggregated

over learners which would obscure important information. Note that Sections 6.7.1 to 6.7.4 each cover one module from Figure 6.1. The feature scale normalization module is omitted in this section structure because it was fixed as a zero-mean, unit-variance transformation for those learners that require it. An argument for this choice will be provided in Section 6.7.4. While Sections 6.7.1 to 6.7.3 cover modules from the training phase of the pattern classifier, Section 6.7.4 is concerned with the feature extraction, conducted before training.

Employed Input Data

As inputs for the conducted evaluations, only pattern detector results from sequential optimization were used, i.e. the detector was optimized independently, without taking subsequent classifier results into account. This is important in choosing the learning algorithm in Section 6.7.1: Optimizing the detector independently means that (unlike in global optimization) no channel exists over which a subsequent learning algorithm used in the classifier can feed information back into the optimization of detector parameters. Otherwise the comparison would not be fair because the learning algorithm could influence detector parameters in a way making detector results particularly well-suited to be classified with that certain learning algorithm. Furthermore, only detector results were examined for which detector Precision was below 0.98 on the synthetic training data. That means there are at least two percent of FP detector responses in the training data, which are to be eliminated by the classifier. The number of such detector results was 21, resulting from different *PAMONO* experiments (cf. Table 7.1), desirability modes (cf. Section 7.3.3) and cross-validation folds (cf. Section 7.3.4).

Optimization of Learner Parameters

For a given setup of the pattern classifier in Figure 6.1, e.g. for enabled balancing, disabled feature selection and the choice of a Random Forest learner, the parameters of that learner are optimized via *SynOpSis*. This is repeated for all setups in question, e.g. for contrasting with a setup where balancing is disabled, which allows evaluating the effect of the balancing module. The obtained results are aggregated over all 21 examined detector results, each undergoing a separate optimization. Hence parameter optimization via *SynOpSis* is conducted separately for each setup and detector result, allowing to assess how changes in the setup affect classifier performance, as will be done in Sections 6.7.1 to 6.7.3.

Optimizing the pattern classifier via *SynOpSis* works as follows in this evaluation: Algorithmic parameters of the learner are optimized with respect to the two objectives Precision and Recall. In order to fully explore the objective space and not narrow the search region, no Desirability Functions (DFs) are applied during optimization. The settings configuring the Genetic Algorithm (GA) are the same as used in the overall sequential classifier optimization detailed in Section 7.3.2. The best-performing individual from the front is determined as the maximizer of the geometric mean Desirability Index (DI) of Precision and Recall, cf. Section 7.3.3 for the employed desirability functions. Optimization is carried out on the training set, the best-performing individual is selected with respect to the DI on the validation set, and performance is estimated/evaluated with respect to the test set, cf. Terminology 3.3. Furthermore, the real sensor data to be classified was manually annotated, and classifier performance attained on this data is reported as well.

Classifier Training and Evaluation of Results

Note that the synthetic data (training, validation and test set) is classified using a model learned from the training data only, while the real sensor data is classified using a model learned from the concatenation of all synthetic data. In both cases, the same algorithmic parameters for the learner are used, as provided by the optimization.

All results are reported in terms of box plots [MTL78] of Precision and Recall, as attained over the 21 examined detector results. Separate box plots are provided for the training, validation and test sets, and for the real sensor data to be classified.

Measurement of Runtime

All timing measurements reported in this section were taken on the following system:

System Specification 6.1. Intel® Xeon® E5-2650 v3 at 2.3 GHz. Cache size: 25 600 kB. Operating system: Virtualized Microsoft® Windows® 7, running in `libvirt` on the Kernel-Based Virtual Machine (KVM) of a Linux host operating system. The Virtual Machine (VM) used four cores and 16 GB of the 2133 MHz DDR4 RAM.

The reported times are mean values over the per-individual times observed during the conducted optimizations. Since a single evaluation of objectives is realized as an individual executable call, these times include the startup of the MATLAB compiler runtime, parsing the input Comma-Separated Values (CSV) files and starting the learning environment. These overhead operations introduce a constant offset to the measured runtimes.

Section Overview

With these prerequisites stated, Section 6.7.1 finalizes the choice of the learning algorithm. Sections 6.7.2 and 6.7.3 examine whether it is advantageous to enable or disable class balancing and feature selection, respectively. Finally, Section 6.7.4 investigates usefulness and distributions of a subset of the features employed for classification.

6.7.1 Learning Algorithms

In order to choose one of the learning algorithms presented in Section 6.6 to be used in applying *SynOpSis* to *PAMONO* data, a comparative evaluation was conducted. This evaluation compares classification quality in terms of Precision and Recall, as attained by the four learners over the 21 examined detector results. *SynOpSis* is used to optimize the algorithmic parameters of each learner for each detector result to be analyzed.

Parameters Optimized via SynOpSis

The parameters tuned by *SynOpSis* will be listed in the following. To limit the search space, some parameters were kept constant, as indicated by the 'Constant' item for each learner. The configuration of the GA is the same as used in sequential optimization during the overall evaluation, cf. Section 7.3.2.

k-Nearest Neighbors Algorithm (k-NN)

Constant
: Euclidean distance is used to measure the distance between points in feature space. As recommended for this choice in [HTF09], a zero-mean, unit variance transformation (cf. Section 6.4.1) is applied to normalize feature scales.

$k \in \{1, \ldots, 50\}$
: The number k of examined nearest neighbors is optimized.

Support Vector Machine (SVM)

Constant
: An RBF kernel was selected as the constant type of the kernel function. Its parameter γ, however, is optimized, cf. below. Choosing an RBF-type kernel is recommended as a general choice, especially if the number of features is smaller than the number of examples [HCL03], as is the case in *PAMONO*. RBF kernels are furthermore general in the sense that they encompass the linear kernel as a special case [KL03] and can mimic some instances of the sigmoid kernel [LL03]. Apart from kernel choice, a zero-mean, unit variance transformation (cf. Section 6.4.1) is selected to normalize feature scales.

$C \in [0.01, 100]$
: The range examined for optimizing the misclassification penalty C takes [HCL03] as a guideline but decreases the maximum C to obtain a smaller hypothesis space. This results in a lower tendency of the SVM to memorize the training data and yields classifying models that are faster to predict because lower values of C result in fewer support vectors. A grid search for C would typically examine exponentially increasing values, and it is recommended to make the randomized components of the GA examine the search space analogously [HCL03].

$\gamma \in \left[10^{-5}, 10\right]$
: γ is a parameter of the RBF kernel, formally defined e.g. in [Bur98]. For increasing γ, the classification boundary becomes smoother [CS00], and the chance of misclassifying training examples increases, along with a decreasing risk of overfitting the training data. The effect is comparable to decreasing C, but is achieved by increasing kernel size, not by changing misclassification penalty. Decreasing γ too far enables the trained SVM to fully memorize the input data, similar to k-NN with $k = 1$ [Bur98]. The classification quality achieved by an SVM with RBF kernel critically depends on a good *combination* of both, C and γ, which is why both are subject to optimization in *SynOpSis*. The range examined for γ was chosen in accordance with [HCL03], and the same rationale as discussed with C applies, for choosing a GA search strategy mimicking exponential grid search.

Random Forest

Constant
: Cross-entropy is selected as the employed split criterion.

$K \in \{1, \ldots, 15\}$
: The number K of features randomly drawn as splitting candidates in each node is optimized. Note though, that Random Forest is claimed to be quite insensitive to its

parameters [LW02]. They are tuned nonetheless by *SynOpSis* because in some practical problems they have a larger impact on prediction quality, as reported in [HTF09] with respect to K. The importance of K is due to its control over the amount of randomness *in* and decorrelation *between* the individual trees in the forest. Choosing 15 as the maximum value for K allows to nearly double the recommended value [Bre01] of $\lfloor\sqrt{F}\rfloor = \lfloor\sqrt{67}\rfloor = 8$, where $F = 67$ is the number of features in *PAMONO* data.

$I \in \{1, \ldots, 500\}$

The number I of trees in the forest just needs to be chosen large enough because it does not increase the tendency of overfitting as proven in Section 2.1 of [Bre01]. However, smaller I are beneficial for model application speed.

$D \in \{0, \ldots, 20\}$

Limiting tree depth D is not essential: The original work by Breiman uses fully grown trees [Bre01]. Hastie, Tibshirani, and Friedman state that tuning D can be motivated mainly by decreased application time and space requirements to store the classifying model [HTF09]. They recommend using fully grown trees if fewer tuning parameters are desired. As this is not the case in *PAMONO*, D is tuned as well. Note that $D = 0$ results in fully grown trees.

Naïve Bayes

Constant

Gaussians were chosen to model the class-conditional distributions of feature values in the training data. No parameters were optimized for the Naïve Bayes learner because it serves as a baseline comparison.

Impact of Learner Choice on Classification Quality

Figure 6.3 shows box plots of classifier Precision and Recall as attained by the different classifiers on training (a), validation (b), test (c) and the real sensor data (d), cf. Terminology 3.3. The plotted populations each consist of the 21 detector results described at the beginning of Section 6.7. Training, validation and test set performances were measured by applying a classifying model learned from the training set only, while the real sensor dataset was classified by a model learned from all three synthetic datasets. This also holds for all further experiments conducted in this section. On the training set in (a), differences between learners are negligible: All medians are 1, except for Naïve Bayes Recall with a value of 0.95968 and a large spread as indicated by the Interquartile Range (IQR). Training set performances include optimism because the data used to train the model is at the same time used for measurement. Learners allowing for more model complexity than Naïve Bayes can better (over)fit the training data in (a). Performance on the training data is only reported for completeness and to give an impression of the magnitude by which performance degrades as one progresses towards more difficult datasets. Instead, performance should be measured on an unseen test set, as done in (c). Note that on the validation set in (b) used for model selection in terms of classifier parameters (cf. Section 3.9.2), performances are similar to test set performance in (c), thus only the latter is discussed here. On the test set, Naïve Bayes median Recall decreases by 0.00432, which is a small degradation compared to the other learners, allowing for more complex models. Median (Precision, Recall) tuples on the test set are: k-NN: $(0.98566, 0.96154)$, SVM: $(0.93069, 0.99308)$, Random Forest: $(0.99331, 0.95597)$,

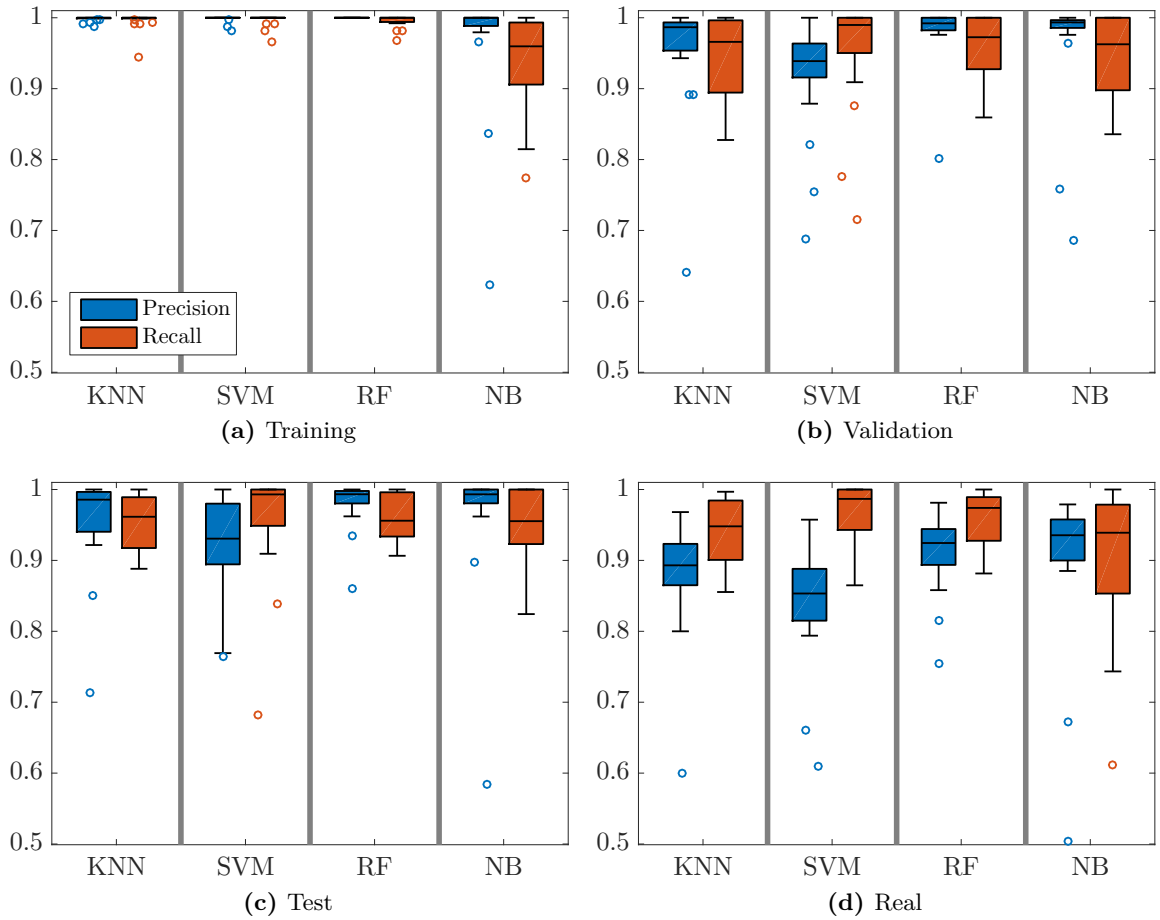


Figure 6.3: Choice of Learning Algorithm. Classifier performance as attained by the different learning algorithms is shown in terms of Precision and Recall. Synthetic datasets, i.e. training (a), validation (b) and test set (c), were classified using a model obtained from the training data, while the model classifying the real data was learned from the conjunction of all synthetic data. Parameters of the learners (if given) were optimized using *SynOpSis*. Input datasets were balanced before learning. The Random Forest learner performed best in terms of an unweighted mean of Precision and Recall on the test and real dataset. This result could be repeated on unbalanced learner inputs. More details are given in the text.

Naïve Bayes: $(0.99320, 0.95536)$. Determining "the best" learner from this information is a multi-objective decision due to its two objective functions, which could be augmented by simultaneously considering the spread over the 21 detector results as measured e.g. in terms of IQR. Regarding solely median Precision and Recall and computing their unweighted mean, i.e. assigning equal importance to both, the following ranking of learners is attained on the test set: 1. Random Forest (0.97464), 2. Naïve Bayes (0.97428), 3. k-NN (0.97360), 4. SVM (0.96189). Since manually created ground truth for the real sensor data was available, the decision for a certain learner can also be taken with respect to that data, shown in (d). Here the median (Precision, Recall) tuples are: k-NN: $(0.89305, 0.94805)$, SVM: $(0.85333, 0.98675)$, Random Forest: $(0.92453, 0.97403)$, Naïve Bayes: $(0.93548, 0.93902)$. Ranking by unweighted mean results in the same order as on the test set: 1. Random Forest: (0.94928), 2. Naïve

Bayes (0.93725), 3. k-NN (0.92055), 4. SVM (0.92004). On real data, the vote for the Random Forest learner is clearer. On this basis, the following Design Decision 6.1 is taken.

Design Decision 6.1. *The unweighted mean of Precision and Recall on real data is employed as the decisive criterion in choosing a learning algorithm for the pattern classifier in Figure 6.1. As the Random Forest learner performed best in this criterion, it is selected as the learning algorithm to be used in the pattern classifier. All subsequent investigations in Chapters 6 and 7 are restricted to this learner, in order to avoid a combinatorial explosion of results to report.*

Note that in obtaining the results stated above, the optional feature selection module in Figure 6.1 was disabled and all features were used because the reason to do feature selection on $F = 67$ features is not classification performance, but to gain insight into which kinds of features perform well, cf. Section 6.7.3. Normalization was carried out as discussed above, in the context of the parameters to be optimized. Class balancing was enabled for the reasons to be given in Section 6.7.2. In order to sort out a chicken-and-egg problem between choosing Random Forest as the learner and choosing to do class balancing with respect to the Random Forest learner results, all learners were also optimized with class balancing disabled. This guards against the potential case of one of the other learners performing better that Random Forest if balancing is disabled. Median (Precision, Recall) tuples on the real sensor dataset in this case are: k-NN: $(0.89933, 0.96403)$, SVM: $(0.88889, 0.98071)$, Random Forest: $(0.91391, 0.98214)$, Naïve Bayes: $(0.92754, 0.93902)$. The resulting ranking is: 1. Random Forest (0.94803), 2. SVM (0.93480), 3. Naïve Bayes (0.93328), 4. k-NN (0.93168). The SVM and k-NN learner benefit from disabling balancing, while Random Forest and Naïve Bayes benefit from keeping it enabled. Regardless of balancing, Random Forest performs best, thus the chicken-and-egg problem described above is resolved. The effects of class balancing are discussed in more detail in Section 6.7.2.

Impact of Learner Choice on Runtime

The runtimes of executing the learning algorithms were tracked over the parameter optimization via *SynOpSis*. Times are reported in seconds and are mean values over all individuals in the optimizations of all 21 examined detector results. They were measured on the computer described in System Specification 6.1, including the overhead operations listed after that specification. For being lazy learning, k-NN training simply means storing the input dataset, as discussed in Section 6.6.1. This makes k-NN an ideal candidate for estimating the offset introduced by the constant-time overhead operations that are executed prior to the actual learning process. Average runtime of the k-NN learner was 26.932 s. SVM training took 27.068 s, which is an increase by approximately 0.50500%. The Random Forest took an average training time of 29.620 s and hence 9.9807% longer than k-NN. Training the Naïve Bayes classifier took 27.068 s, averaging over all detector results, putting it on par with the SVM.

Note that while these per-individual runtimes are very similar, the cumulated runtimes over an optimization may differ: For the Naïve Bayes classifier, no parameters are optimized, hence in sequential optimization it needs to be learned only once for learning the classifying model. In k-NN only one integer parameter is optimized, such that exhaustive search over its range of size 50 can be used. SVM and Random Forest have more parameters, justifying

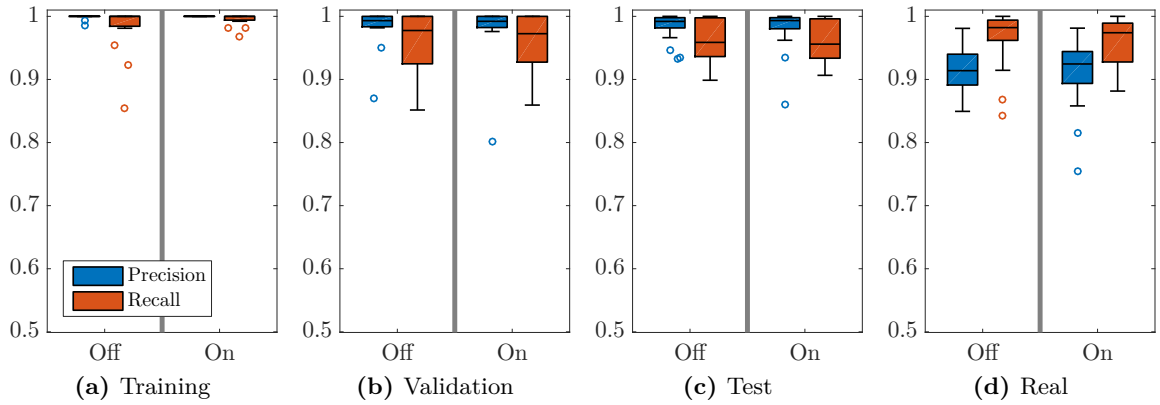


Figure 6.4: Class Balancing off versus on. Precision and Recall as attained by optimized Random Forest classifiers over 21 detector results are plotted. Each of the datasets in (a)–(d) was classified by models learned from unbalanced inputs ("Off") and from balanced inputs ("On"). The datasets themselves were not balanced, i.e. the evaluation occurs with respect to their original distribution. For the test (c) and real (d) datasets, balancing increases Precision at the cost of Recall.

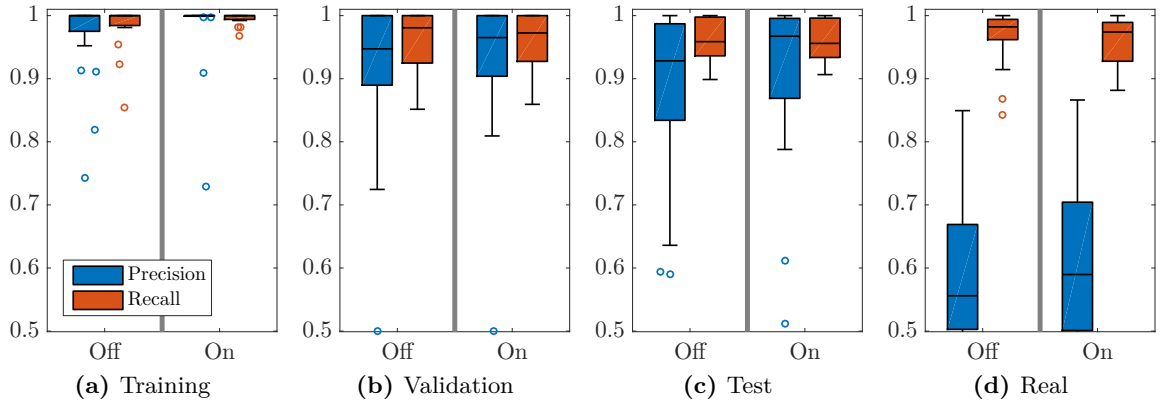


Figure 6.5: Class Balancing off versus on – Balanced Application Sets. Results for the same experiment as in Figure 6.4 are shown, except for one modification: Here the datasets in (a)–(d) were artificially balanced before measuring the attained Precision and Recall. In conjunction with Figure 6.4, this experiment investigates how the performance of classifying models learned from unbalanced and balanced inputs is affected by changes in class prevalence in the data to be classified. A detailed analysis is given in the text.

their evolutionary optimization which requires executing the learning algorithm more often as described in the GA settings in Section 7.3.2.

6.7.2 Balancing Class Prevalence

For the choice of the Random Forest learner (Design Decision 6.1), this investigation provides more details on the effect of enabling and disabling the module for balancing class prevalence, as presented in Section 6.3. In both cases, Random Forest parameters are optimized separately via *SynOpSis* as stated in Section 6.7, and results over the same 21 detector results are reported. ADASYN parameters (cf. Section 6.3.2) were constantly set to the following values: The balance parameter b was set to 1, meaning that classes will be fully balanced. The

number K_s of nearest neighbors from which the SMOTE subroutine randomly draws an interpolation target was set to 5. The number K_a of nearest neighbors examined by ADASYN for determining whether a minority example is close to a class boundary was also set to 5. Standardized Euclidean distance is used in both k-NN searches, thus implicitly handling feature scale normalization during determination of neighborhoods.

Impact of Balancing Choice on Classification Quality

Figure 6.4 shows box plots of Precision and Recall as attained on training (a), validation (b), test (c) and real sensor data (d). Within each plot, the results for turning the balancing module off are contrasted with those for turning it on. Differences are negligible. On the test set, enabling balancing increases median Precision from 0.99200 to 0.99331 while median Recall decreases from 0.95873 to 0.95597. Similarly, on the real sensor data, enabling balancing increases median Precision from 0.91391 to 0.92453 while median Recall decreases from 0.98214 to 0.97403.

While like in Section 6.7.1, a decision for or against balancing could be taken by computing an aggregate statistic of Precision and Recall, conducting a modified experiment provides more insight into the effect of class balancing. The modified experiment can be derived as follows: Using unbalanced input data emphasizes the majority class stronger than the minority class in learning the classifying model [HG09]. This implicitly introduces the assumption that class prevalence in the actual data to be classified will be similar or exhibit even larger dominance of the majority class. If this assumption does not hold, i.e. if minority class prevalence increases, compared to the data used in learning, a classifier learned from unbalanced inputs is expected to perform worse. This effect is illustrated by modifying the balancing experiment as follows: Like in Figure 6.4, Random Forest classifying models learned from unbalanced and balanced inputs are compared, but this time the respective datasets on which they are applied for measuring the attained Precision and Recall are synthetically balanced[3] using ADASYN.

Figure 6.5 displays the corresponding results obtained for applying the Random Forest classifiers to datasets with synthetically balanced class prevalence. The differences between classifiers trained with balancing disabled, compared to those with balancing enabled have increased in comparison to Figure 6.4. On the test set, enabling balancing increases median Precision from 0.92835 to 0.96743 while median Recall decreases from 0.95873 to 0.95597. On the real sensor data, enabling balancing increases median Precision from 0.55597 to 0.58964 while median Recall decreases from 0.98214 to 0.97403. In both cases, the gain in Precision is considerably larger than the loss in Recall. While this result provides the argument for taking Design Decision 6.2, its discussion continues after that.

Design Decision 6.2. *For* PAMONO *data analysis with* SynOpSis*, the class balancing module is enabled in all further investigations because this alleviates the assumption that the learned classifying model is applied solely to data with class prevalence similar to that observed in the data it was learned from. Learning from balanced inputs allows for larger differences of spatiotemporal nano-object density between the synthetic datasets used for learning and the real input to be classified, for which this density is unknown. Hence, with class balancing enabled, the* Synthesis *stage need not be informed about the expected nano-object prevalence*

[3] This creates synthetic examples even in the real data.

in the real data in order to generate corresponding synthetic data. Furthermore, the classifier learned from balanced inputs is expected to better handle changes in nano-object concentration over time: Such changes result in spatiotemporal density of target patterns increasing or decreasing over time. The density of non-target patterns can not be expected to vary in the same way over time. Hence, changes in nano-object concentration over time change class balance over time, which is better handled if the classifying model is learned from balanced input, as demonstrated by the results above. As a summary, by being learned from balanced inputs, the resulting classifying model does not implicitly prefer one class over another.

Now, to continue the discussion of Figure 6.5, the observed results can provide more insight into the effect of class balancing. Concerning this, it is important to note that only four of the 21 detector results examined in this investigation exhibit a detector Precision < 0.5, i.e. with the target class being the minority class. In all other 17 detector results, examples from the non-target class are in the minority, i.e. class balancing synthetically adds patterns from the non-target class. The following argument focuses on this more frequent case: More non-target (negative) patterns introduce the possibility of incurring more False Positives (FPs), in case they are incorrectly classified as target patterns. As a consequence, classifier Precision is the measure that potentially suffers when a classifier is applied to balanced instead of the original input data. This can be seen from comparing Figure 6.4 to Figure 6.5: While Recall remains nearly unchanged, Precision decreases considerably. Now the next question is: In how far does class balancing alleviate this drop in Precision? Figure 6.5 provides the corresponding evaluation: On the validation, test and real sensor dataset, Precision benefits from conducting class balancing prior to learning the classifier. This benefit is larger than in Figure 6.4, while the loss in Recall is comparable between both experiments. Nevertheless, for artificially balanced real data, Precision of the Random Forest learned from balanced inputs is still low. Note that ADASYN was used to artificially balance the data to be classified. The conjecture is that the low Precision attained on such data is related to the way ADASYN selects the input examples from which to create new synthetic examples: It particularly identifies borderline examples and creates new examples from those. Such examples are characterized by high proximity to the class boundaries and are thus the hardest to classify. As they belong to the non-target class for 17 out of 21 experiments, a severe degradation in median Precision occurs. This creation of worst-case examples can only incompletely be compensated for by learning the classifier from balanced inputs.

Impact of Balancing Choice on Runtime

Runtimes, as measured in seconds on the computer described in System Specification 6.1, were as follows: Without balancing, learning the Random Forest classifier took 25.841 s on average. Enabling balancing increased this runtime to the average of 29.620 s reported in Section 6.7.1. The relative increase is thus 14.624%. Note that this extra runtime is mostly due to a larger training set undergoing cross-validation in *SynOpSis* optimization. ADASYN itself is faster than the time difference above. The average time taken for ADASYN alone is 0.532 74 s. The smaller the size of the minority class in comparison to the majority class, the larger the impact of class balancing on the runtime of the learning algorithm will be.

Table 6.1: Feature Usefulness. The frequencies with which features were selected over the 21 examined detector results were counted and tabulated as measures of feature usefulness.

Features	Selection Frequency
$f_{3.5}^{\text{spot}}$	20
f^{area}, $f_{0.7}^{\text{spot}}$, $f_{\text{time}}^{\text{std}}$	16
$f_{4.9}^{\text{blob}}$, $f_{5.6}^{\text{spot}}$, $f_{\text{time}}^{\text{L1AC}}$	15
f^{circul}, $f_{4.9}^{\text{det}}$, $f_{1.4}^{\text{spot}}$, $f_{3.5}^{\text{trace}}$, $f_{\text{time}}^{\text{CV}}$, $f_{\text{time}}^{\text{PPA}}$	14
$f_{3.5}^{\text{blob}}$, $f_{6.3}^{\text{blob}}$, $f_{1.4}^{\text{trace}}$, $f_{\text{time}}^{\text{RMS}}$, $f_{\text{time}}^{\text{skew}}$	13
$f_{\text{AABB}}^{\text{width}}$, $f_{\text{OBB}}^{\text{axis1}}$, $f_{2.8}^{\text{spot}}$, $f_{7.0}^{\text{trace}}$, $f_{\text{ori}}^{\text{axis1}}$, $f_{\text{time}}^{\text{step}}$	12
$f_{1.4}^{\text{blob}}$, $f_{2.1}^{\text{det}}$, $f_{2.8}^{\text{det}}$, $f_{4.2}^{\text{spot}}$, $f_{4.9}^{\text{spot}}$, f^{rect}	11
f^{perim}, $f_{2.1}^{\text{spot}}$, $f_{7.0}^{\text{spot}}$, $f_{2.8}^{\text{trace}}$, $f_{5.6}^{\text{trace}}$, $f_{6.3}^{\text{trace}}$, $f_{\text{space}}^{\text{std}}$, $f_{\text{time}}^{\text{mean}}$	10
$f_{\text{OBB}}^{\text{aspect}}$, $f_{\text{space}}^{\text{mean}}$, f^{C_4}, $f_{0.7}^{\text{det}}$, $f_{3.5}^{\text{det}}$, $f_{4.2}^{\text{det}}$, $f_{0.7}^{\text{trace}}$, $f_{2.1}^{\text{trace}}$, $f_{4.2}^{\text{trace}}$	9
$f_{\text{OBB}}^{\text{axis2}}$, f^{C_2}, $f_{5.6}^{\text{blob}}$, $f_{5.6}^{\text{det}}$, $f_{7.0}^{\text{det}}$, $f_{4.9}^{\text{trace}}$	8
$f_{\text{AABB}}^{\text{height}}$, f^{compact}, f^{elong}, $f_{2.8}^{\text{blob}}$, $f_{1.4}^{\text{det}}$, $f_{6.3}^{\text{det}}$, $f_{\text{time}}^{\text{cross}}$	7
$f_{2.1}^{\text{blob}}$, $f_{7.0}^{\text{blob}}$	6
$f_{0.7}^{\text{blob}}$, $f_{\text{OBB}}^{\text{area}}$, $f_{\text{time}}^{\text{kurt}}$	5
$f_{6.3}^{\text{spot}}$, $f_{\text{space}}^{\text{max}}$	4
$f_{4.2}^{\text{blob}}$	3

6.7.3 Feature Selection

The feature selection module in Figure 6.1 was realized as described in Section 6.5.2. The GA was run with a population size of 20 individuals undergoing 20 generations of evolution. Feature selections were encoded as binary vectors of length F, where $F = 67$ is the total number of features to select from. On initialization, each feature is selected with probability 0.5. Elitism was enabled and tournament selection [Luk13] with tournament size 5 was applied. A one-point crossover between the selected individuals was carried out. The probability of mutating a gene was set to $1/F$. No criterion for early termination was used, so the full set of 400 individuals was always evaluated.

Feature Usefulness as Computed by Feature Selection

Table 6.1 presents the results of this feature selection strategy by listing feature names in the order of decreasing selection frequency. Note that the top-ranked feature $f_{3.5}^{\text{spot}}$ was selected in 20 of the 21 examined detector results and that the second rank consists of three features that were chosen in 16 cases, which sets the usefulness of $f_{3.5}^{\text{spot}}$ apart from the rest. This does however not imply that the f_{σ}^{spot} features are all useful: They can also be found in the middle and lower ranks. The same holds for all features of spatial intensities. Features of spatiotemporal intensities are primarily found in the upper ranks, which is presumably related to their integration of information from all three dimensions of the input data. Features of polygon shape are represented over all ranks. A more detailed and more visual discussion of feature usefulness follows in Section 6.7.4.

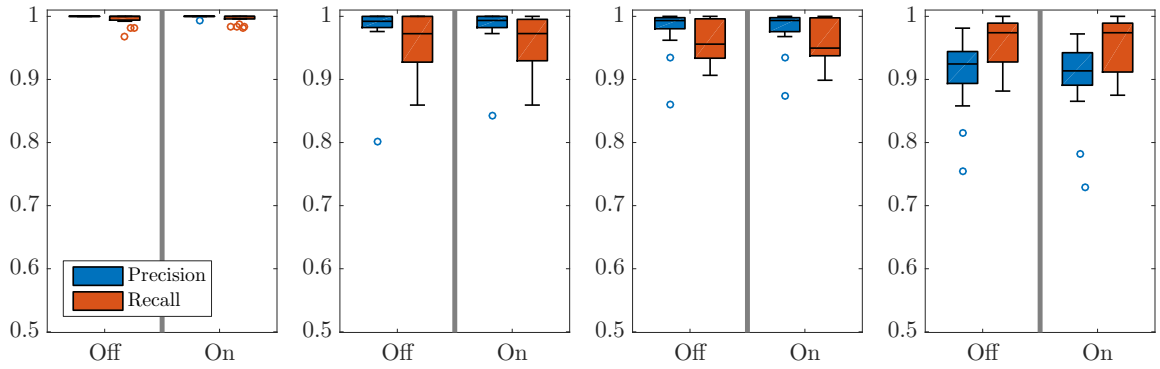


Figure 6.6: Feature Selection off versus on. The impact of the wrapped Naïve Bayes feature selection described in Section 6.5.2 on Precision and Recall over the 21 detector results is shown. A Random Forest learner on balanced inputs was used. Both, Precision and Recall are nearly unaffected by feature selection.

Impact of Feature Selection Choice on Classification Quality

Figure 6.6 plots the impact of disabling respectively enabling the feature selection module in Figure 6.1 on the attained Precision and Recall. This impact is generally small in terms of median objective values, and on the test and real dataset there is no clear vote: Enabling feature selection on the test dataset increases median Precision from 0.99331 to 0.99346, while median Recall decreases from 0.95597 to 0.94969. On the real sensor input, enabling feature selection decreases median Precision from 0.92453 to 0.91358, while Recall remains unchanged at 0.97403. Given the low number of $F = 67$ features in total, feature selection was carried out primarily to obtain the feature ranking in Table 6.1 and not to improve classifier performance[4]. Hence, Design Decision 6.3 is taken as follows:

Design Decision 6.3. *For* PAMONO *data analysis with* SynOpSis*, the feature selection module is disabled in all further investigations because its purpose was primarily to obtain an assessment of feature usefulness as summarized in Table 6.1. As expected due to the low number of overall features, the impact of feature selection on classification quality is low.*

Impact of Feature Selection Choice on Runtime

The feature selection scheme realized in *SynOpSis* is independent of the learning algorithm and parameters used to compute the classifying model. Hence it needs to be executed only once, on the training dataset. The average net runtime of this feature selection over all 21 examined detector results was 30.677 s, excluding the overhead operations listed below System Specification 6.1.

6.7.4 Feature Extraction

Given the votes of feature selection as summarized in Table 6.1, this section provides a visual insight into the most and least useful features as determined by that feature selection. Each visualization consists of class-separated histograms of feature values as observed in training

[4]This was discussed in more detail at the beginning of Section 6.5.

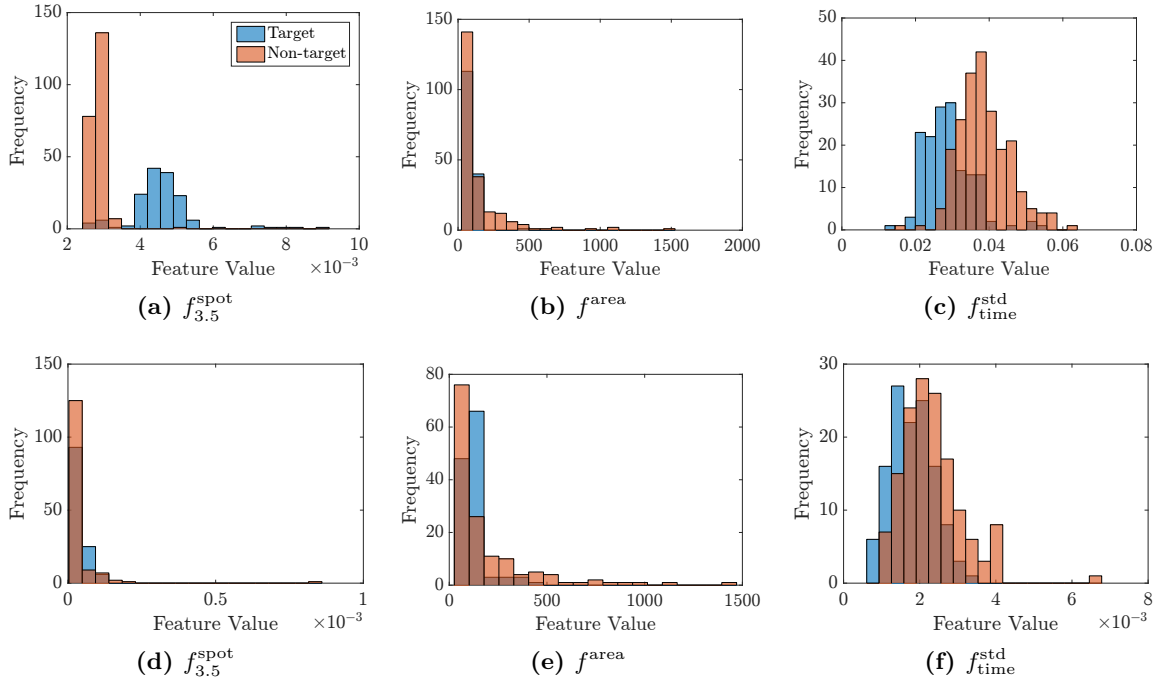


Figure 6.7: Feature Histograms – Top Ranks. Class-separated histograms of feature values are shown for the three top-ranked features in Table 6.1. The first row (a)–(c) shows these histograms for a detector result exhibiting good separation on the single feature level, while the second row (d)–(f) does the same for a detector result with bad separation. Conclusions are given in the text.

data before balancing was applied. Among the 21 detector results examined in Table 6.1, two were selected for in-depth consideration in the following. They were selected because one exhibits good per-feature class separation, while the other does not, i.e. there is a large difference in per-feature class separation. For the well-separating detector result, the classifier achieves both Precision and Recall 1, while for the badly-separating detector result, Precision 1 and Recall 0.99194 are attained. Each of the two datasets was classified by a Random Forest classifier optimized via *SynOpSis*.

Figure 6.7 shows histograms of feature values as observed in three top-ranked features as according to Table 6.1. Features observed in the well-separating detector result are shown in the first row (a)–(c), while the same features as observed in the badly-separating detector result are shown in the second row (d)–(f). Note that among these most frequently selected features only $f_{3.5}^{\text{spot}}$ on the well-separating detector result (a) can provide a good separation if used in a single-feature classifier. The other features, as well as $f_{3.5}^{\text{spot}}$ on the badly-separating detector result, seem to be useful primarily *in conjunction* with further features, because on the single-feature level there is strong class overlap.

Interestingly, class overlap is by far smaller for the least frequently selected features on the well-separating detector result, as depicted in the first row of Figure 6.8. This suggests that they were rated the worst features due correlation with other features already in the selection and thus low usefulness for the classifier. On the badly-separating detector result in the second row of Figure 6.8, their class separation worsens, as expected.

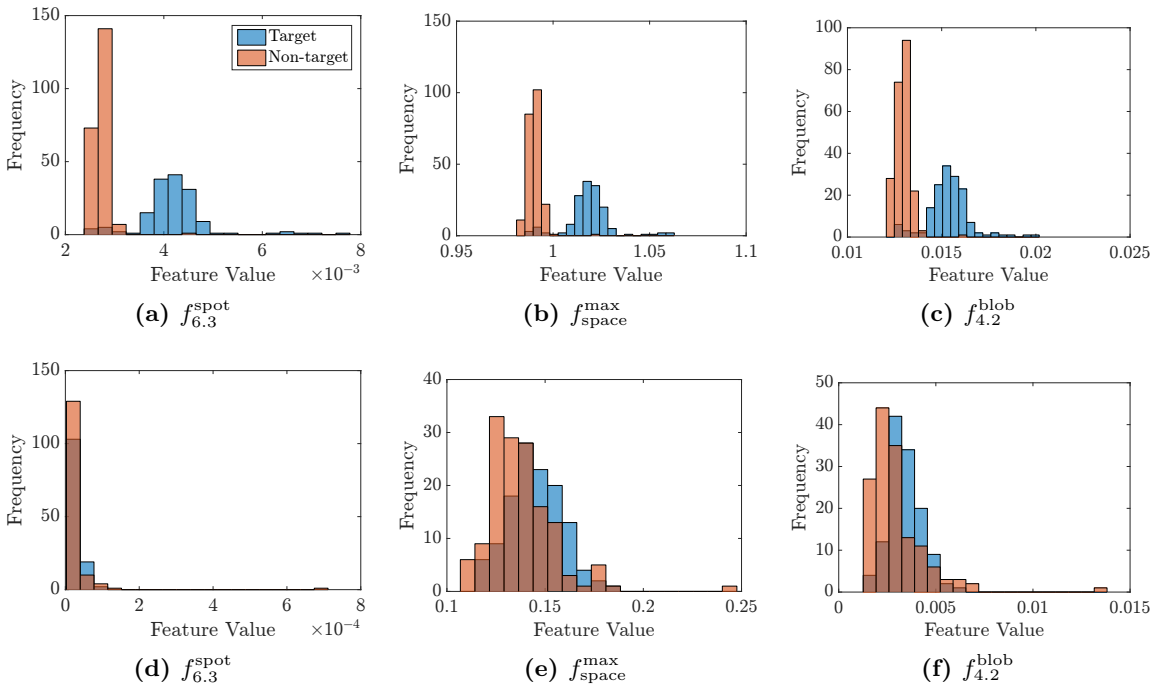


Figure 6.8: Feature Histograms – Bottom Ranks. The same class-separated histograms of feature values as in Figure 6.7 are shown, but the features depicted here are those ranked worst in Table 6.1. The first row (a)–(c) is from the same well-separating detector result as the first row of Figure 6.7, and the second row (d)–(f) is from the same badly-separating detector result. Conclusions are given in the text.

Note that the distributions of feature values are approximately Gaussian in most cases. Exceptions to this rule occur in features with a lower limit of zero and a heavy tendency for this lower limit: It is presumed that possibly Gaussian distributions are compressed towards this lower limit as a result of the generally low intensities and small scales in *PAMONO* data, cf. Section 7.2. Approximate Gaussianity of the other features indicates suitability of the zero-mean, unit-variance transformation used in feature scale normalization (cf. Section 6.4) and of the Gaussian fits that were used to model the class-conditional distributions of feature values in the Naïve Bayes classifier.

6.8 Remaining Parameters of the Classifier

In analogy to Table 5.6 in Section 5.7, which listed the parameters to be optimized for the pattern detector, Table 6.2 lists the parameters remaining to be optimized for the pattern classifier. Design Decisions 6.1 to 6.3 fixed the Random Forest learner to be used with prior class balancing and without feature selection as the final choice for the evaluation of the overall system in Chapter 7. Therefore, all parameters remaining to be optimized for the pattern classifier are the Random Forest parameters described in Section 6.6.3. Table 6.2 lists these parameters along with the examined ranges, which are the same as used in the results section of this chapter, cf. Section 6.7.1.

Table 6.2: Parameters Optimized for the Pattern Classifier after Random Forest was Selected as the Learning Algorithm.

Name	Type	Range	Brief Description
K	int	$\{1, \ldots, 15\}$	Number of features available for splitting
I	int	$\{1, \ldots, 500\}$	Number of trees in the forest
D	int	$\{0, \ldots, 20\}$	Maximum tree depth (0: no limit)

6.9 Conclusion

A pattern classifier was presented to eliminate spurious responses of the pattern detector from Chapter 5. It uses features of polygon shape, spatial and spatiotemporal intensities in order to distinguish target patterns from non-target patterns in the output of the detector. This distinction is realized by a classifying model trained on synthetic ground truth data with a supervised machine learning algorithm. Four eligible algorithms were presented along with optional modules for data preprocessing in terms of class balancing, feature scale normalization and feature selection. Benefits and drawbacks of preprocessing methods and learners were evaluated and discussed, leading to Design Decisions 6.1 to 6.3 restricting the search space to be examined in optimizing the pattern classifier. As a summary, the decisions are to use the Random Forest learner with prior class balancing and without feature selection. Feature scale normalization is not required by this learner. In this setup, the pattern classifier will be used in the evaluation of the overall *SynOpSis* approach for analyzing *PAMONO* data. This evaluation is the topic of Chapter 7.

Future Work

Future work with regard to the pattern classifier may examine an ensemble of classifying models, each obtained as one non-dominated solution in optimizing the algorithmic parameters of the employed learning algorithm. Using multiple non-dominated models means that the entire Pareto front can be considered in the results, respecting all objectives that were deemed important for optimization. Votes can be weighted with respect to model position in objective space: For example a model with low Recall should have lower weight if it classifies an example as negative because low Recall models incur a large number of False Negatives (FNs). This can be regarded as a mixture-of-experts model [JJN+91] in objective space: Each model is an expert for certain predictions, depending on its position in objective space. Clustering can be used to decrease the redundancy and accelerate the application of such an ensemble: The non-dominated solutions can be clustered either in parameter space or in objective space, with the latter option allowing stronger statements about the coverage of the Pareto front.

Furthermore, an ensemble approach enables integration of classifying models produced by different types of learners: As an example, SVM models can be combined with Random Forest models. The SVM models attained higher median Recall than any other learner in Figure 6.3 but suffered from low Precision. Combining both types of models may yield an ensemble that performs well in both objectives.

Chapter 7

Evaluation of SynOpSis for PAMONO

Contents

7.1 Introduction

Chapters 4 to 6 described a concrete realization of the *SynOpSis* approach that was introduced abstractly in Chapter 3. The application case of this concrete realization is automatic analysis of the time series of images produced by the *PAMONO* sensor presented in Chapter 2, which can be used e.g. for real-time-capable detection of biological viruses in liquid samples. Evaluations given in the previous chapters were restricted to finalizing design choices that are not subject to the *Optimization* stage. An evaluation and validation of *SynOpSis* for *PAMONO* as a whole was postponed up to now, i.e. until all its constituents had been presented. This evaluation and validation is the subject of this chapter, examining all parts of *SynOpSis* in interplay, and reporting the results finally attained in *PAMONO* data analysis.

Before these evaluations are presented, the experimental settings on both, the physical and the algorithmic side are depicted: Section 7.2 is concerned with the physical setup of the *PAMONO* sensor. Variations of this setup provide the experimental data used as the basis of this evaluation. Hence these variations are described along a characterization of the resulting data. Section 7.3 delineates the experimental setup of *SynOpSis*, including a determination of the degrees of freedom that were left open up to now. Furthermore, quality measures and different modes of applying *SynOpSis* are discussed, which are examined in the evaluation.

Sections 7.4 to 7.7 then report the results of applying *SynOpSis* for *PAMONO* data analysis. As a starting point, Section 7.4 illustrates *SynOpSis* for a single experiment, displaying the convergence of objectives during optimization, the resulting Pareto front and the detector and classifier confusion matrix as attained by applying the results to the real sensor data. After this single example gave an illustration of the type of results, Section 7.5 delivers cross-validated, aggregate results for all experiments, hence examining *SynOpSis* on a larger scale. In this context, a decision between sequential and two types of global optimization is taken, as well as a decision among three different ways of applying the desirability approach during optimization. With these decisions fixed, the final analysis results for all examined *PAMONO* experiments are presented, followed by an assessment of the quality of performance estimates. Specificity of results is evaluated by analyzing data not containing any nano-objects, and finally, the computational effort of optimization and analysis is examined. In this context, the real-time capability of *PAMONO* data analysis with *SynOpSis* is verified. Section 7.6 identifies algorithm choices and parameter values that occur frequently on Pareto fronts. Thus, this section takes a closer look at what makes a good parameter set for the detector and classifier. Furthermore, a regression model predicting objective function values from parameter sets is computed, and the predictability of objective values is investigated in consideration of a potential meta-modeling-based enhancement of the *Optimization* stage. Finally, Section 7.7 assesses how well parameter sets and classifying models generalize across different *PAMONO* experiments.

7.2 PAMONO Experiments

SynOpSis analysis results were validated with respect to several *PAMONO* experiments, made under varying physical conditions. Section 7.2.1 details these physical parameterizations of the *PAMONO* sensor and their variations. Varying the physical parameters resulted in six different *PAMONO* experiments to be presented in Section 7.2.2. These experiments serve as the basis used throughout this evaluation to demonstrate the capabilities of *SynOpSis*. In

particular, these experiments target the capability of finding suitable algorithmic parameters for *PAMONO* data analysis in face of varying physical parameters of the sensor. In order to quantify the respective difficulty level of analyzing the data from each experiment, Section 7.2.3 measures the observed Signal-to-Noise Ratios (SNRs).

7.2.1 PAMONO Sensor Setup and Variations

In Chapter 2, the general experimental setup of the *PAMONO* sensor is explained. This setup, depicted in Figure 2.2, has many parameters, which are in the following referred to as physical parameters because they are determined in the physical world, before any data is processed. In contrast to that, the parameters optimized by *SynOpSis* are called the algorithmic parameters, as was done before. The concrete choice of the physical parameters of the sensor was left open in Chapter 2 because such a choice describes one particular sensor setup and not a general method. The concrete physical parameterizations [STM+15] of the *PAMONO* sensor given now determine the degrees of freedom remaining from Chapter 2. For the experiments analyzed in this evaluation, different combinations of the following physical parameters were chosen, as will be summarized in the caption of Table 7.1. All components described here can be found in Figure 2.2, and the order of presentation in this list follows the path of the light rays in that figure.

- **Diode (Light Source):**
  As the light source a superluminescent diode (QPhotonics QSDM-680-9) was used, which emits light at a wavelength of 680 nm. Brightness of the light source varies across experiments and was in each case adjusted to closely reach the full well capacity of the respective Charge-Coupled Device (CCD) camera in the brightest observed pixel. This aims at fully utilizing the intensity resolution (range of measurable values) of the CCD.
- **Gold Sensor Surface:**
  Gold layers may exhibit differences in quality, as discussed in more detail in the context of optimizing Surface Plasmon Resonance (SPR) measurement conditions in [ZSS+17]. Within the scope of this thesis, the different physical parameters impacting gold layer quality are summarized in a single rank order comprised of three levels: low, medium and high quality, abbreviated as LQ, MQ and HQ respectively. These quality levels are indicated in the 'Name'-column of Table 7.1. With all other physical parameters unchanged, lower quality of a gold layer results in decreased SNR in the recorded data, cf. also Section 7.2.3. Different gold layers typically vary in terms of production quality, even if they underwent the same manufacturing process. As gold layers wear out after approximately ten positive measurements, such variations are common in practice.
- **Nano-Objects:**
  The nano-objects were either 100 nm or 200 nm polystyrene particles as also indicated in the 'Name'-column of Table 7.1. Polystyrene particles were used as a safe proxy for actual biological viruses and as a more economic alternative to Virus-Like Particles (VLPs) [GA06]. The observed indirect effect caused by a nano-object in *PAMONO* is determined by its size [ZKG+10], not by its type.
- **Coating of the Sensor Surface:**
  The sensor surface needed no coating with antibodies because polystyrene particles can be bound to the surface by means of electrostatic charge.

- **Buffer Solution:**
  The nano-objects were diluted in either a Phosphate Buffered Saline (PBS) plus 0.3% Sodium Chloride (NaCl) buffer solution, or in water (H_2O) plus 0.25% NaCl as indicated in the caption of Table 7.1.
- **Lens:**
  The employed lens was a Minolta MD Rokkor with $f = 50\,\text{mm}$ and aperture 1/1.7.
- **CCD Camera:**
  The images of the sensor surface were recorded using 12 bit grayscale industrial strength CCD cameras. Two such cameras were examined: The Allied Vision Prosilica GC 2450 (2448 px × 2050 px) and the Allied Vision Guppy PRO F-503B (2588 px × 1940 px). They were run at recording rates ranging from 15 to 40 Frames per Second (FPS), while images were cropped to the Region of Interest (ROI) on the sensor, i.e. to the part where the gold sensor surface is in focus. Cropping leads to different resolutions across experiments. All variable camera parameters are listed in Table 7.1, and the respective employed camera type is indicated in its caption.

7.2.2 Description of PAMONO Experiments

Variation in physical parameters is very common in developing prototypes of new sensor technology. Several degrees of freedom in the experimental setup of the *PAMONO* sensor were determined above, while others, like combinations of gold layer quality, nano-object size and camera were left open. The experiments described in the following serve to assess how well *SynOpSis* can adapt to variations in these and other physical sensor parameters. The purpose of these variations is twofold: Firstly, an overall validation of *SynOpSis* analysis results under varying conditions is to be conducted, and secondly, the limits of the method are to be identified, particularly with respect to gold layer quality and nano-object size.

Table 7.1 lists the six experiments examined throughout this evaluation, along with their respective properties and combinations of physical parameters. All details are provided either in the table itself or in its caption, in order to centralize them in a single place that is convenient to reference throughout the remainder of this chapter. To avoid redundancy, the details are not repeated here, and the reader is referred to the table instead.

The different experiments listed in Table 7.1 refer to real data recorded from the sensor. Consequently, no synthetic ground truth is available, but instead, ground truth was created manually by human experts: Any actual nano-object appearing in the data was delineated by a polygon, using the same procedure as for the archetypes measurement in Section 4.3.1. Therefore, the output format of these expert segmentations is the same as that of the synthetic ground truth, and the same method for matching and labeling can be applied in evaluating *SynOpSis* results quality, cf. Section 5.8. Ground truth for real data was used *solely* in assessing analysis quality. It was neither used in optimization, nor model selection, nor performance estimation.

In addition to the real sensor data *with* nano-objects listed in Table 7.1, real sensor data *without* nano-objects was recorded prior to the inserting the specimen into the flow cell. This data will be used in the specificity analysis in Section 7.5.7. Besides these two real datasets, three synthetic datasets were created for each experiment: These datasets are called training, validation and test set, and their roles and purposes are presented in detail in Section 7.3.4. Each such triplet of synthetic datasets was created the same way, following Chapter 4.

Table 7.1: PAMONO Experiments. The *PAMONO* experiments that provided the examined real sensor data are listed in this table. In all cases, polystyrene nano-objects of the sizes indicated in the 'Name'-column were to be detected. The suffixes after nano-object sizes indicate gold layer quality in terms of a rank order: LQ, MQ and HQ are for low, medium and high quality, respectively. Gold layer quality primarily affects Signal-to-Noise Ratio (SNR), and the rank order simplifies measures of gold layer quality discussed in more detail in [ZSS+17]. All experiments were performed using an Allied Vision Prosilica GC 2450 camera to record nano-objects in a PBS plus 0.3% NaCl buffer solution. An exception is the 200 nm Gpy experiment, which used an Allied Vision Guppy PRO F-503B camera (hence the suffix 'Gpy') and an H_2O plus 0.25% NaCl buffer solution. Column 'G' displays the number of ground truth nano-objects in the data, as determined by a human expert. The next column contains the size of the spatiotemporal volume to be analyzed: Its first two values are the size of the recorded Region of Interest (ROI), i.e. the focus region of the sensor. Its third value is the number of images, which were recorded at the rate indicated by the 'FPS'-column, where FPS means Frames per Second. Finally, the last three columns display the minimum, median and maximum SNRs observed in each experiment, cf. Section 7.2.3.

Name	G	Volume Size (px^3)	FPS	SNR^{min}	SNR^{med}	SNR^{max}
200 nm HQ	352	$1080 \times 145 \times 2000$	20	1.75363	2.20402	3.20017
200 nm MQ	333	$742 \times 127 \times 2000$	20	1.37127	2.50311	5.10115
200 nm LQ	93	$706 \times 167 \times 2000$	20	1.22561	2.12493	4.34309
200 nm Gpy	195	$1024 \times 270 \times 500$	15	2.04267	3.73221	6.69059
100 nm HQ	195	$750 \times 230 \times 4100$	40	0.92184	1.82544	5.16014
100 nm LQ	56	$450 \times 170 \times 4000$	40	0.82224	1.24673	5.09924

Downloading PAMONO Experiments

In the spirit of reproducible research, five of the six *PAMONO* experiments listed in Table 7.1 were made publicly available on the internet. This includes real sensor data with and without nano-objects, as well as synthetic data with ground truth. The Digital Object Identifiers (DOIs) for the individual experiments can be found in the bibliography under the following references: 200 nm HQ [SZS+14c], 200 nm MQ [SZS+14e], 200 nm LQ [SZS+14d], 100 nm HQ [SZS+14b], 100 nm LQ [SZS+14a]. Note that by the time this thesis was written, experiment 200 nm Gpy was not available on the internet.

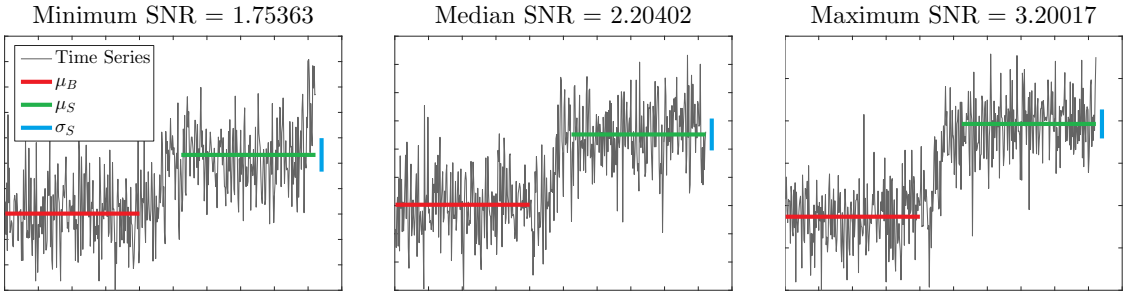
7.2.3 Signal-to-Noise Ratios

The Signal-to-Noise Ratio (SNR) is a frequently used measure in quantifying the quality of data in signal processing tasks [CWG01; SLN+09; JZK+07; TRS+02; FM06; SHM+12; MBW+12]. The SNR of a *PAMONO* time series is defined in agreement with [CWG01] as

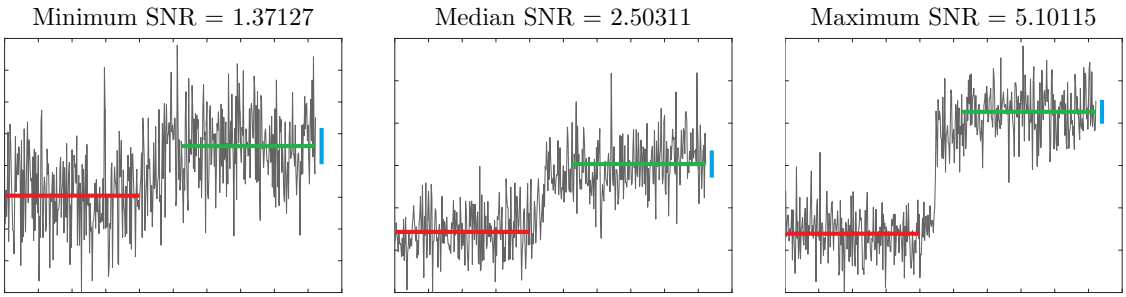
$$\mathrm{SNR} = \frac{(\mu_S - \mu_B)}{\sigma_S}, \tag{7.1}$$

the constituents of which are illustrated e.g. in Figure 7.1a: For a given single time series with step[1] time t, μ_S is the mean time series value after time t, and μ_B analogously before time t. Subscripts are for step and base level. The difference $(\mu_S - \mu_B)$ in the numerator estimates the magnitude of the step signal. The denominator σ_S, i.e. the standard deviation

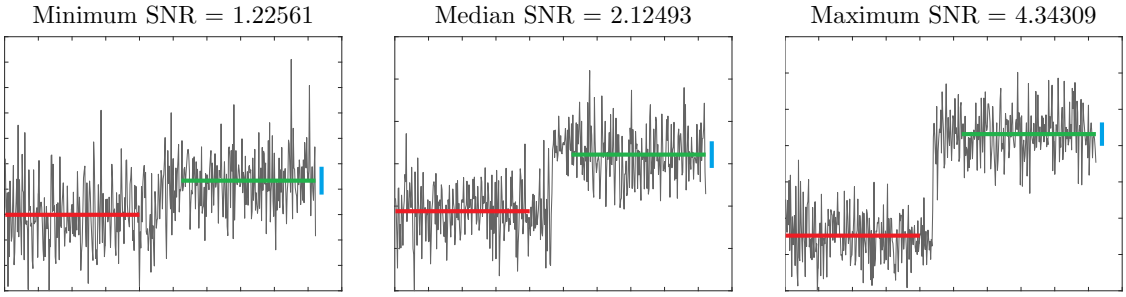
[1] SNRs were measured on the raw time series data as provided by the CCD camera, therefore nano-object adhesions manifest as step-like functions in the time domain, as illustrated in Figure 2.2.



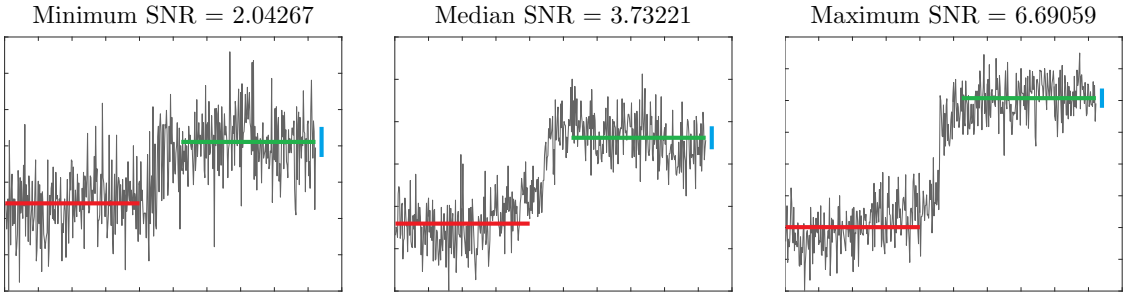
(a) 200 nm HQ



(b) 200 nm MQ

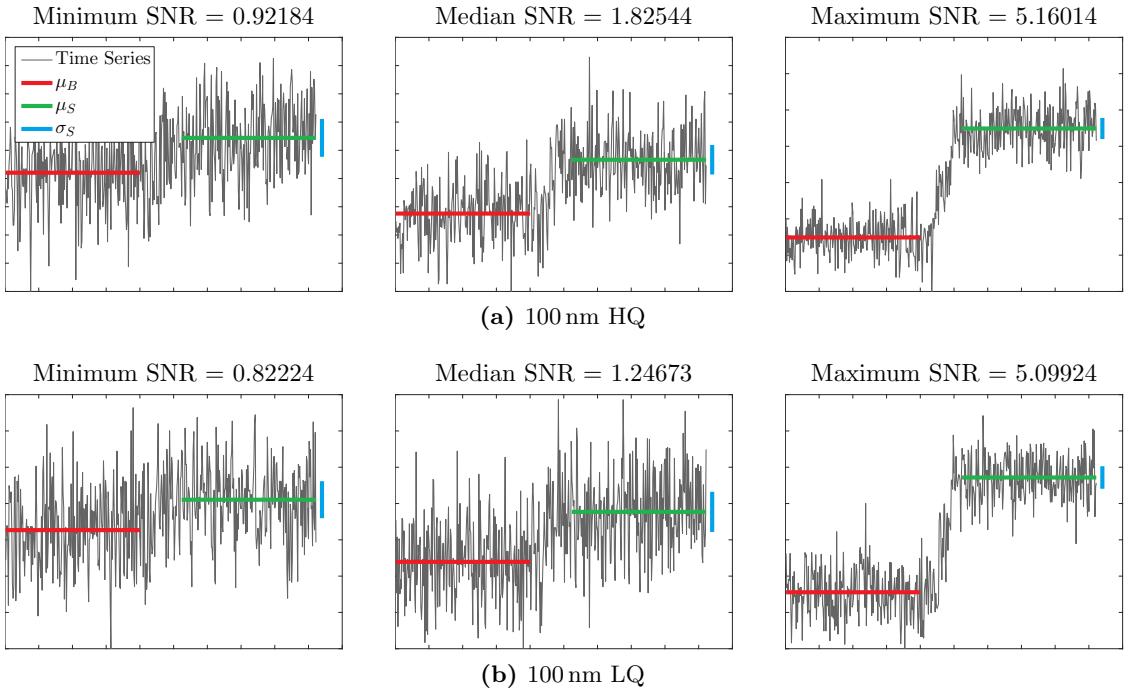


(c) 200 nm LQ



(d) 200 nm Gpy

Figure 7.1: Signal-to-Noise Ratios in Experiments with 200 nm Nano-Objects. Minimum, median and maximum SNRs of the 200 nm *PAMONO* experiments listed in Table 7.1 are illustrated by plotting the respective raw time series and the terms constituting the SNRs, as defined by Equation (7.1).



(a) 100 nm HQ

(b) 100 nm LQ

Figure 7.2: Signal-to-Noise Ratios in Experiments with 100 nm Nano-Objects. Minimum, median and maximum SNRs of the 100 nm *PAMONO* experiments listed in Table 7.1 are illustrated by plotting the respective raw time series and the terms constituting the SNRs, as defined by Equation (7.1).

of the values averaged in μ_S, estimates the magnitude of the noise, making Equation (7.1) a Signal-to-Noise Ratio. The lengths of the regarded temporal windows were 200 frames each for μ_S, σ_S and μ_B, respectively. Note that ±30 frames centered about t were ignored in computing the SNR, in order to make it robust against the exact temporal coordinate of the step. This can be seen as a small gap between the two horizontal lines in Figures 7.1 and 7.2.

Given this definition of the SNR of a single time series, the notion of SNR is extended to single nano-objects and to entire *PAMONO* experiments as follows: For a single nano-object, defined as a spatial region at a step time t, the SNR is defined as the maximum value attained by Equation (7.1) over that region. With SNRs for the individual nano-objects in an experiment defined, the SNR of that experiment itself is characterized by the minimum, median and maximum SNRs observed over the nano-objects. These are the values reported in the corresponding columns in Table 7.1. Note that computing these values in practice assumes availability of ground truth information because regions and step times of the nano-objects appearing on the sensor are required. To this end, the manually created ground truth also used in assessing overall analysis quality on real sensor data can be used.

Figures 7.1 and 7.2 give an impression of the minimum, median and maximum SNRs as observed in each of the six *PAMONO* experiments listed in Table 7.1. Each individual figure plots the respective time series, along with the values of μ_S, μ_B and σ_S. Note that some minimum SNRs in the 200 nm experiments are close to one, while those in the 100 nm experiments are even below one. Visually distinguishing these steps from pure noise on the level of regarding individual time series is a demanding task. Manual ground truth creation in these cases can only take place after denoising and by considering multiple spatially

adjacent time series simultaneously. The SNR *measurements*, however, i.e. the evaluations of Equation (7.1), were all carried out on the raw data as provided by the CCD camera.

Note that the maximum SNRs of the 100 nm experiments shown in Figure 7.2 are large in comparison to the median and minimum SNRs. Examining the steps more closely reveals what causes these large SNRs: Two nano-objects attach in high spatial and temporal proximity. The first nano-object attachment causes a step about half the height of the final step. A short noisy "plateau" follows, from which a second step related to the second nano-object commences, which takes the overall step to its final height, resulting in the high SNR. Visual inspection of the processed images in the spatial and temporal domain confirmed the presence of two partially overlapping nano-objects.

Conclusion

A notion of a Signal-to-Noise Ratio (SNR) was defined for *PAMONO* time series, nano-objects and experiments. SNRs of the six examined *PAMONO* experiments were determined (Table 7.1), providing a quantitative measure of the difficulty of the respective analysis tasks.

Considering minimum and median SNRs, the difficulty of these analysis tasks in several cases exceeds that of tasks solved in the literature: All algorithms surveyed in [CWG01] fail for SNRs approaching four, while [SLN+09] report some methods that can handle SNRs as low as two[2]. One of the goals of the evaluations given in this chapter is to show that by optimized combination of denoising methods in two spatial and one temporal dimension, *SynOpSis* can find nano-objects in *PAMONO* data with a median SNR close to one, i.e. at least half of the occurring nano-objects are likely to be missed by the best algorithms surveyed in [CWG01] and [SLN+09].

7.3 Setup of SynOpSis for PAMONO

While Section 7.2 determined the remaining degrees of freedom in the physical parameters of the *PAMONO* sensor and presented the experimental data to be used in this evaluation, this section will determine the remaining degrees of freedom in the setup of *SynOpSis* and present the hardware upon which it is run.

SynOpSis is employed to find optimized values for the 28 parameters of the detector, as listed in Section 5.7 and for the three parameters of the Random Forest classifier, as listed in Section 6.8. Section 7.3.1 briefly summarizes the objectives with respect to which the parameters are optimized and introduces two further measures used in reporting results. Section 7.3.2 presents the configuration of the Genetic Algorithm (GA) along with two examined modes for global and one mode for sequential optimization. Section 7.3.3 gives the concrete parameterizations of the employed desirability functions and discusses three modes of using these functions during optimization. Section 7.3.4 presents the concrete cross-validation strategy used for model selection and performance estimation, while Section 7.3.5 describes how the final classifying model is learned. Finally, Section 7.3.6 lists the hardware of the computer used for all measurements in this chapter.

[2]Note that *PAMONO* data has two spatial and one temporal dimension while the studies in [CWG01; SLN+09] relate to images with two spatial dimensions only. As a consequence, those studies measure SNR with respect to the two spatial dimensions. Table 7.1, on the other hand, lists SNRs measured with respect to the temporal dimension because the presented detector (Chapter 5) is based on time series classification.

7.3.1 Objectives and Reported Measures

The measures used as objective functions in optimizing the detector and classifier parameters have been presented in detail in Sections 3.5.2 and 3.6.2. Now the purpose of the following itemization is to briefly list these objectives in one place for reference, while assigning unique names to them which are used throughout this chapter. Furthermore, Terminology 3.1 is concretized towards the *PAMONO* context:

Terminology 7.1. *In the context of* PAMONO *data analysis, the terms* ***pattern***, ***target pattern*** *and* ***non-target pattern*** *from Terminology 3.1 are used synonymously with their concrete* PAMONO *instances: Target patterns are the actual* ***nano-objects*** *to be detected in* PAMONO *data, while non-target patterns are the spurious detector responses not related to actual nano-objects, but to* ***artifacts****. The superordinate abstract term of patterns refers to the union of all detector responses, i.e. to the union of nano-objects and artifacts. Hence it is synonymously denoted as the set of* ***candidate*** *objects because each pattern is a candidate for being related to an actual nano-object.*

This terminology is used interchangeably with its more abstract variant, depending on the context. In stating the objectives to be optimized for *PAMONO* in the following, the concrete terminology is used.

Measures Used as Objectives in Optimization

Recall^{D}
: Detector Recall, as defined in Equation (3.3), is denoted as Recall^{D} in the following. It measures the ratio of nano-objects that have been *found* by the detector, among *all* nano-objects in the data. Note that Recall^{D} is defined with respect to the number $\widehat{\text{TP}}$ of detector responses cleansed from multiple detections. This prevents rewarding repeated detections in optimizing this objective, as discussed in Section 3.5.2.

M-Rate
: Detector M-Rate, as defined in Equation (3.2), is to be minimized. It penalizes multiple detection of the same nano-object.

$\text{Precision}^{\text{C}}$
: Classifier Precision, as defined in Equation (3.5), is denoted as $\text{Precision}^{\text{C}}$ in the following. It measures the ratio of *actual* nano-objects that have been classified as nano-objects, among *all* detector responses that have been classified as nano-objects.

Recall^{C}
: Classifier Recall, as defined in Equation (3.4), is denoted as Recall^{C} in the following. It measures the ratio of *actual* nano-objects that have been classified as nano-objects, among the subset of *all* actual nano-object responses provided by the detector. This means in particular, that nano-objects missed by the detector do not influence Recall^{C}.

Reported Measures

Besides the objectives listed above, two further measures of analysis quality are reported in the evaluations in this chapter. These measures are not subject to optimization but merely serve informative purposes. This should be kept in mind when interpreting the reported results.

Precision$^{\mathrm{D}}$

This quantity measures Precision (cf. Appendix A) attained by the detector. Unlike Recall$^{\mathrm{D}}$, Precision$^{\mathrm{D}}$ is not defined with respect to the number $\widehat{\mathrm{TP}}$ of TP detector responses cleansed from repeated detections:

$$\mathrm{Precision}^{\mathrm{D}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{7.2}$$

Instead, TP is the *raw* number of true positive detector responses, and FP is its number of false positives, as shown in the confusion matrix of the detector in Table 3.1. The reason for using TP instead of $\widehat{\mathrm{TP}}$ in Equation (7.2) is that Precision$^{\mathrm{D}}$ serves as a measure of class balance for the subsequent classifier: It is defined as the ratio of actual nano-objects (TP detector responses) among all candidate objects passed to the classifier (TP + FP detector responses). Therefore, Precision$^{\mathrm{D}}$ directly reports class balance. A low value of Precision$^{\mathrm{D}}$ does not imply low quality analysis results, as long as the objective Precision$^{\mathrm{C}}$ of the classifier is large enough to sort out the FPs of the detector. A value of Precision$^{\mathrm{D}} = 0.5$ means that the input for the classifier is perfectly class-balanced, while Precision$^{\mathrm{D}} = 1$ means that the ideal classifying model is the trivial model that classifies all candidate objects as positives. In this case the detector did not respond to any spurious artifact.

D-Rate

While Precision$^{\mathrm{D}}$ provides information on how important a good classifying model is for a good analysis, D-Rate is a measure of overall analysis quality. It quantifies the relative deviation of the number of reported nano-objects from the actual number of nano-objects and is defined as

$$\text{D-Rate} = -1 + \frac{\mathrm{TP} + \mathrm{FP}}{G}, \tag{7.3}$$

where TP and FP refer to the confusion matrix of the classifier in Table 3.2. Hence the sum in the numerator is the number of detector responses finally reported by the classifier as being related to nano-objects. The denominator G is the number of actual ground truth nano-objects. Therefore D-Rate measures the relative deviation between the number of reported nano-objects and their actual number G. Positive values of D-Rate indicate that the number of target patterns was overestimated, whereas negative values indicate underestimation. A value of 0 means that the number was estimated exactly.

Note that while D-Rate is an intuitive measure to report as a summary of overall analysis quality, it is not a good objective to be optimized: An argument for this can be seen, if the number of ground truth patterns G is decomposed as follows:

$$G = \mathrm{TP} + \mathrm{FN} - R + U,$$

where TP and FN refer to entries of the classifier confusion matrix in Table 3.2. R is the number of repeated responses of the detector to single nano-objects, i.e. the number of *surplus* responses to actual nano-objects. U is the number of undetected nano-objects in the ground truth and is thus equivalent to the FN entry of the detector confusion matrix in Table 3.1. With G decomposed, Equation (7.3) becomes

$$\text{D-Rate} = -1 + \frac{\mathrm{TP} + \mathrm{FP}}{\mathrm{TP} + \mathrm{FN} - R + U}, \tag{7.4}$$

from which it can be immediately seen, why D-Rate is not suitable as an objective function to be optimized: Optimizing it means minimizing its absolute value, aiming at zero deviation. Zero absolute D-Rate is attained for any case where numerator and denominator in the previous equation have the same values, i.e. if FP equals $\mathrm{FN} - R + U$. Optimizing D-Rate as a single objective neither rewards large values of classifier TPs, nor does it penalize the adverse components in Equation (7.4). Instead, it aims at a certain ratio between them. In contrast to that, the optimized objectives do all this. Hence, optimizing for these objectives and merely *assessing* deviation in terms of D-Rate yields an intuitive summary measure of analysis results quality.

Early Cancellation

As a means for accelerating objective function evaluations during optimization, an early cancellation mechanism is used: In case the number of detector responses is a times larger than the number G of ground truth nano-objects, processing is canceled, and objectives are assigned worst-case dummy values indicating this condition to the GA. Hence over-sensitive parameters, that indiscriminately respond to everything, are precluded and the time taken for matching these responses to the ground truth is saved. A value of $a = 5$ was chosen empirically.

7.3.2 Genetic Algorithm Settings

Section 3.7 introduced the methodological prerequisites and terminology required for conducting optimization via Multi-Objective Genetic Algorithms (MOGAs). While MOGAs are very general metaheuristics, they allow for some adaptation towards different application scenarios by means of parameters to be configured. The following list summarizes the particular MOGA settings used by *SynOpSis* for optimizing the objectives in Section 7.3.1:

- The **Non-Dominated Sorting Genetic Algorithm II (NSGA-II)** [DPA+02], as presented in Section 3.7.5, was chosen as the concrete MOGA to be employed.
- The **population** size and the number of **generations** were chosen as follows: In global optimizations, populations of size 50 were optimized over 50 generations. In sequential optimizations, these numbers were the same with regard to optimizing detector parameters, and for the subsequent optimization of classifier parameters, population size and generation count were both set to 20. In all cases, the size of the external NSGA-II elite was equal to the respective population size.
- **Initialization** of the population was conducted by drawing each parameter uniformly at random from the set of its valid values.
- The tournament **selection** scheme was used with tournament size two, both of which are the most common choices in GAs [Luk13] and for NSGA-II [DPA+02].
- Chromosomes are recombined via non-empty one-point **crossover**. As a preparation for this, the parameter vector is mapped unto the chromosome such that parameters belonging to the same processing element are placed in adjacent genes on the chromosome, i.e. high linkage exists particularly between *adjacent* genes. One-point crossover has a higher probability of breaking genes (and thus parameters) apart that reside further apart on the chromosome, compared to those residing more closely. Encoding highly

linked parameters in genes residing in close proximity on the chromosome makes the one-point crossover operator a suitable choice.
- **Mutation** of a gene takes place with probability 0.1, and the mutation operator assigns a new value by drawing uniformly from the set of values that parameter may assume.

With the MOGA configured as described above, three variants of the *Optimization* stage of *SynOpSis* are evaluated in this chapter, as presented now.

Global Optimization with Four Objectives

This variant means that all four objectives listed in Section 7.3.1 are optimized simultaneously. Hence the detector and classifier are optimized in conjunction, as detailed in Section 3.7.6.

Global Optimization with Three Objectives

This is a variation of the previous four-objective setup for global optimization. The rationale behind reducing the number of objectives is related to choosing NSGA-II for the *Optimization* stage: Even though in their original paper [DPA+02], Deb et al. applied NSGA-II on a problem with five objectives and seven constraints, Jain and Deb later stated that optimizing more than three[3] objectives may leave the domain of application where NSGA-II is effective [JD13].

Therefore, a global optimization with three objectives is conducted, leading to a territory in which NSGA-II is known to work well. The number of objectives is reduced by optimizing the product

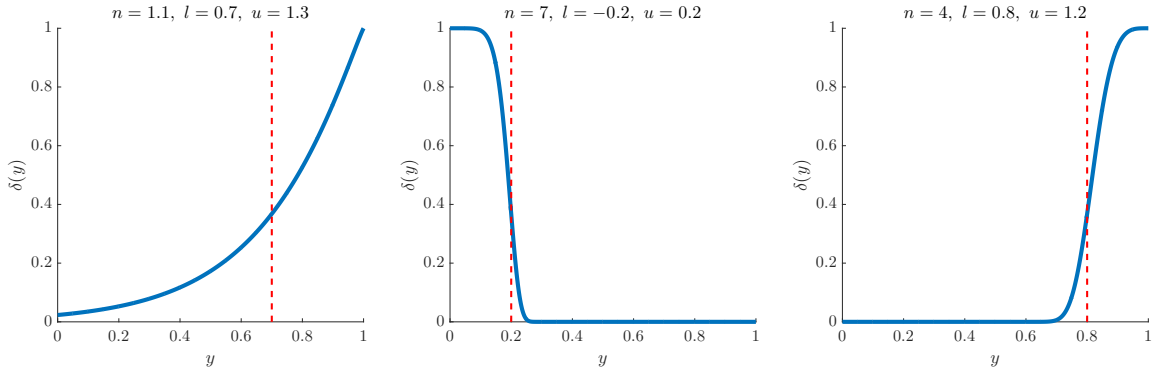
$$\text{Recall}^{\Pi} = \text{Recall}^{\text{D}} \cdot \text{Recall}^{\text{C}}, \tag{7.5}$$

instead of optimizing Recall^{D} of the detector and Recall^{C} of the classifier as separate objectives. The rationale behind optimizing this product objective is that for overall results quality *assessment* it does not matter, whether missed nano-objects are missed by the detector or by the classifier. During *optimization* however, this coupling of objectives can have adverse effects because it makes individuals with different Recall^{D} and Recall^{C} but equal Recall^{Π} indistinguishable in objective space.

Sequential Optimizations with Two Times Two Objectives

As detailed in Section 3.7.6, a sequential optimization of the detector, followed by optimizing the classifier can be conducted to find out, whether dividing global optimization into two separate optimizations with two objectives each, reduces the quality of the attained results. Comparing global to sequential optimizations serves to determine whether being able to devalue detector parameters that produce patterns that are hard to classify, as is possible in

[3] Multi-objective optimization with more than three objectives is denoted as many-objective optimization [JD13]. In 2013, Jain and Deb proposed NSGA-III for many-objective optimization, which was demonstrated to work well on problems with up to 15 objectives [JD13]. However, NSGA-III has issues in dealing with less than three objectives, which gave rise to its follow-up U-NSGA-III [SD15] (cf. [SD14] for a more detailed version) as a unified approach to single-, multi- and many-objective optimization. These techniques have not been considered in this thesis due to the lack of reference implementations. The overhead of producing one was deemed unjustified in the context of this thesis, given the quality of results achieved with NSGA-II [DPA+02] for global optimization, cf. particularly Section 7.5.



(a) Recall$^{\mathrm{D}}$, Recall$^{\Pi}$ **(b)** M-Rate **(c)** Precision$^{\mathrm{C}}$, Recall$^{\mathrm{C}}$

Figure 7.3: Desirability Functions (DFs) Used in PAMONO Data Analysis. The DF used for Recall$^{\mathrm{D}}$ and Recall$^{\Pi}$ (a) does not saturate and is non-zero even for low objective values. In contrast to that, the DFs for M-Rate (b), respectively Precision$^{\mathrm{C}}$ and Recall$^{\mathrm{C}}$ (c) do saturate around their target values and assign desirability ≈ 0 in case objective values are too far away from the targets. DF coefficients are provided above the plots, while the heuristics behind their choice is explained in the text.

global optimization, has a positive influence on results quality in practice. This feedback is not possible in sequential optimization because the detector is optimized separately, and the classifier must learn from whatever patterns the optimized detector parameters produce.

In order to refer to these three variants of executing the *Optimization* stage, the following abbreviations are introduced:

Terminology 7.2. *The different* ***optimization modes*** *presented above will in the following text and plots be denoted as* ***Global 4***, ***Global 3*** *and* ***Sequential***, *respectively.*

7.3.3 Desirability Settings

Section 3.8 presented the desirability approach to formalize expert preferences. The desirability approach enables automatic selection of the single "best" (in terms of desirability) individual from a Pareto front *after optimization*, and it can furthermore be used to narrow the search region to the relevant parts of the Pareto front *during optimization*. Section 3.8 provides the methodological background and will now be complemented with the concrete desirability settings chosen for the *PAMONO* objectives listed in Section 7.3.1.

Harrington's two-sided Desirability Function (DF) [Har65; TW06] as defined in Equation (3.7) was used for all objectives. A single DF of this type exhibits three parameters n, l, u, encoding expert preferences for the associated objective. Figure 7.3 provides the values of these parameters for all objectives, along with plots of the resulting DFs. The kurtosis parameter n controls the peakedness of the DF, with higher values of n making constraint satisfiability softer around the target value, cf. Section 3.8 for details. Parameters l and u are the lower and upper specification limit, respectively. Since all objectives in Figure 7.3 are one-sided[4], only one of these parameters is shown as a dashed vertical line, while the other

[4]With all objectives being one-sided, the two-sided DFs were chosen not for their two-sidedness but for their convenient control of peakedness in terms of the kurtosis parameter n, which the one-sided Harrington DFs do not offer, cf. Section 3.8.

one resides on the irrelevant side and was chosen symmetrically to center the peak of the DF upon the target value $\frac{u+l}{2}$ of the objective. Concrete choices of the coefficients n, l, u (top part of Figure 7.3) for all objectives adhere to the following heuristics:

Recall^{D}, Recall^{Π} (Figure 7.3a)

As can be seen from the plot, the DF of Recall^{D} (and Recall^{Π}) does not saturate, thus favoring a highly sensitive oversegmentation in the detector. Choosing a small n results in increasing slope, instead of a decreasing one as would be the case in saturation. If regarded as a constraint, this DF is a constraint that is never fully satisfied, except if the underlying objective exactly attains the target value of one. However, it is also never completely violated, i.e. even very small values in the underlying objective result in a non-zero value in the DF, which serves to guide optimization into the correct direction: Detector parameters capable of attaining even low non-zero Recall^{D} should be treated differently from those with zero Recall^{D}, otherwise optimizing DFs would have no sense of search direction in case of low Recall^{D}. As high values in Recall^{C} are obtained more easily than in Recall^{D}, the latter is regarded as the bottleneck in optimizing Recall^{Π}, which is why the same DF is chosen for Recall^{D} and Recall^{Π}.

M-Rate (Figure 7.3b)

M-Rate on the other hand can be regarded as a constraint that is both violable and satisfiable: Coefficient values are chosen such that for an M-Rate above 0.25, the DF becomes ≈ 0, and for M-Rate below 0.10 it becomes ≈ 1.

$\text{Precision}^{\text{C}}$ and Recall^{C} (Figure 7.3c)

DFs of $\text{Precision}^{\text{C}}$ and Recall^{C} were chosen equal because both objectives are deemed equally important. Encoding this in terms of two identical DFs instead of e.g. optimizing the arithmetic mean of these objectives bears the advantage that it specifies more exactly, which part of the Pareto front is to be searched: If the geometric mean Desirability Index (DI) from Equation (3.8) is chosen as a single objective, this tells the optimizer that a low desirability in one objective can not be compensated for by a high value in the other. The geometric mean DI assumes a low value if one of its factors assumes a low value, independent of the other factors.

Coefficients of the DF were chosen rather tightly: The DF grows, starting from about 0.7, thus asking for classifiers that are markedly better than random guessing of the two classes (which would attain $\text{Precision}^{\text{C}}$ and Recall^{C} values around 0.5 in case of perfect class balance). Starting from objective values about 0.95, the DF is close to one. The rationale behind making the DF soft on the high end is as follows: Mediocre sensitivity of the detector, i.e. mediocre Recall^{D}, frequently results in the absence of FP detector responses because only the most salient nano-objects are targeted by the detector parameter set. This makes the classification task trivial: A classifier is constructed that simply classifies all candidates as nano-objects, and both, $\text{Precision}^{\text{C}}$ and Recall^{C}, attain their maximum of one. Despite the good classifier results, parameter sets with mediocre Recall^{D} should be easily dominated by parameters that trade off incurring some misclassifications for better Recall^{D}. Making the DFs of $\text{Precision}^{\text{C}}$ and Recall^{C} soft on the high end realizes this heuristic because the returns for increasing $\text{Precision}^{\text{C}}$ and Recall^{C} diminish beyond 0.95, while the returns for increasing Recall^{D} do not.

For selecting the single most desirable parameter set from a Pareto front obtained in the *Optimization* stage of *SynOpSis*, the geometric mean DI from Equation (3.8) [TW06] is used.

This is one of three purposes for which the desirability approach can be used in *SynOpSis*, as discussed in detail in Section 3.8.3. These three purposes will be termed now for future reference within text and plots.

Terminology 7.3. *Three **Desirability modes** are evaluated: Desirability mode **Disabled** means that computing DFs is disabled during optimization but is used solely after optimization, in selecting the individual maximizing the geometric mean DI from the Pareto front. Search is conducted over the full front, and the result can be used to explore the trade-offs between objectives. Desirability mode **Multi-Objective** optimizes the DFs in a multi-objective fashion, instead of doing so for the raw objective values. It aims at narrowing the region visited in raw objective space and by that at increasing convergence speed. Desirability mode **Scalarizing** pursues the same goals but does so by single-objective optimization of the geometric mean DI. The strategy for selecting individuals from the Pareto front is the same in all modes.*

Desirability modes and optimization modes can be combined arbitrarily, and all arising nine combinations are evaluated in detail in Section 7.5.

7.3.4 Model Selection and Performance Estimation Strategies

Cross-validation is a technique that originated in statistics and machine learning to prevent undue optimism in the assessment of performance [Koh95; SH97]. *SynOpSis* applies cross-validation not only to its machine learning-based components that constitute the classifier, but to the overall optimization approach: Parameters are tuned for both, the detector and classifier, and parameter selection as well as performance estimation are embedded into this cross-validation, aiming at avoiding overfitting and optimism. Hence, disjoint datasets are used in optimization (parameter tuning), model selection (picking the most desirable parameters from the Pareto front) and performance estimation (predicting performance to be expected on unseen data). Using disjoint datasets in all three mentioned tasks prevents optimism[5] in each one. The thus-required three datasets and their roles will be explained now.

Datasets Used in Cross-Validation

Conducting optimization, model selection and performance estimation with respect to a separate dataset each, can easily be embedded into a three-fold cross-validation, wherein one third of the data is used for each task. Throughout the course of the evaluations in this chapter, this strategy is adhered to. As each fold of this cross-validation requires a separate run of the *Optimization* stage, the number of folds is kept at its minimum of three.

In order to denote the three employed synthetic datasets, Terminology 3.3 is used: Detector and classifier parameters are optimized on the **training** dataset, the single best parameter set is selected from the Pareto front with respect to its DI on the **validation** dataset, and its performance is estimated on the **test** dataset. Doing so constitutes one fold of cross-validation. Then the roles of the datasets are swapped, and the process is iterated, constituting the next fold. Regarding only derangements of the roles (i.e. roles are permuted such that no dataset assumes the same role twice) results in three-fold cross-validation. This cross-validation can be regarded as an outer loop around the *Optimization* stage, model selection and performance

[5] Theoretical background on the issue of optimism was given in Section 3.9.3, cf. also [SH97].

estimation in the overview of the *SynOpSis* approach in Figure 3.2. Each fold yields different detector parameters and thus a different classifying model to be used in the *Application* stage for analyzing the real sensor data.

Synthesis of Training, Validation and Test Datasets

Prior to this cross-validation, and specifically for the experiment to be analyzed, the *Synthesis* stage (cf. Chapter 4) generates the three disjoint datasets, each consisting of synthetic images and a ground truth segmentation. As input, the *Synthesis* stage receives a background measurement $I^{T=1}$ recorded prior to inserting nano-objects. $I^{T=1}$ should be three times as long as the intended length of a single one of the three datasets because one third of this data serves as the background measurement in each. Hence the length of the synthetic training, validation and test dataset is invariant under swapping the roles between folds. Besides the disjoint background measurements, disjoint sets of archetypes, i.e. of template nano-objects, are used. Therefore, cross-validation increases the manual segmentation effort arising in the *Synthesis* stage by factor three. In the evaluations within this chapter, 20 archetypes were manually segmented per dataset. Over all datasets and experiments, synthetic nano-objects were rendered at a constant density of $2.5 \cdot 10^{-6}$ nano-objects per pixel of the underlying third of the background measurement $I^{T=1}$. This is roughly three times the average density observed in the real sensor data, cf. Table 7.1.

Optimization on the Training Dataset

The *Optimization* stage uses solely the training dataset, and one evaluation of detector and classifier parameters works as follows: The detector is run on the training images, and Recall^{D} and M-Rate are evaluated by matching the result to the synthetic ground truth. Furthermore, this matching assigns ground truth labels to the detector output. For computing classifier objectives, an additional five-fold cross-validation is used to reduce undue optimism in them, cf. [Koh95] and Section 3.9. This inner cross-validation is applied only during optimization, and it pertains *solely* to the classifier. Hence the detector output processed within this cross-validation remains the same over folds. During classifier cross-validation, this output is divided into five subsets using stratified sampling, and the classifying model in each fold is learned using the classifier parameters to be currently assessed. Objective values reported for the classifier are the averages achieved over the folds.

If the selected optimization mode is *Sequential*, the detector is optimized first, until its final parameters have been determined, followed by optimizing the classifier separately. In optimization modes *Global 4* and *Global 3*, detector and classifier objectives are optimized together. Note that in global optimization modes, classifier objectives serve two purposes: Firstly, they assess the quality of the classifier parameters, and secondly, they indirectly measure "classifiability" of the data produced by the detector, thus providing feedback to detector parameter optimization. In sequential optimization, i.e. with detector parameters already fixed, classifier objectives can only influence classifier parameters, as no feedback can be given to the optimization of detector parameters.

Model Selection on the Validation Dataset

Model selection in *SynOpSis* serves not only to select an appropriate classifying model in terms of the employed classifier parameters but it encompasses detector parameters as well, hence counteracting overfitting of both, detector and classifier parameters. The objectives measuring the quality of these parameters are evaluated with respect to the unseen validation dataset. A scalarization of the objective vector in terms of the geometric mean DI (cf. Section 7.3.3) is used, i.e. the expert preferences modeled in the desirability approach are incorporated. The detector and classifier parameters that maximize the DI on the unseen validation dataset are selected.

Performance Estimation on the Test Dataset

Performance estimation utilizes the detector and classifier parameters chosen by model selection: The selected detector parameters are applied to analyze the training dataset, and the result is labeled by matching it to the ground truth. Then, from the labeled data, a classifying model is learned using the selected classifier parameters. Finally, the selected detector parameters and the trained classifying model are used to analyze the unseen test dataset, and the objective values resulting from this analysis serve as the performance estimates. Using a fully separate test set ensures that no information about this test set is used in neither optimization and training, nor in model selection.

7.3.5 Computing Classifying Models

Tightly connected to model selection and performance estimation, as discussed in the previous section, is the computation of the classifying models used to these ends, and of the final classifying model used to analyze real sensor data. As argued for in Section 6.7.1, the Random Forest algorithm is used to learn all classifying models.

Classifying Models Used for Training, Validation and Test Datasets

During optimization, model selection and performance estimation, classifying models are learned solely from the ground truth-labeled detector results for the training dataset. In optimization, the classifier parameters are input by the MOGA, and the inner five-fold cross-validation described in Section 7.3.4 generates subsets for testing. In model selection, the classifier parameters are those to be scored, and ground truth-labeled detector results for the validation dataset are used for testing. In performance estimation, the classifier parameters are those chosen by model selection, and ground truth-labeled detector results for the test dataset are used for testing. *Any classification results reported in the course of this chapter with respect to training, validation and test datasets are based on classifying models from performance estimation, learned from the training dataset only, using the classifier parameters from model selection.*

Classifying Model Used for Real Sensor Data

The final classifying model used by the *Application* stage to classify the real sensor data is learned as follows: In order not to waste any data, it is learned from the union of the ground truth-labeled detector results for the training, validation and test datasets. The employed

detector and classifier parameters are those chosen by model selection. In order to attain real-time-capability, this final classifying model is learned prior to the *Application* stage, as all its constituents are already available. *Any classification results reported in the course of this chapter with respect to real sensor data are based on classifying models created as described in this paragraph.*

7.3.6 Measurement System

As the last component in the utilized setup of *SynOpSis* for analyzing *PAMONO* data, the employed hardware and operating system are listed here:

System Specification 7.1. Intel® Core™ i7-2600 with four cores at 3.4 GHz and eight threads. Cache size: 8192 kB. Memory: 12 GB DDR3 RAM at 1333 MHz. GPU: Nvidia® GeForce® GTX 980 (GM204), 2048 shaders at 1126 MHz (boost: 1216 MHz), 4 GB GDDR5 RAM at 1753 MHz, 256-bit interface width. Operating system: Microsoft® Windows® 7.

All computations described in this chapter, including all measurements of execution time, were carried out on this system.

7.4 Illustrated Results of a Single Optimization and Analysis

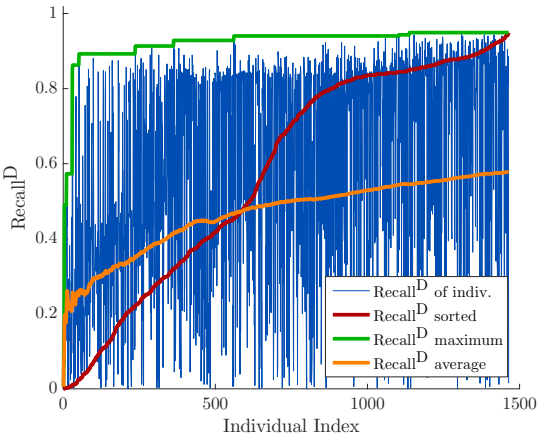
As a starting example, this section gives a complete walkthrough of a single run of the *Optimization* stage: Convergence of objectives and the obtained Pareto front are examined, and trade-offs between objectives are investigated. Finally, the optimized parameters and the classifying model are utilized in the *Application* stage to analyze the real sensor data. Regarding only a *single optimization* enables thorough visualization of all results, providing a visual intuition behind the more abstract results in Section 7.5. The latter then aggregate over a *multitude of optimizations*, conducted for different experiments, setups and folds. In the current section, however, experiment 100 nm HQ, as described in Table 7.1, serves as the example, i.e. one of the more difficult experiments is examined. *SynOpSis* was run with optimization mode *Global 4* and desirability mode *Scalarizing*, as argued for later, in Section 7.5.4.

Objectives during the Optimization Stage

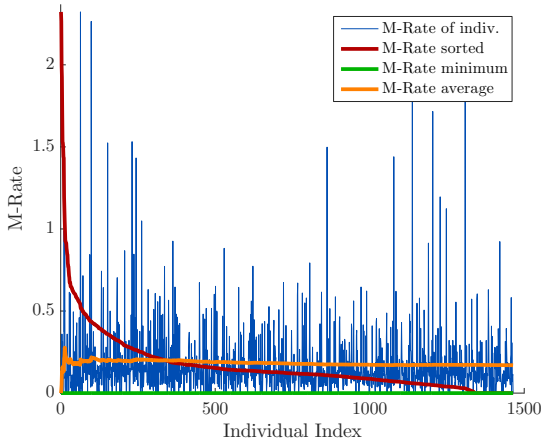
Figure 7.4 plots the development of the four objectives (cf. Section 7.3.1) and of the DI $\Delta(\mathbf{y})$ (cf. Section 7.3.3) over the course of optimization. All measures refer to the training dataset (cf. Section 7.3.4). In (a)–(e), the individuals fulfilling the early cancellation criterion[6] have been removed. Doing so serves to avoid clutter in the plots in (a)–(e). Besides the original function values depicted in blue, these plots also show sorted function values in red, illustrating how values distribute across the codomain. Furthermore, a green curve displays the best values attained so far in an objective, and an orange curve presents the average over the individuals examined thus far.

As can be seen from the green curves in (b)–(d), individuals performing perfectly in terms of M-Rate, Precision$^{\mathrm{C}}$ and Recall$^{\mathrm{C}}$ can be found very quickly. In contrast to that, Recall$^{\mathrm{D}}$

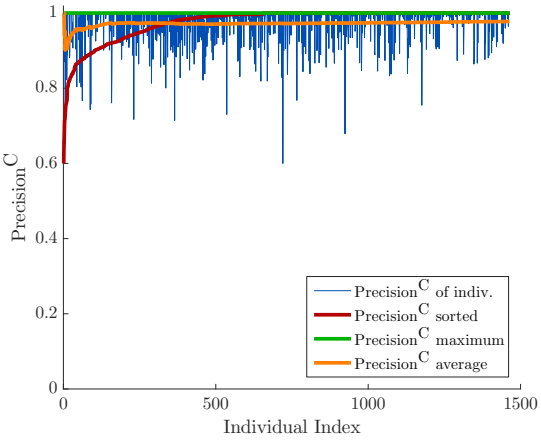
[6] The early cancellation criterion is fulfilled if an individual produces more than $a = 5$ times more detector responses than there are ground truth target patterns in the training set, cf. Section 5.8.



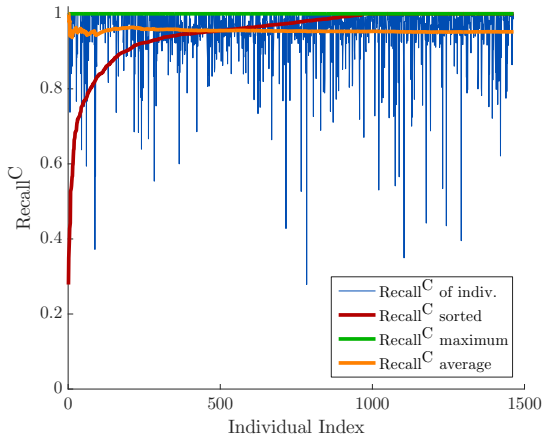
(a) Recall$^{\mathrm{D}}$



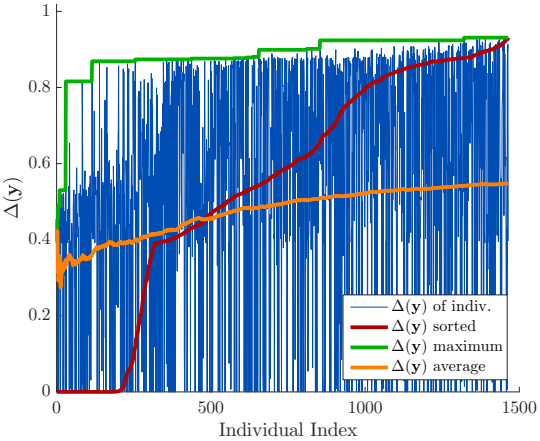
(b) M-Rate



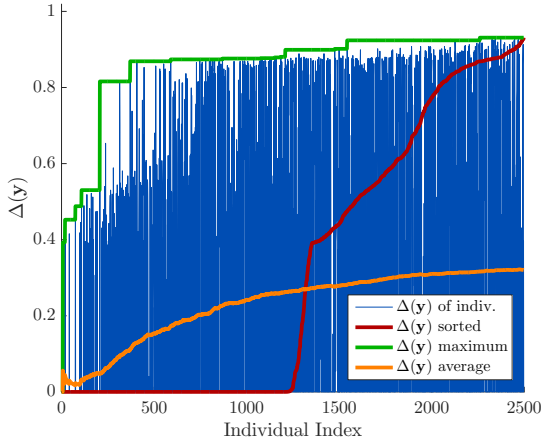
(c) Precision$^{\mathrm{C}}$



(d) Recall$^{\mathrm{C}}$



(e) $\Delta(\mathbf{y})$



(f) $\Delta(\mathbf{y})$ (incl. early cancellations)

Figure 7.4: Development and Convergence of Objectives. The development of objectives and of the Desirability Index (DI) $\Delta(\mathbf{y})$ is shown over the number of evaluated individuals. In (a)–(e), individuals fulfilling the early cancellation criterion were removed to avoid clutter. In contrast, (f) shows *all* individuals, enabling to assess convergence (saturation) of $\Delta(\mathbf{y})$ over all function evaluations. Depending on the favored ratio between $\Delta(\mathbf{y})$ and execution time, the number of individuals can be reduced in future optimizations.

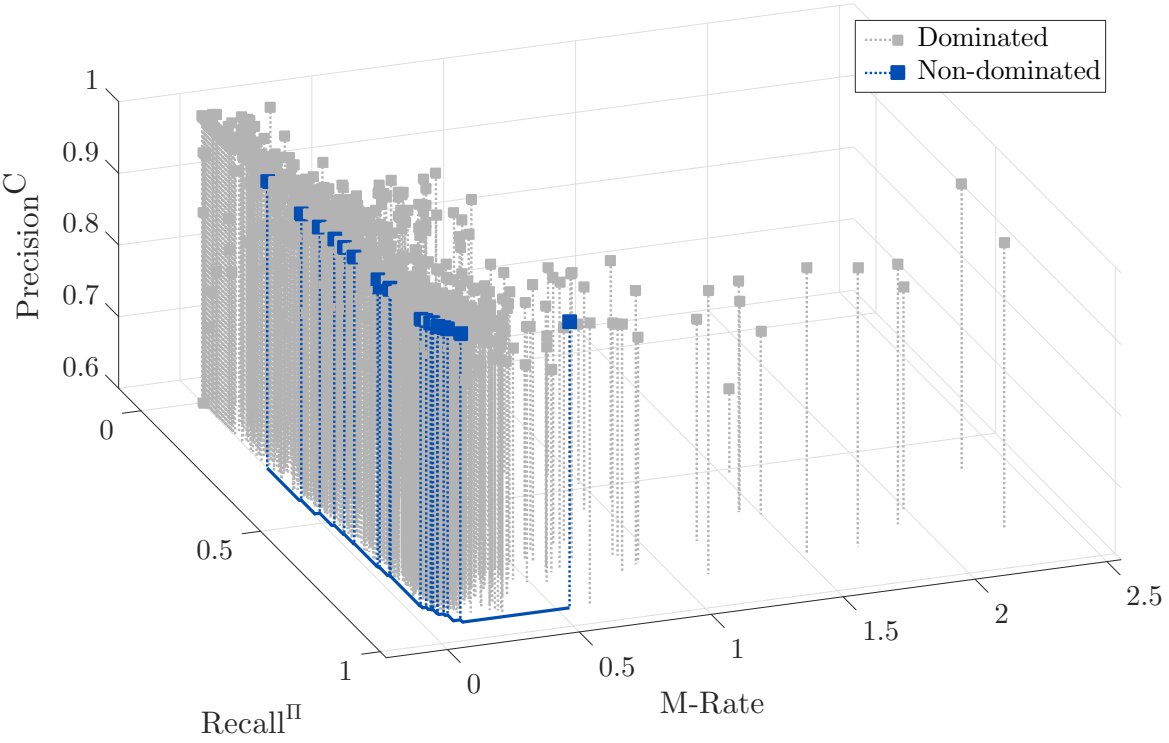


Figure 7.5: Pareto Front, Reduced to 3-D. The Pareto front resulting from the optimization in Figure 7.4 is shown. The dimension of objective space has been reduced to three by multiplying Recall^{D} and Recall^{C} to form the Recall^{Π} axis, thus allowing for this type of visualization. Large gains in Recall^{Π} can be traded off for small increases in M-Rate. All but one individual with M-Rate > 0.15 are dominated, and all non-dominated individuals exhibit $\text{Precision}^{\text{C}} > 0.99$.

in (a) takes more individuals to attain good values, and the perfect value of one is never achieved in this experiment. Hence, if there were no dependencies and thus no trade-offs between objectives, the conducted search could be regarded as a search among individuals with perfect values in M-Rate, $\text{Precision}^{\text{C}}$ and Recall^{C}, made distinguishable by different values in Recall^{D}. However, as e.g. larger Recall^{D} typically results in a larger rate of multiple detections (M-Rate) and increases the difficulty of the classification task (affecting $\text{Precision}^{\text{C}}$ and Recall^{C}), the search identifies good trade-offs between competing objectives.

In order to assess convergence speed and saturation of the DI, (f) repeats (e) including *all* individuals, i.e. also those for which the early cancellation criterion was fulfilled are plotted. As the utilized geometric mean DI attains high values only if all its constituent DFs attain high values (cf. Section 3.8.2), this plot shows how many individuals are needed to find parameter sets that perform well in *all* objectives simultaneously. For example, a DI value of 0.9239 is attained after 1543 individuals have been evaluated, and only a single minor increase in DI is observed after that (DI 0.9310 at individual 2260), cf. green curve in (f). A behavior of saturation similar to that of the DI in (f) was observed in all experiments, i.e. the number of individuals regarded in the *Optimization* stage of *SynOpSis* can be reduced without severe loss in results quality on training data.

Table 7.2: Confusion Matrix of the Detector. The confusion matrix of the detector, as attained on real sensor data from experiment 100 nm HQ, is shown in terms of absolute and relative values. Its structure is analogue to Table 3.1, hence there are no TNs [WHS+12; SLN+09]. As discussed in Section 3.5.2, the TP entry, which was 157, has been replaced with $\widehat{\mathrm{TP}} = 153$, i.e. it was cleansed from four repeated detector responses. The attained values of objectives and measures are: $\mathrm{Precision}^{\mathrm{D}} = 0.25758$, $\mathrm{Recall}^{\mathrm{D}} = 0.78462$, M-Rate = 0.02614.

		Ground Truth	
		Target	Non-Target
Detector	Response	$\widehat{\mathrm{TP}} = 153$ (24%)	FP = 441 (69%)
	No Response	FN = 42 (7%)	TN = 0

Pareto Front

Figure 7.5 shows the Pareto front obtained after the optimization in Figure 7.4. In order to make the four objectives of optimization mode *Global 4* eligible for this type of visualization, $\mathrm{Recall}^{\mathrm{D}}$ and $\mathrm{Recall}^{\mathrm{C}}$ have been multiplied to become Recall^{Π}, as argued for in Section 7.3.2 in the context of optimization mode *Global 3*. Hence the dimension of objective space is reduced to three. The Pareto front was recomputed with respect to this reduced objective space, which decreases the number of non-dominated individuals, compared to the original four-dimensional space.

The Recall^{Π} dimension in Figure 7.5 exhibits the largest spread: Points disperse rather uniformly along the entire Recall^{Π} axis. Concerning the other dimensions, they tend to focus in a region of high $\mathrm{Precision}^{\mathrm{C}}$ and low M-Rate. The lowest observed value in $\mathrm{Precision}^{\mathrm{C}}$ across all individuals is about 0.6, while in the non-dominated individuals $\mathrm{Precision}^{\mathrm{C}}$ is always above 0.99. The vast majority of points exhibits M-Rate below 0.5. For increasing Recall^{Π}, dominated points with higher M-Rate occur, but their density is comparably low.

Concerning the trade-offs between objectives, the following observations can be made in Figure 7.5: Non-dominated points occur primarily along the Recall^{Π} axis, with large gains in this dimension trading off for low increases in M-Rate. Only the very last gain in Recall^{Π} is unduly expensive in terms of M-Rate: The rightmost non-dominated point has coordinates $(0.93167, 0.54747, 1)$ while the neighbor with next-lower Recall^{Π} has coordinates $(0.92234, 0.14194, 1)$: Reducing M-Rate by 0.4055, i.e. $\approx -40\%$ in excess detector responses, comes at the cost of 0.00933 in Recall^{Π}, i.e. $\approx -1\%$ in correct primary responses. The non-dominated point with lowest Recall^{Π} is $(0.27164, 0, 1)$, i.e. accepting an increase of 0.14194 in M-Rate yields an increase of 0.65070 in Recall^{Π}. All other non-dominated points occur closely to a line between these two points.

Analysis Result for Real Sensor Data

Results of running the *Application* stage of *SynOpSis* (bottom part of Figure 3.2) on the real sensor data, using the outputs of the previous optimization are reported now. Ground truth for experiment 100 nm HQ was obtained via manual segmentation by human experts, and results quality was evaluated via the matching and labeling procedure from Section 5.8.

Table 7.2 displays the confusion matrix of the detector, with details discussed and objectives reported in the table caption. This also holds for Table 7.3, which does the same for the classifier.

Table 7.3: Confusion Matrix of the Classifier. The classifier confusion matrix as attained for the detection results from Table 7.2 is shown. Its structure is analogue to Table 3.2. The 'Positive'-column corresponds to the TPs of the detector and sums to 157 because the classifier also receives the four repeated detector responses, cf. caption of Table 7.2. The 'Negative'-column corresponds to the FPs of the detector. 94% of the detector responses are classified correctly. The attained values of objectives and measures are: $\text{Precision}^{\text{C}}$ = 0.85119, Recall^{C} = 0.91083, D-Rate = −0.13846.

		Ground Truth	
		Positive (TPs of detector)	Negative (FPs of detector)
Classifier	Positive	TP = 143 (24%)	FP = 25 (4%)
	Negative	FN = 14 (2%)	TN = 416 (70%)

As a unified visual depiction of the results tabulated across the two confusion matrices, Figure 7.6 gives a 3-D perspective view of the analysis results in the spatiotemporal coordinate system of *PAMONO* sensor data. In this figure, red color indicates the manually segmented ground truth polygons, while blue color is used for polygons output by the detector. Saturated color indicates ground truth or detector polygons for which a match of the respective other type exists. Black lines connect primary matches, while green lines connect repeated detections to the respective ground truth polygon. Unsaturated color is used for ground truth or detector polygons without a matching polygon of the respective other type. In particular, unsaturated red polygons are ground truth polygons missed by the detector, which *can not* be recovered by the classifier (False Negatives (FNs) of the detector). In contrast to that, unsaturated blue polygons are FPs of the detector which *can* be recovered by the classifier. Cases where the classifier was not able to do so are marked with a black cross, i.e. these crosses mark FPs of the classifier. Conversely, black circles indicate its FNs, i.e. cases where the classifier sorted out a correctly detected nano-object. Any unmarked detector polygon (blue) was classified correctly.

7.5 Optimization Options and Final Analysis Results

Section 7.4 provided a visual understanding of the results produced by the *Optimization* stage and the *Application* stage of *SynOpSis* within a single run. For conveying the bigger picture, this section provides more abstract results, aggregating over a multitude of optimizations to provide a more representative overview of the capabilities of *SynOpSis*. The results of 162 executions of the *Optimization* stage serve as the basis for aggregation within this section. This number is composed as follows: Six different *PAMONO* experiments, as listed in Table 7.1, are used as the input data, and each one undergoes optimization for all combinations of the three optimization modes from Section 7.3.2 with the three desirability modes from Section 7.3.3. Each of these optimizations is repeated in three folds of cross-validation, as described in Section 7.3.4, resulting in a total number of 162 executions of the *Optimization* stage. All optimization results, i.e. detector parameters and the final classifying models, learned as according to Section 7.3.5, are used in the *Application* stage to analyze the real sensor data. Complementing these results, the *Application* stage is furthermore executed on the training, validation and test set, using a classifying model learned solely from the training dataset. This serves to assess performance within a purely synthetic, cross-validated context.

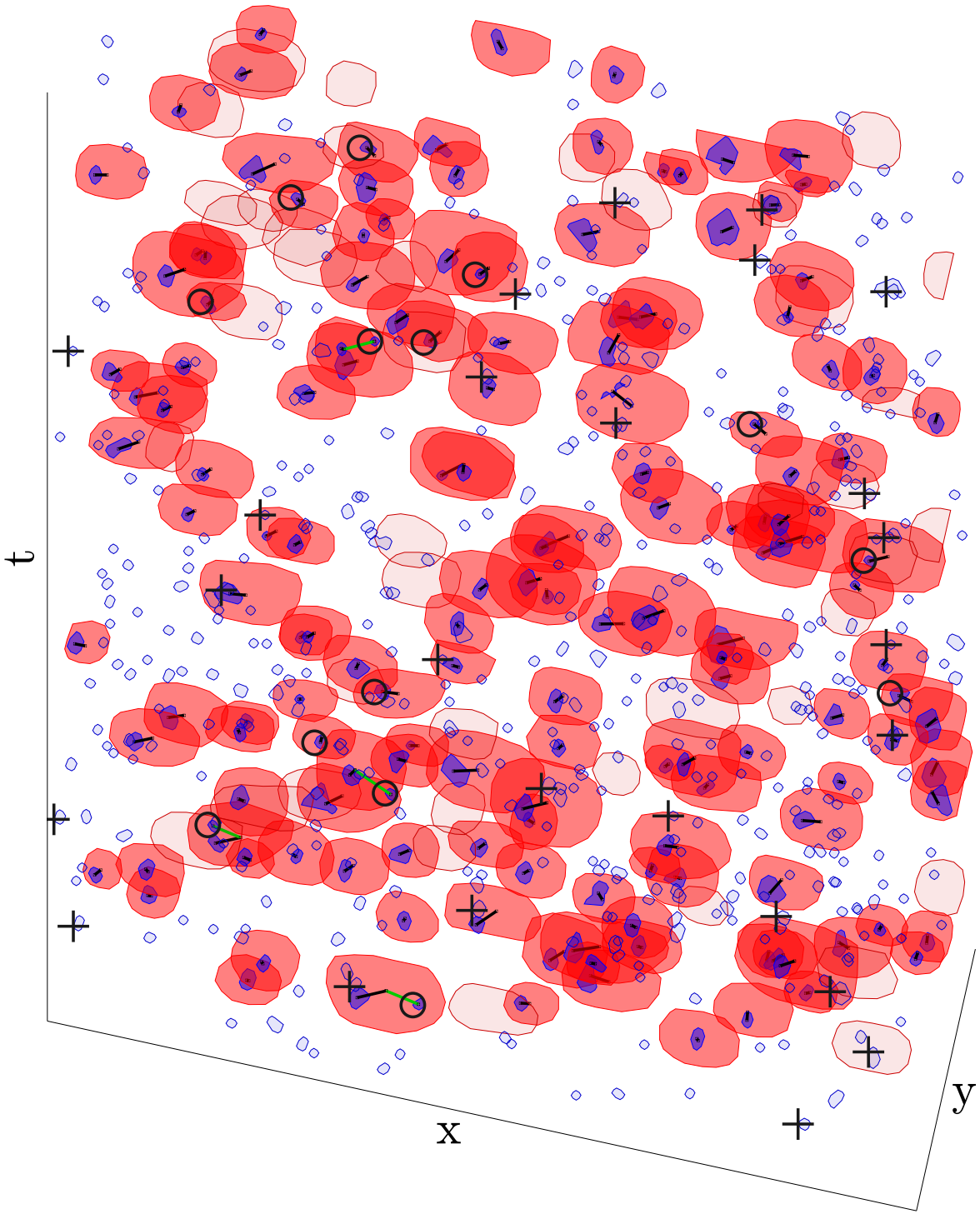


Figure 7.6: 3-D Visualization of Analysis Results for Real Sensor Data. This 3-D perspective plot in the spatiotemporal coordinate system of *PAMONO* sensor data visualizes the content of both, the detector and classifier confusion matrix in Tables 7.2 and 7.3, respectively. A detailed description of the semantics behind the employed colors, saturations and symbols is given in the text.

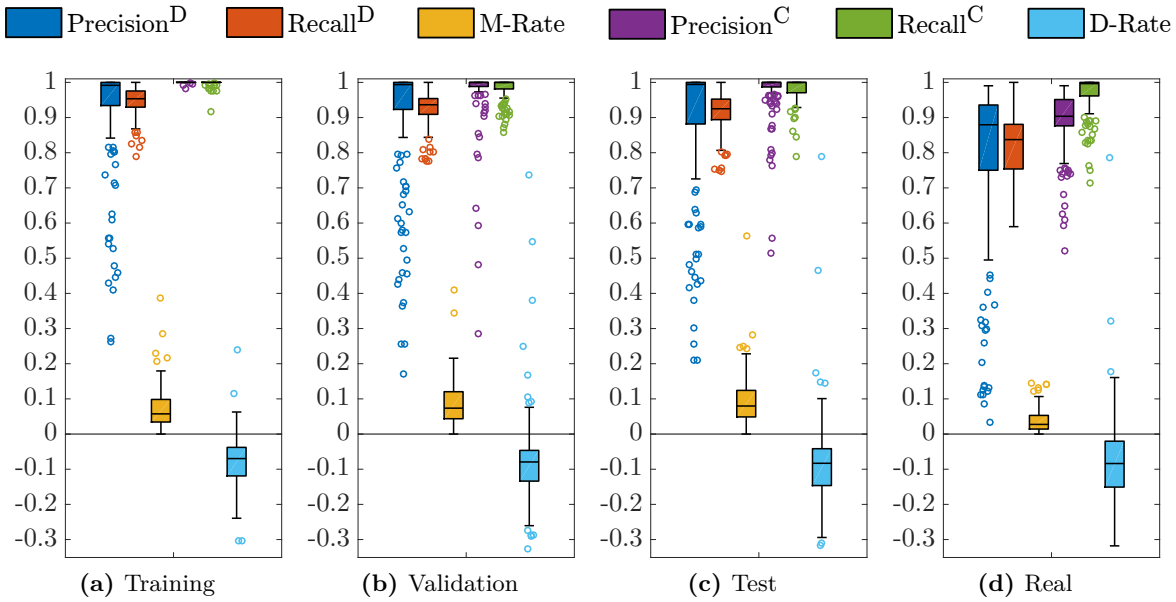


Figure 7.7: Objectives and Measures over Datasets. Box plots summarizing the distributions of objectives and measures (cf. legend above the figure) attained in 162 optimizations are shown. This number results from all combinations of six experiments, three optimization modes and three desirability modes across three folds of cross-validation. Each of the figures (a)–(d) summarizes these 162 points for another dataset. A detailed comparison between analyses of the synthetic datasets in (a)–(c) to the real sensor data in (d) is given in the text.

The purpose of this section is to firstly identify, which combination of optimization and desirability mode works best on the given *PAMONO* experiments. This is done in a coarse-to-fine fashion, starting with Section 7.5.1 summarizing the results from all 162 optimizations. Sections 7.5.2 and 7.5.3 separate these results by optimization and desirability modes, respectively, while Section 7.5.4 regards all of their nine combinations, followed by identifying the setup of *SynOpSis* that is used throughout the remainder if this section. This remainder serves the second purpose, which consists in reporting the final analysis results obtained for the six examined *PAMONO* experiments. Having chosen a single setup of *SynOpSis*, results are no longer aggregated over different modes, but reported individually per experiment, thus providing insight into finer variations of analysis quality between experiments. Section 7.5.5 does so by evaluating analysis performance in terms of the objectives and measures from Section 7.3.1, in order to validate the quality of *SynOpSis* results, and to identify the limits of the method. Section 7.5.6 evaluates the quality of the performance estimates delivered by *SynOpSis*. Then, Section 7.5.7 investigates specificity of the obtained results by running the *Application* stage on real sensor data not containing any nano-objects. Finally, Section 7.5.8 reports execution times in terms of the attained frame rate of real-time analysis, evaluation cost per individual and overall optimization time.

7.5.1 Results Over Datasets

As a first impression of the analysis quality attained by applying *SynOpSis* to the *PAMONO* experiments in Table 7.1, Figure 7.7 shows four box plots [MTL78], each summarizing the results of all 162 conducted optimizations, as described in the first paragraph of Section 7.5. The first three box plots relate to synthetic datasets (training (a), validation (b), test (c)),

and the utilized classifying model was learned solely from the training dataset. The fourth box plot relates to the real sensor data (d), and the utilized classifying model was learned from the union of all synthetic datasets, cf. Section 7.3.5. Each box plot summarizes the distributions of the objectives and measures from Section 7.3.1 in terms of their quartiles (box), the two most extreme points not deemed outliers (whiskers), and an individual marker for each outlier (circles). Outliers are defined as follows: Let q_1 and q_3 denote the first and third quartile, respectively, making $r = q_3 - q_1$ the Interquartile Range (IQR). Then any point with a value smaller than $q_1 - 1.5r$ or larger than $q_3 + 1.5r$ is considered an outlier.

A mapping between box colors and names of measures and objectives is given in the legend at the top of the figure, in the same order as they appear in each box plot. Note that the leftmost and rightmost measures, i.e. Precision$^{\mathrm{D}}$ and D-Rate merely serve to report properties of the analysis, but are not optimized, for the reasons discussed in Section 7.3.1. Precision$^{\mathrm{D}}$ measures class balance for the classifier, thus lower values indicate increased necessity of a well-performing classifying model. D-Rate is a summary measure of analysis quality, quantifying the relative deviation between the reported and the actual number of nano-objects. In between these two measures, the distributions of the four optimized objectives Recall$^{\mathrm{D}}$, M-Rate, Precision$^{\mathrm{C}}$ and Recall$^{\mathrm{C}}$, as explained in Section 7.3.1, are characterized. This scheme of displaying the results of optimization is adhered to throughout the entire section, hence this introduction can be used for reference in interpreting later results.

In particular, at this coarsest level regarding all 162 conducted optimizations, Figure 7.7 provides a summary of how measures and objectives behave over the four types of considered datasets, while aggregating over all other aspects, i.e. over experiment, optimization and desirability modes and folds of cross-validation. Most objectives deteriorate from left to right, i.e. for increasing difficulty of the analysis task, where difficulty arises from decreasing fit of the respective employed data. This can be seen particularly with regard to Precision$^{\mathrm{C}}$ and Recall$^{\mathrm{C}}$: Both attain median value one and IQR zero on training data (a), suggesting that the classification errors incurred on validation (b) and test data (c) are primarily due to differences in the concepts of the two classes between training and the other datasets. While deterioration from training (a) over validation (b) to test (c) is mild, the real sensor data (d) is affected more strongly: Median Recall$^{\mathrm{D}}$ drops from 0.9250 on the test set to 0.8373 on real data, and the IQR widens. While Recall$^{\mathrm{C}}$ is nearly unaffected, median Precision$^{\mathrm{C}}$ drops from 1 to 0.9036, and the increased number of FPs in the classifier partly compensate for the increase in FNs in the detector, resulting in a nearly unchanged median D-Rate of -0.0838, which illustrates why D-Rate is not suitable as an objective, cf. Section 7.3.1. M-Rate is the only objective that improves for real data, which can be explained by the fact that synthetic data contains on average approximately three times more nano-objects than real data (cf. Section 7.3.4), thus delivering more occasions for repeated detections. Median Precision$^{\mathrm{D}}$ drops from 0.9942 on the test set to 0.8795 on real data, indicating an increased importance of having a good classifying model for the latter. Note that low Precision$^{\mathrm{D}}$ provides no evidence of low analysis quality, and its large numbers of outliers and wide IQR, as observed throughout this section, merely point out a high diversity in class balance for the classifier.

7.5.2 Results Over Optimization Modes

Figure 7.8 groups all data shown in Figure 7.7 by optimization modes *Global 4*, *Global 3* and *Sequential*, as defined by Terminology 7.2 in Section 7.3.2. Hence, for each dataset,

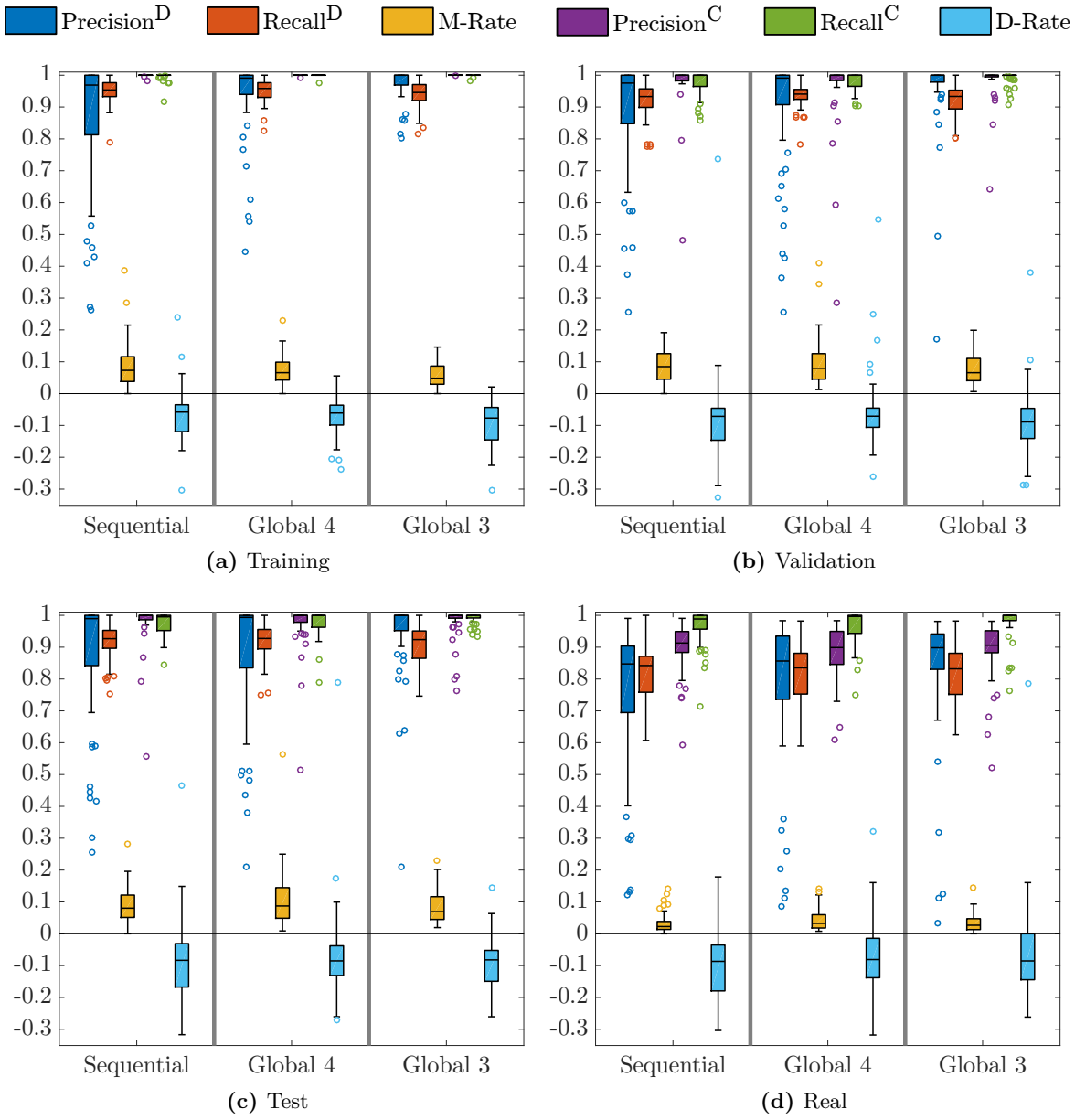


Figure 7.8: Objectives and Measures over Optimization Modes. Box plots grouping the data from Figure 7.7 by the three optimization modes defined in Section 7.3.2 are shown. Optimization mode *Global 4* is superior in terms of D-Summary (Equation (7.6)) on real sensor data (d). However, differences in objectives and measures are small over optimization modes.

each group summarizes the distribution of the 54 points obtained in the optimization mode corresponding to the group name, while aggregating over experiment, desirability mode and fold. Deterioration comparable to Figure 7.7 can be observed over datasets (a)–(d).

In this section, however, the focus lies on identifying the optimization mode that performs best in analyzing real data, which is why Figure 7.8d is examined more closely: Given this data, the choice of a certain optimization mode is to be taken with respect to *empirical distributions* in two measures and four objectives. One way of facilitating this choice is to regard medians (or any other summary statistic) of these distributions only, resulting in a

six-dimensional space containing three individuals. Doing so for the medians in (d) yields three non-dominated individuals, hence Pareto dominance can not readily be applied in taking the choice. One solution would be to compute a DI with respect to suitable DFs. However, this ignores the spread around the median, unless it is incorporated into the DI, requiring additional DFs to be defined. A simpler, empirically driven heuristic can be derived with respect to the summary measure D-Rate: From the perspective of practical application of *SynOpSis* in *PAMONO* data analysis, the decisive criterion of performance is the deviation between the reported and the actual number of nano-objects in real data, as measured by D-Rate. Besides a median of D-Rate that should be close to zero, its spread should be as small as possible, such that good D-Rate is achieved not only for the median. One way of formalizing this is D-Rate Summary (D-Summary), defined as

$$\text{D-Summary} = |\text{Median}(\text{D-Rate})| + \text{IQR}(\text{D-Rate}), \tag{7.6}$$

where the functions Median(∘) and IQR(∘) yield the median and Interquartile Range over the values observed in the empirical distributions of D-Rate. Minimization of D-Summary realizes the described heuristic. Doing so over optimization modes on the real data in (d) identifies *Global 4* (Median(D-Rate) = −0.0810, IQR(D-Rate) = 0.1238) as superior to *Sequential* (Median(D-Rate) = −0.0867, IQR(D-Rate) = 0.1443) and *Global 3* (Median(D-Rate) = −0.0852, IQR(D-Rate) = 0.1443): Median and IQR vote for *Global 4*.

Note that taking this choice with respect to a measure derived from D-Rate may encounter the issue that good values in D-Rate can be attained even in case of very bad values in all four objectives, as long as the errors cancel each other out (cf. Section 7.3.1). However, this choice is taken with respect to optimization *results*. Hence, for the points in Figure 7.8d it is already ensured that they provide good values in Recall^{D}, M-Rate, $\text{Precision}^{\text{C}}$ and Recall^{C}.

Despite the data in Figure 7.8d voting for optimization mode *Global 4* in terms of D-Summary, *Global 4* is beaten by *Sequential* and *Global 3* in three of the four optimized objectives each. This superiority, however, is by a small margin, as differences between optimization modes are small in all objectives and measures. Therefore, the choice of an optimization mode is re-examined in combination with the choice of a desirability mode. This is done in Section 7.5.4, after Section 7.5.3 examined desirability mode alone.

7.5.3 Results Over Desirability Modes

Analogous to how the previous section examined the behavior of measures and objectives over different choices of optimization mode, this section does so for the three desirability modes *Disabled*, *Multi-Objective* and *Scalarizing* as defined by Terminology 7.3 in Section 7.3.3. Figure 7.9 shows the corresponding results, grouping the data from Figure 7.7 by desirability modes, while aggregating over experiments, optimization modes and folds of cross-validation. As observed before, deterioration in objectives and measures is small over the synthetic datasets in (a)–(c) and more pronounced for the real sensor data in (d). Examining D-Summary from Equation (7.6) for the real sensor data yields superiority of desirability mode *Scalarizing* (Median(D-Rate) = −0.0680, IQR(D-Rate) = 0.1367) over *Disabled* (Median(D-Rate) = −0.0979, IQR(D-Rate) = 0.1117) and *Multi-Objective* (Median(D-Rate) = −0.0897, IQR(D-Rate) = 0.1636). Differences in all measures and objectives are small over desirability modes, so like with optimization modes in the previous section, these results are to be viewed with caution.

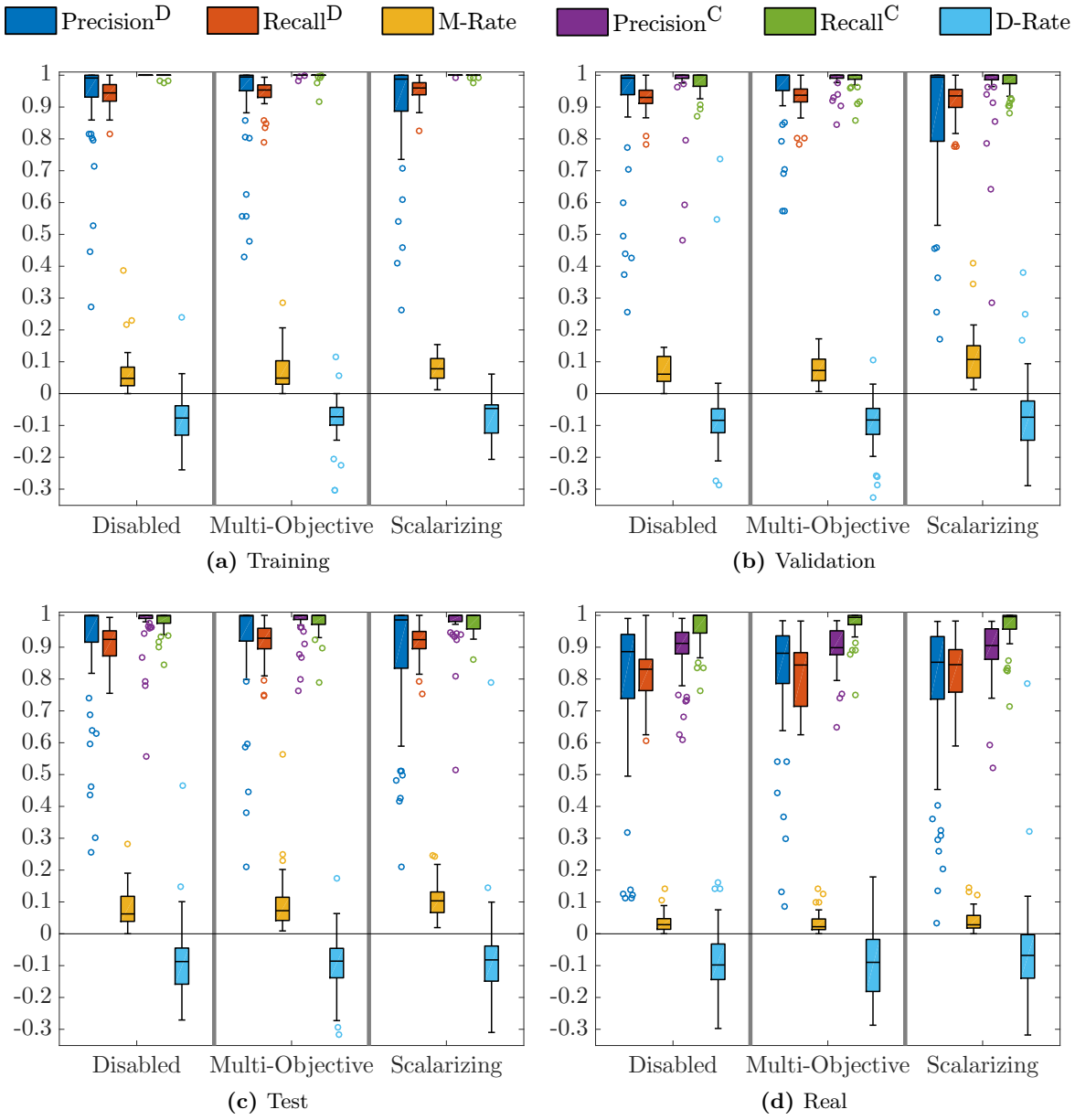


Figure 7.9: Objectives and Measures over Desirability Modes. Box plots grouping the data from Figure 7.7 by the three desirability modes defined in Section 7.3.3 are shown. Desirability mode *Scalarizing* is superior in terms of D-Summary (Equation (7.6)) on real sensor data (d). However, like with optimization modes in Figure 7.8, differences in objectives and measures are small over desirability modes.

7.5.4 Choice of Optimization and Desirability Mode

Sections 7.5.2 and 7.5.3 regarded optimization and desirability modes in *isolation* while aggregating over the respective other. Now this section groups the data from Figure 7.7 more finely, resolving all nine *combinations* of optimization and desirability modes.

Figure 7.10 does so in terms of box plots, showing objectives and measures as attained for the real sensor data, where each group aggregates over experiments and folds only. As a first observation, the decreased amount of aggregation resulting from the finer grouping

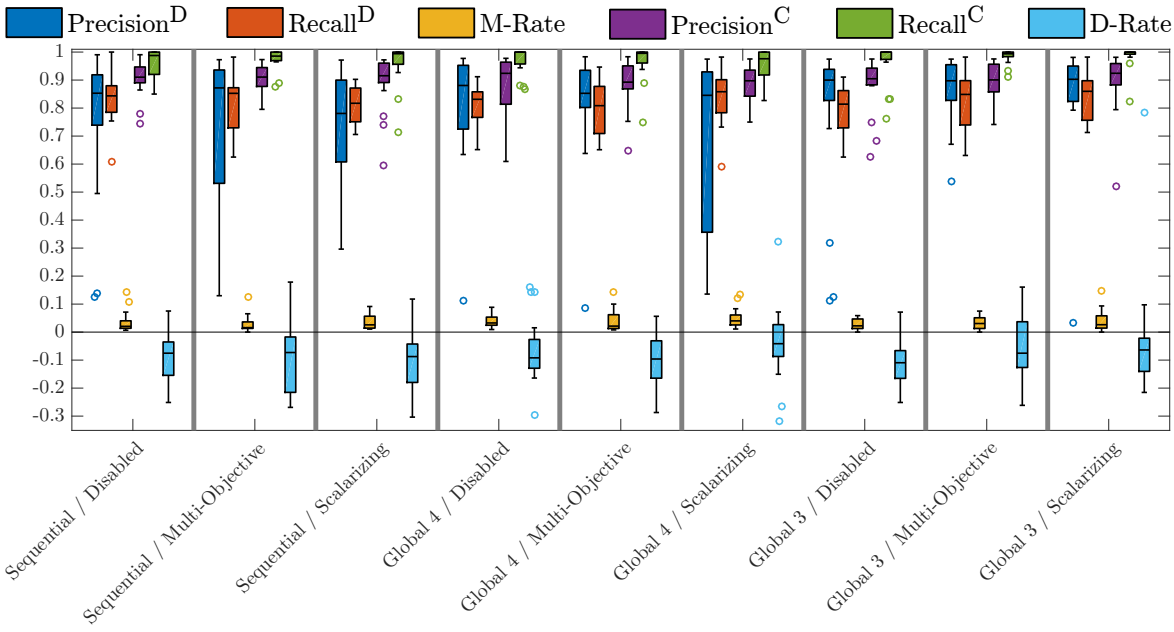


Figure 7.10: Objectives and Measures over Optimization and Desirability Modes. Box plots grouping the data from Figure 7.7 by all combinations of optimization and desirability modes (cf. Sections 7.3.2 and 7.3.3) are shown for real sensor data only. Combining optimization mode *Global 4* with desirability mode *Scalarizing* is superior in terms of D-Summary (Equation (7.6)).

does not increase definiteness of the votes in terms of D-Summary from Equation (7.6): Differences in D-Summary (and in all measures and objectives except for Precision^{D}) remain small. However, optimization mode *Global 4* and desirability mode *Scalarizing* which minimized D-Summary in isolation also do so in combination (Median(D-Rate) = −0.0414, IQR(D-Rate) = 0.1138). The second-best combination in terms of D-Summary is *Global 3*, *Scalarizing* (Median(D-Rate) = −0.0638, IQR(D-Rate) = 0.1186), while the worst one is *Sequential*, *Multi-Objective* (Median(D-Rate) = −0.0729, IQR(D-Rate) = 0.1978). The small difference between the *Global 4*, *Scalarizing* combination and the second-best indicates that this vote in terms of D-Summary is not clear enough to serve as the sole justification for running *SynOpSis* with this combination. A definitive decision would require more experiments. Furthermore, minimizing D-Summary is not the only conceivable way of choosing the combination of optimization and desirability mode, cf. Section 7.5.2. So the point to be made in the following paragraph is not that the decision taken there is the *best* or the *only* reasonable decision, but merely that it is *a* decision, obtained in a systematic fashion, i.e. with respect to a computable criterion. Its primary benefit lies in having a decision, allowing for a more detailed analysis of the results that were obtained adhering to it.

The small differences in the selected criterion D-Summary and the small number of experiments, as mentioned in the previous paragraph, put this decision on a thin basis. However, small differences also mean that among the top-ranking combinations, the concrete decision is of negligible importance, if suboptimality on the order of these small differences can be tolerated. Therefore, the final decision is taken as follows: In order to remove the optimization and desirability mode variables from the data in Figure 7.7, only the *Global 4*, *Scalarizing* combination is regarded in the subsequent evaluations. As a consequence, from the 162 optimizations displayed in Figure 7.7, only 18 are considered further, which are due

to the six experiments in three folds of cross-validation. This allows Sections 7.5.5 to 7.5.8 to examine measures and objectives on the per-experiment level.

Before this is done, the consequences of setting optimization and desirability mode to *Global 4* and *Scalarizing*, respectively, are examined more closely: This combination means that the four objectives from Section 7.3.1 undergo global optimization, i.e. detector and classifier parameters are optimized simultaneously. Global optimization allows for feedback from the classifier to detector parameter optimization via the path that parameters producing detector results that are hard to classify are easily dominated in classifier objectives. The employed optimization, however, is single-objective because the four objectives are summarized by computing their geometric mean DI, which aims at focusing the search on the desirable part of objective space. The reduction from a multi-objective to a single-objective optimization makes the problem amenable to single-objective optimization algorithms, which is, however, not in the scope of this thesis. As the DI-based approach does not aim at covering an entire Pareto front in objective space, usually fewer evaluations of points in parameter space suffice (cf. the convergence of the DI in Figure 7.4f). Note, however, that scalarization via the DI should only be applied if the Pareto fronts are not of interest, e.g. in optimizations conducted for everyday lab practice, where the goal is to find a single point in the desirable region of objective space with as few function evaluations as possible. In contrast to that, if the goal is to investigate how the different objectives trade off with each other, multi-objective optimization with desirability mode *Disabled* is recommended because of its tendency to produce solutions that spread more diversely over objective space.

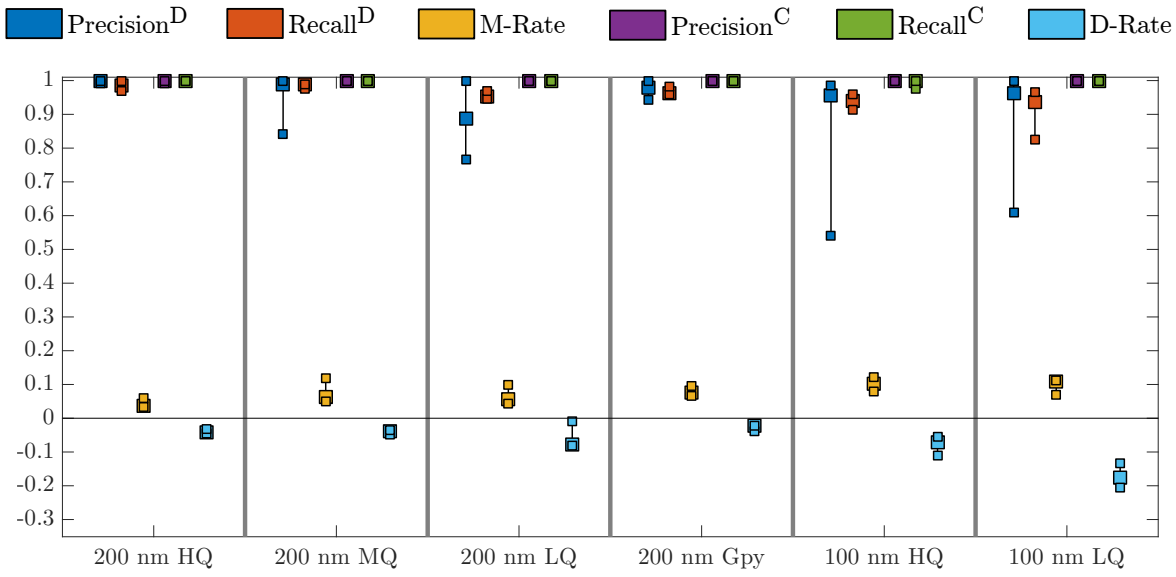
7.5.5 Final Analysis Results Over Experiments

With optimization and desirability modes fixed to *Global 4* and *Scalarizing*, respectively, as determined in Section 7.5.4, this section reports the finally obtained analysis results for the six examined *PAMONO* experiments from Table 7.1. These results are no longer aggregated but reported individually per experiment, enabling to assess the impact of varying nano-object sizes and SNRs in *PAMONO* experiments on analysis quality of *SynOpSis*. This serves to validate *SynOpSis* as an approach to analyzing *PAMONO* sensor data, and to identify its limitations.

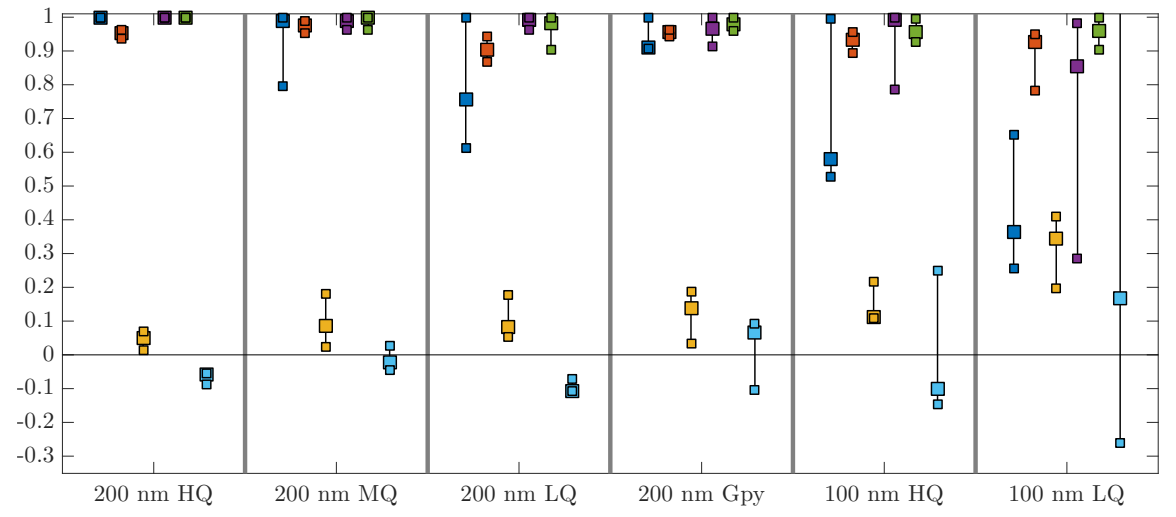
Figures 7.11 and 7.12 visualize measures and objectives, similarly to the box plots in the four previous sections. Results are grouped by the names of the experiments from Table 7.1, and for each measured quantity three values are indicated by squares. These three values correspond to the three folds of cross-validation (cf. Section 7.3.4). Their median is drawn slightly larger than the higher and lower value.

Training

With regard to the training datasets in Figure 7.11a, one can see that Table 7.1 lists the experiments approximately in the order of increasing difficulty of analysis: Recall^{D} tends to decrease, M-Rate increases, and the deviation of D-Rate from zero increases in the negative direction, i.e. the number of nano-objects is underestimated with increasing severity. $\text{Precision}^{\text{C}}$ and Recall^{C} are close to one in all cases. The decrease in analysis quality observed from top to bottom of Table 7.1, respectively from left to right of Figure 7.11a is in agreement with the fact that the SNRs listed in Table 7.1 and visualized in Figures 7.1 and 7.2, solely capture step function height versus noise variability. In particular, they neither measure

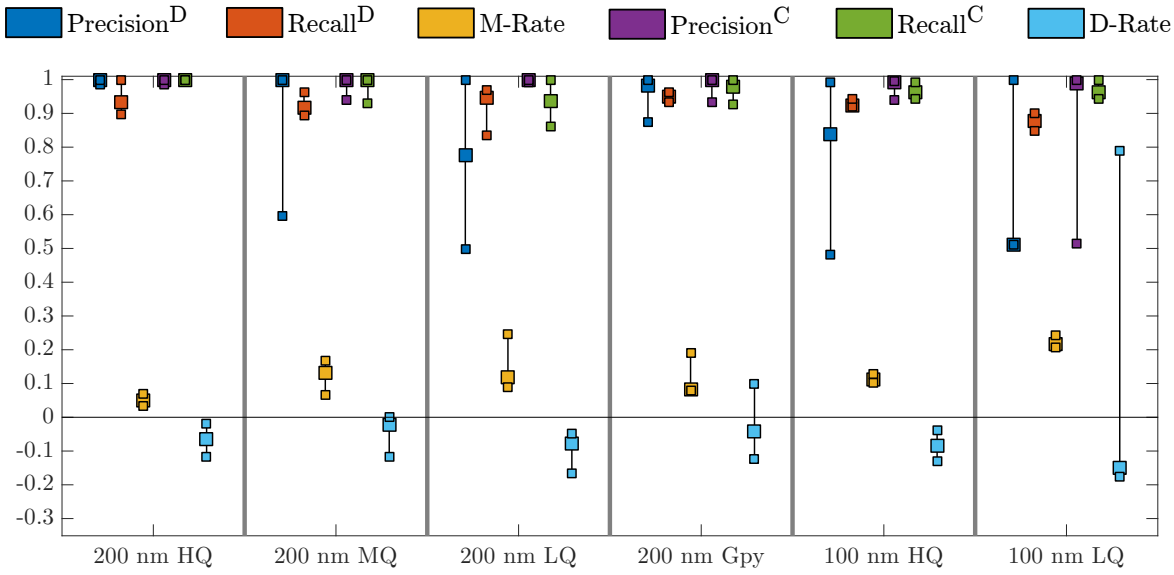


(a) Training

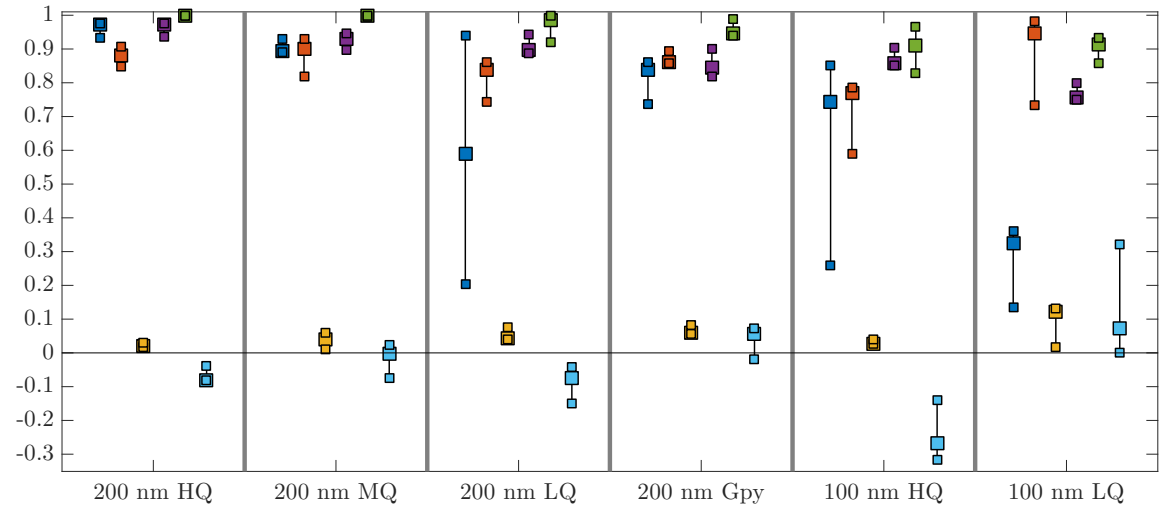


(b) Validation

Figure 7.11: Objectives and Measures over Experiments – Training and Validation Datasets. Analysis results for optimization mode *Global 4* combined with desirability mode *Scalarizing* are shown, as chosen in Section 7.5.4. Results are resolved by experiment, and each of the three squares represents one of the three folds of cross-validation, with a larger square indicating the median. (a) shows results on the training dataset, while (b) does the same for the validation dataset. The truncated value in the D-Rate measure of experiment 100 nm LQ is 2.80165. Figure 7.12 shows the same results for the test and real sensor dataset.



(a) Test



(b) Real

Figure 7.12: Objectives and Measures over Experiments – Test and Real Datasets. This is a continuation of Figure 7.11, also showing analysis results for optimization mode *Global 4* combined with desirability mode *Scalarizing*. Here, results for the test (a) and real sensor datasets (b) are shown. Results are resolved by experiment, and each of the three squares represents one of the three folds of cross-validation, with a larger square indicating the median.

the amount and severity of artifacts appearing in the images, nor do they capture similarity of artifacts and actual nano-object adhesions. Hence the approximate order of increasing analysis difficulty in Table 7.1 is not fully reflected in its SNR-columns. As an example, the 200 nm Gpy experiment has the highest minimum, median and maximum SNR, but its analysis is aggravated by many large-scale artifacts with amplitudes similar to nano-object adhesions, cf. Figure 2.4 for an image from this experiment after background elimination. Despite exhibiting the highest SNRs, analysis results for experiment 200 nm Gpy are not better than for the other experiments.

Validation and Test

As deterioration of measures and objectives from the training to the validation and test datasets is similar, only the test dataset is discussed here, which is displayed in Figure 7.12a: Compared to the training dataset, Recall^{D} generally drops, with the lowest value of 0.8487 attained in experiment 100 nm LQ, and the highest value of 1 in 200 nm HQ. M-Rate increases, while $\text{Precision}^{\text{C}}$ and Recall^{C} nearly retain their high values, except for one severe drop in $\text{Precision}^{\text{C}}$, entailing a large increase in D-Rate on experiment 100 nm LQ, thus hinting at the limits of method. D-Rate in the other experiments is comparable to its values in training, which in combination means that the decreased Recall^{D} is obfuscated primarily by increased M-Rate and only secondarily by the slightly decreased $\text{Precision}^{\text{C}}$. Having chosen optimization and desirability modes with respect to D-Rate, as was done in Section 7.5.4, promotes this phenomenon.

Real Sensor Data

As the test datasets are used to estimate the performance to be expected on the real sensor data (cf. also Section 7.5.6), the differences between test and real datasets, both displayed in Figure 7.12, will be examined now. A general drop in Recall^{D} can be observed, except for the 100 nm LQ experiments, where two folds produced higher Recall^{D} on real than on test data. This comes at the cost of a large decrease in $\text{Precision}^{\text{D}}$ for this experiment, i.e. for the real data, the detector yields more FP responses. However, these can mostly be compensated for by the classifier, resulting in D-Rate values between zero and 0.3214 for the most difficult-to-analyze experiment 100 nm LQ. Comparing the higher quality experiment 100 nm HQ to experiment 100 nm LQ, the same transition from under- to overestimation of nano-object counts as on the validation and test set can be observed in D-Rate, albeit being less pronounced on the real data. Experiment 100 nm HQ exhibits the largest underestimation of nano-object counts, with D-Rate ranging from −0.3179 to −0.1385, which is primarily due to low Recall^{D}, ranging from 0.5897 to 0.7846. On the 200 nm experiments, D-Rate values between −0.1505 and 0.07179 are observed. Regarding the test set objectives as estimates for the values to be expected on real data, M-Rate is estimated to be worse than it actually is on real data. $\text{Precision}^{\text{C}}$ and Recall^{C} are overestimated in most cases, with $\text{Precision}^{\text{C}}$ deteriorating more strongly between test and real than Recall^{C}.

Limits of the Method

The transition from under- to overestimation of nano-object counts observed for validation, test and real sensor data between experiments 100 nm HQ and 100 nm LQ can be interpreted as indicating a limit in terms of input quality for *SynOpSis*. High quality sensor data with 100 nm nano-objects behaves comparably to 200 nm experiments: The number of nano-objects is underestimated, and the severity of underestimation increases, which is in agreement with the linear relationship between nano-object size and signal amplitude impeding nano-object detection [ZKG+10]. In contrast to that, low quality sensor data with 100 nm nano-objects results in an overestimation of the number of nano-objects. This can be explained by looking at the SNRs of experiment 100 nm LQ in Table 7.1: The minimum SNR is 0.82224, while the median SNR is 1.24673, which means that half of the nano-objects in experiment 100 nm LQ exhibit an SNR from this range. For comparison, all algorithms surveyed in [CWG01]

break down for SNRs approaching four, while the best of those surveyed in [SLN+09] do so for SNRs approaching two. *SynOpSis* achieves values of Recall^{D} between 0.7321 and 0.9821 in this experiment, and this high sensitivity entails the risk of many FP responses of the detector and thus low $\text{Precision}^{\text{D}}$. This low $\text{Precision}^{\text{D}}$ can in parts be contained by the classifier as can be seen from Figure 7.12b. However, the classification task is impeded because low SNRs of nano-objects can increase their feature-space similarity to artifacts, resulting in lower $\text{Precision}^{\text{C}}$. Hence the low $\text{Precision}^{\text{D}}$ can not be completely compensated for by the classifier, and the result is an overestimation of the number of nano-objects. The transition from high to low quality gold layers in the context of 100 nm nano-objects is a point where the combination of $\text{Precision}^{\text{D}}$ and $\text{Precision}^{\text{C}}$ drops below the critical level between underestimation on experiment 100 nm HQ and overestimation[7] on experiment 100 nm LQ. Hence, to be on the safe side, only high quality gold layers should be used in the context of 100 nm nano-objects.

Stability of Optimization Results over Folds

Stability of optimization results over the three folds of cross-validation can be assessed by examining the variation of the four objectives Recall^{D}, M-Rate, $\text{Precision}^{\text{C}}$ and Recall^{C} over folds. In Figures 7.11 and 7.12, the black lines connecting the lowest, median and highest value over the three folds serve as a visual cue of this variation (for values as numbers cf. Table 7.4). On the training datasets in Figure 7.11a, i.e. on the datasets upon which the detector and classifier were optimized, variation is low in all objectives (except for Recall^{D} on experiment 100 nm LQ). This indicates stability of optimization results not only over different runs of the *Optimization* stage, but also over different data because each fold uses another synthetic dataset for training, and all such datasets are pairwisely disjoint, cf. Section 7.3.4. $\text{Precision}^{\text{D}}$ exhibits larger variations, however, this measure is not subject to the optimization.

Examining variation of objectives and measures over folds for datasets other than training does not, in a strict sense, allow for a statement about stability of optimization, but in a more general sense about stability in applying optimization results to different datasets they have not been optimized for. Nevertheless, low variation of analysis quality over folds of unseen data is a desirable property of a method, which is why it is investigated here. Concerning validation, test and real data, variation in objectives increases compared to training, as expected. This affects particularly $\text{Precision}^{\text{C}}$ on 100 nm experiments and Recall^{D} on 100 nm real data, while generally, variation in objectives remains low. Higher variation besides that can only be observed in the measures $\text{Precision}^{\text{D}}$ and D-Rate that were not used as objectives.

Results Table

Up to this point, final analysis results were discussed solely on the basis of their visual representations in Figures 7.11 and 7.12. Table 7.4 complements this with a numerical representation of the analysis results for the real sensor data, as displayed in Figure 7.12b.

Rows relate to experiments and folds of cross-validation. Rows marked with 'MAE' in the 'Fold'-column are exceptions to this rule as they report Mean Absolute Error attained

[7] Note however, that another factor in the high D-Rate of experiment 100 nm LQ is the low number of $G = 56$ nano-objects in the ground truth, i.e. a low density of nano-objects on the sensor, cf. Table 7.1. Hence, as a further handicap for experiment 100 nm LQ, an FP classifier response has a larger weight in this experiment than in experiments with a higher density of nano-objects.

in performance estimation. Details and an interpretation of these values, as well as of the rightmost column 'D-Diff' are given in the context of evaluating the quality of performance estimates in Section 7.5.6. All other columns refer to the objectives and measures from Section 7.3.1.

Having a numerical representation of Figure 7.12b enables to better resolve the mostly small variations in values over the folds of cross-validation. The point to be made with respect to these is related to stability of overall analysis results on real data: Parameters used over the folds are the results of different optimizations, each conducted with respect to a different training dataset, all of which are pairwisely disjoint. The mostly small variations, particularly in the 200 nm experiments, indicate that choice of the training dataset does not affect analysis quality for real data strongly. Therefore, the three-fold cross-validation that was done here for establishing stability can be omitted in everyday lab practice. As a result, the *Optimization* stage need not be run three times for a new experiment, but only one time. However, in order to reduce optimism in model selection and performance estimation (cf. Sections 3.9 and Section 7.3.4), the three-fold cross-validation-like data division strategy into training, validation and test dataset should be maintained, followed by optimizing on the training dataset only, without permuting the roles of training, validation and test datasets afterwards.

Conclusions

Taking a look at the 'D-Rate'-column of Table 7.4 provides a summary of the analysis outcomes for the six *PAMONO* experiments from Table 7.1: The reported numbers of nano-objects in the real sensor data deviate from their ground truth numbers between −31.795% and +32.143%. Both of these extremes are attained on the 100 nm experiments. The median deviation amounts to −4.139%. Regarding only the 200 nm experiments, the range is from −15.054% to +7.179%, while the median remains the same. Therefore, *SynOpSis* can be applied successfully for *PAMONO* data analysis within applications that tolerate errors of the given magnitudes in reported nano-object counts.

These results validate that parameters optimized on (and classifying models learned from) synthetic data, transfer to the real data to be analyzed. They thus establish practicability of the *SynOpSis* approach and provide baselines on attainable analysis quality. Furthermore, they validate the *PAMONO* signal model in exactly the aforementioned terms: The signal model is accurate enough such that a classifying model can be learned from synthetic data, achieving high quality on real sensor data. An analogous relation holds for detector parameters optimized on synthetic data, which validates the signal model in this respect. The latter point answers the question of transferability from synthetic to real data in *parameter* space. Now the following section repeats this question in *objective* space: It evaluates how well the objective values and other measures obtained on the unseen synthetic test dataset predict those observed on the real data. Hence it examines the performance estimation strategy of *SynOpSis*.

7.5.6 Quality of Performance Estimates

Table 7.4 reports objectives and measures as attained for the six *PAMONO* experiments in Table 7.1. The reported values were determined from a manually created ground truth segmentation of the real sensor data. In everyday lab practice, ground truth for the real data

Table 7.4: Objectives and Measures for Real Sensor Data – Numerical Representation. This table provides a numerical representation of the analysis results for real sensor data as visualized in Figure 7.12b. Rows refer to experiments and folds of cross-validation, except for the rows marked with 'MAE' in the 'Fold'-column. These rows report the Mean Absolute Error incurred when using test set performance, as displayed in Figure 7.12a, as a prediction for performance on real sensor data, cf. Section 7.5.6. In this context, the 'D-Diff'-column resolves by fold the errors made in estimating D-Rate. All other columns refer to the objectives and measures described in Section 7.3.1.

Experiment	Fold	Precision^{D}	Recall^{D}	M-Rate	Precision^{C}	Recall^{C}	D-Rate	D-Diff
200 nm HQ	1	0.97484	0.88068	0.01935	0.97531	1.00000	-0.07955	-0.06044
	2	0.97256	0.90625	0.03135	0.97337	1.00000	-0.03977	0.07679
	3	0.93417	0.84659	0.01678	0.93498	1.00000	-0.08239	-0.01660
	MAE	0.03485	0.07249	0.02871	0.03408	0.00000	0.05128	
200 nm MQ	1	0.92879	0.90090	0.04000	0.93072	0.99677	-0.00300	-0.00300
	2	0.89508	0.81982	0.01099	0.89610	1.00000	-0.07508	0.04120
	3	0.89080	0.93093	0.05806	0.94721	0.99691	0.02402	0.04755
	MAE	0.15717	0.04998	0.08471	0.06021	0.02386	0.03059	
200 nm LQ	1	0.93976	0.83871	0.03846	0.94186	1.00000	-0.07527	-0.02649
	2	0.20513	0.86022	0.07500	0.88764	0.91860	-0.04301	0.12366
	3	0.58974	0.74194	0.04348	0.89873	0.98611	-0.15054	-0.07302
	MAE	0.18002	0.11982	0.09950	0.08779	0.03585	0.07439	
200 nm Gpy	1	0.86139	0.89231	0.05747	0.90052	0.94505	-0.02051	0.10449
	2	0.83920	0.85641	0.05988	0.84466	0.98864	0.05641	0.09808
	3	0.73684	0.86154	0.08333	0.81818	0.93956	0.07179	-0.02765
	MAE	0.14003	0.07815	0.05092	0.12377	0.02351	0.07674	
100 nm HQ	1	0.85185	0.58974	0.02609	0.85714	0.96610	-0.31795	-0.23437
	2	0.25758	0.78462	0.02614	0.85119	0.91083	-0.13846	-0.00815
	3	0.74257	0.76923	0.04000	0.90210	0.82692	-0.26667	-0.22756
	MAE	0.15277	0.21449	0.08323	0.10625	0.06506	0.15669	
100 nm LQ	1	0.32515	0.94643	0.13208	0.75676	0.93333	0.32143	0.47019
	2	0.13576	0.73214	0.12195	0.75000	0.91304	0.00000	0.17647
	3	0.35948	0.98214	0.01818	0.80000	0.85714	0.07143	-0.71730
	MAE	0.40075	0.08924	0.13178	0.25576	0.06695	0.45465	

is usually unavailable, and analysis quality can only be estimated from the synthetic data, for which ground truth is available. In order to do so, the untouched test dataset is kept aside for performance estimation, as described abstractly in Section 3.9, and concretely for *PAMONO* in Section 7.3.4. Now this section evaluates the quality of the performance estimates obtained by using objectives and measures attained on the test dataset as predictions for the (usually unknown) actual values on the real sensor data.

Let p_i denote the prediction for a certain objective or measure, i.e. the value it attains on the test dataset, as displayed in Figure 7.12a. Subscript i indicates the fold of cross-validation

in which p_i was computed. Furthermore, let a_i denote the corresponding actual value for the real sensor data, as displayed in Figure 7.12b and Table 7.4, respectively. Then the error incurred in fold i in the prediction of that objective or measure is $a_i - p_i$, where a negative/positive sign indicates an over-/underestimation of the predicted quantity. Since D-Rate is an important summary measure of analysis quality, individual differences of the type $a_i - p_i$ are reported for it, cf. the 'D-Diff'-column of Table 7.4, where D-Diff is an abbreviation of D-Rate Difference. Therefore, this column quantifies per fold the error incurred by predicting D-Rate on real sensor data, which is unknown in practice, as equal to D-Rate on test data. These signed errors in performance estimates range from −71.730% to +47.019% by which the number of ground truth nano-objects in the real sensor data is over- and underestimated, respectively. Both extremes arise for the most difficult-to-analyze experiment 100 nm LQ. Regarding the 200 nm experiments only, errors range from −7.302% to +12.366%.

In order to summarize prediction errors over folds, an additional row, marked with 'MAE', is provided per experiment in Table 7.4. For each objective and measure from Section 7.3.1, it tabulates the Mean Absolute Error (MAE) over folds, incurred by the respective prediction. As three folds are conducted, the MAE in this special case is defined as:

$$\frac{|a_1 - p_1| + |a_2 - p_2| + |a_3 - p_3|}{3}. \tag{7.7}$$

Generally speaking, the 'MAE'-rows in Table 7.4 can be interpreted as follows: If for a certain objective or measure, the value in the 'MAE'-row is small compared to the values in the three rows above, this indicates that the values on the test sets are good predictors for those attained on real data for the considered objective or measure. Conversely, a comparably large value in the 'MAE'-row indicates low quality predictions. Examining Table 7.4 reveals that MAE magnitudes, and thus prediction qualities, vary considerably by experiment, with the more easily-to-analyze 200 nm experiments also being more easily-to-predict. Furthermore, predictions for the four objectives, i.e. Recall^{D}, M-Rate, $\text{Precision}^{\text{C}}$ and Recall^{C}, exhibit lower MAE than those for measures not used as objectives, i.e. $\text{Precision}^{\text{D}}$ and D-Rate.

With regards to the summary measure D-Rate, which was already discussed in the context of the 'D-Diff'-column, taking the mean over the three folds of cross-validation gives slight improvements: The largest mean absolute estimation error amounts to 45.465%, again on experiment 100 nm LQ. Regarding only 200 nm experiments, MAE ranges from 3.059% to 7.674%. As a conclusion, independent of whether or not mean values over cross-validation results are taken, performance estimation provides useful results for 200 nm experiments, while the quality of performance estimates for 100 nm experiments needs improvement to be serviceable in practice.

7.5.7 Specificity of Final Analysis Results

Optimizing Recall^{D}, i.e. sensitivity of the detector, while putting the classifier alone in charge of specificity, raises the question of specificity in the overall analysis results. This question is to be investigated here by running the *Application* stage on real sensor data that does not contain any nano-objects. Hence any detector response is a False Positive (FP) of the detector. Now the Specificity measure[8] [Pow11] quantifies how well the classifier can sort out those

[8] Details concerning the Specificity measure can be found in Appendix A.

detector FPs by classifying them as negatives, making them True Negatives (TNs) of the classifier. Hence, in this context, Specificity is defined solely with respect to the entries of the *classifier* confusion matrix (cf. Table 3.2). It measures the ratio between TN classifications and all actually negative examples to be classified:

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}. \tag{7.8}$$

Results of measuring Specificity are reported in Table 7.5. They were obtained using the same parameters per *PAMONO* experiment as were used for the final analysis results in Table 7.4. The respective employed classifying model was the same as well, i.e. the model learned from the union of the ground truth-labeled detector results for the training, validation and test datasets, cf. Section 7.3.5.

Table 7.5 reports the results per experiment and fold of cross-validation. FPs incurred by the detector are tabulated as absolute numbers in the 'FP'-column and as relative densities in the 'FP per px^3'-column. The latter normalize the former by the number of pixels in the respective nano-objects-free spatiotemporal input volume. These two columns are also given for the classifier, thus tabulating the absolute numbers and relative densities of FPs *remaining* after classification. Finally, the 'Specificity'-column of the classifier tabulates the relative amount of FP detector responses that could be sorted out by the classifier. It can be derived as according to Equation (7.8) from the detector and classifier 'FP'-columns: The total number of actually negative examples in the denominator of Equation (7.8) is equal to the number of FP detector responses. The number of TN classifications in the numerator of Equation (7.8) is the difference between all actually negative examples in the denominator and the FP classifications. As an example, for fold one of experiment 100 nm HQ, Specificity = $\frac{10-2}{10} = 0.8$.

In examining the numbers in Table 7.5, perfect Specificity can be observed over all folds in experiments 200 nm HQ and 200 nm Gpy. In the former, this is due to zero FP responses in the detector rendering classification unnecessary, while in the latter, FP responses occur, but they can all be sorted out by the classifier. Two folds in the 200 nm experiments do not produce perfect Specificity, but the number of FP responses is reduced from 77 to one, and from 245 to two, respectively. On the 100 nm experiments, FP detector responses ranging from nine to 984 are encountered, which are reduced to numbers between zero and 15 by the classifier. These numbers of remaining FPs after classification are small enough to allow for a fast visual inspection by a human expert. The lowest occurring Specificity is 0.8 in fold one of the 100 nm HQ experiment. However, its absolute number of FPs after classification is two, and the low value of Specificity is due to the overall small number of ten FP detector responses. FP densities before classification range from zero to $1.39130 \cdot 10^{-6}$, and the classifier reduces this to a range from zero to $4.90196 \cdot 10^{-8}$, with all but one case exhibiting a density below $8.5 \cdot 10^{-9}$, and eleven out of 18 cases with a perfect Specificity of one. Among the 200 nm experiments, ten out of twelve cases exhibit Specificity one.

7.5.8 Computation Time

As a sensor technique, *PAMONO* enables detection of nano-objects in real-time, while they attach to the sensor surface. Hence a vital goal in developing an automated analysis process for *PAMONO* data is retaining this real-time capability in the analysis, while providing a real-time preview of processed sensor images to the operator. In order to demonstrate

Table 7.5: Specificity with Respect to Nano-Object-Free Sensor Data. For all experiments and folds from Table 7.4, the *Application* stage of *SynOpSis* was additionally run on real sensor data that was recorded before any nano-objects were inserted into the flow cell of the sensor. The same parameters and classifying models as before were used. This enables assessing Specificity of the results: The 'FP'-column of the detector tabulates the absolute number of spurious detector responses, while its 'FP per px^3'-column normalizes this number by the number of pixels in the nano-object-free spatiotemporal input volume. The same columns are also given for the classifier results, and the 'Specificity'-column reports the relative amount of FP detector responses that could be eliminated by the classifier. Eleven out of 18 cases attain perfect Specificity of one. Among the 200 nm experiments ten out of twelve cases exhibit Specificity one.

		Detector		Classifier		
Experiment	Fold	FP	FP per px^3	FP	FP per px^3	Specificity
200 nm HQ	1	0	0	0	0	1
	2	0	0	0	0	1
	3	0	0	0	0	1
200 nm MQ	1	1	$5.30594 \cdot 10^{-9}$	0	0	1
	2	1	$5.30594 \cdot 10^{-9}$	0	0	1
	3	77	$4.08557 \cdot 10^{-7}$	1	$5.30594 \cdot 10^{-9}$	0.98701
200 nm LQ	1	0	0	0	0	1
	2	245	$1.03900 \cdot 10^{-6}$	2	$8.48162 \cdot 10^{-9}$	0.99184
	3	54	$2.29004 \cdot 10^{-7}$	0	0	1
200 nm Gpy	1	15	$4.68915 \cdot 10^{-8}$	0	0	1
	2	19	$5.93959 \cdot 10^{-8}$	0	0	1
	3	36	$1.12540 \cdot 10^{-7}$	0	0	1
100 nm HQ	1	10	$1.41393 \cdot 10^{-8}$	2	$2.82785 \cdot 10^{-9}$	0.8
	2	984	$1.39130 \cdot 10^{-6}$	1	$1.41393 \cdot 10^{-9}$	0.99898
	3	176	$2.48851 \cdot 10^{-7}$	2	$2.82785 \cdot 10^{-9}$	0.98864
100 nm LQ	1	9	$2.94118 \cdot 10^{-8}$	0	0	1
	2	369	$1.20588 \cdot 10^{-6}$	1	$3.26797 \cdot 10^{-9}$	0.99729
	3	394	$1.28758 \cdot 10^{-6}$	15	$4.90196 \cdot 10^{-8}$	0.96193

real-time capability of the *Application* stage of *SynOpSis*, the Frames per Second (FPS) it can process were measured for the experiments from Table 7.1, using the hardware from System Specification 7.1. Note that all times reported in this section relate to optimization mode *Global 4* and desirability mode *Scalarizing*, i.e. to the modes determined in Section 7.5.4 and already examined in Sections 7.5.5 to 7.5.7.

Frames per Second (FPS) and Real-Time-Capability

Table 7.6 shows the results of measuring computation times. The 'FPS'-column refers to the average number of *PAMONO* images that can be processed per second, given the resolution in the first two dimensions of the 'Training Vol. (px^3)'-column. Image sizes in the training datasets are the same as in the real sensor data, and only the size of the temporal dimension

Table 7.6: Computation Times. Computation times of different types are shown per experiment. The 'FPS'-column (Frames per Second) tabulates the number of *PAMONO* images that can be processed per second during the *Application* stage of *SynOpSis*. The 'f_e'-column divides FPS by the number of images recorded per second (cf. Table 7.1), thus giving the factor of excess-speed attained by the *Application* stage. The 'Training Vol. (px^3)'-column lists the sizes of the spatiotemporal volumes used as training data in the *Optimization* stage. Evaluating one individual on the training dataset during optimization takes the number of seconds displayed in the 'Eval. (s)'-column, while the number of hours taken for the overall optimization is tabulated in the 'Opt. (h)'-column.

Experiment	FPS	f_e	Training Vol. (px^3)	Eval. (s)	Opt. (h)
200 nm HQ	64.36249	3.21812	$1080 \times 145 \times 666$	23.80570	16.53174
200 nm MQ	98.04402	4.90220	$742 \times 127 \times 666$	18.03593	12.52495
200 nm LQ	83.74859	4.18743	$706 \times 167 \times 666$	21.48633	14.92108
200 nm Gpy	34.32416	2.28828	$1024 \times 270 \times 385$	25.33500	17.59376
100 nm HQ	59.72236	1.49306	$750 \times 230 \times 1366$	46.25623	32.12238
100 nm LQ	129.04058	3.22601	$450 \times 170 \times 1333$	27.67687	19.22004

varies. Processing one image means that the detector is applied to it, and that the classifier in terms of the Random Forest classifying model is evaluated for the output of the detector. The classifying model is known ahead (cf. Section 7.3.5) and applied in real-time on the GPU [Lib15b]. Each entry in this 'FPS'-column tabulating analysis speed is larger than the corresponding entry for recording speed, given in the 'FPS'-column of Table 7.1. To simplify interpretation, the 'f_e'-column tabulates the factor by which analysis speed *exceeds* recording speed: It ranges from 1.49306 for the 100 nm HQ experiment to 4.90220 for the 200 nm MQ experiment. Hence, real-time capability of the *Application* stage of *SynOpSis* has been demonstrated for the examined experiments, with a safety-margin starting at an excess-speed factor of approximately 1.5.

Optimization Time

A high speed analysis process is not only vital for real-time *PAMONO* data analysis in the *Application* stage of *SynOpSis*, but it also accelerates the *Optimization* stage. The 'Eval. (s)'-column in Table 7.6 tabulates the time taken to evaluate one individual on the training dataset, the size of which is listed in the 'Training Vol. (px^3)'-column. Evaluation times include all overheads arising from writing and parsing the text files containing detector results, matching these results to the ground truth and conducting five-fold cross-validation of the Random Forest classifier with the parameter set to be evaluated, cf. Section 7.3.4. Learning the Random Forest classifying models involved in this cross-validation takes place on the CPU. Due to these overheads, evaluation times are not comparable with the times reported in the 'FPS'-column. Evaluation times were averaged over the 2500 individuals examined in the *Optimization* stage, cf. the configuration of the GA in Section 7.3.2. Times ranging from 18 s to 46 s per average individual can be observed, exhibiting approximate proportionality to the size of the spatiotemporal volume used for training. The 'Opt. (h)'-column complements individual times by reporting the accumulated time required for running the *Optimization* stage, i.e. it multiplies evaluation times by the 2500 individuals and presents the result in hours. Optimization times range from 12 h to 32 h.

7.6 Parameter Choices of the Optimization Stage

In the previous section, results of applying *SynOpSis* for *PAMONO* data analysis were reported in terms of the objective function values attained after optimization. The focus of those evaluations was hence on how the parameters selected for being the most *desirable* ones perform in *objective space*. In this section, the focus is shifted in two ways: In addition to the desirable parameters, all parameters on the *Pareto front* are regarded, i.e. all non-dominated individuals from the optimization. Furthermore, these individuals are no longer examined in objective space, but in *parameter space*, allowing to draw conclusions about what parameter values make a good individual. This is the topic of Section 7.6.1. While that section takes the viewpoint of analyzing the parameter values observed in optimizations already conducted, Section 7.6.2 takes the idea one step further by taking a viewpoint of prediction: The systematics between parameter space values and objective space values are investigated by predicting the latter from the former. In order to do so, the pool of examined data is extended further, to also encompass the dominated individuals. On this basis, regression functions, mapping from parameter- to objective space are computed to predict objective values for arbitrary parameter values. The quality of these predictions is evaluated in a cross-validation [Koh95].

Employed Data

For the two tasks described above, it is beneficial to use as much data as is available. Therefore, in contrast to the second half of Section 7.5, the considered optimization- and desirability modes are no longer restricted to *Global 4* and *Scalarizing*. The focus is shifted to having a large number of tuples of parameter- and objective vectors, while the modes in which these were obtained are no longer relevant. However, one exception is the *Sequential* optimization mode: It first optimizes detector parameters and then, with a single set of detector parameters fixed, it optimizes classifier parameters. Therefore, all but one detector parameter set are associated with no classifier parameter set, and all classifier parameter sets are associated with the same single detector parameter set. Hence, to ensure a bijective correspondence between detector and classifier parameter sets, the analysis in this section regards only the results of global optimizations, i.e. of optimization modes *Global 4* and *Global 3*, cf. Terminology 7.2. Global optimizations ensure bijective correspondence between detector and classifier parameter sets because both are optimized *simultaneously*. Aside from this restriction, all available data is used, i.e. data from all experiments, desirability modes and folds.

7.6.1 Examining Pareto Fronts in Parameter Space

In this section, the most desirable parameter sets, i.e. those finally chosen by model selection, are regarded in parameter space, thus complementing their evaluation in objective space, as conducted in Section 7.5. In conjunction with the desirable parameter sets, the parameter sets on the Pareto fronts are regarded as well. This investigation serves to gain insight into which parameter values are desirable or non-dominated, allowing to conclude what makes a good parameter set. In particular, the algorithms and combinations thereof that perform best in analyzing *PAMONO* sensor data can be identified.

Content of Parameter Histograms

The primary tool in this investigation is histograms over parameter values. Taking Figure 7.13 as an example, the histograms therein plot values observed in parameters for background elimination in the detector, cf. Section 5.2. Values observed in individuals on the Pareto front are plotted in blue, while those observed in desirable individuals, i.e. in those finally selected, are overlaid in semi-transparent red. All histograms are normalized as probabilities, i.e. the bars in the red and blue histograms each sum to unit area. This normalization serves to equalize the difference in numbers between individuals on the Pareto front and those selected as desirable: The total number of parameter sets over all Pareto fronts was 3981, after removal of duplicates, which prevents them from voting multiple times for the same parameter values. The total number of desirable parameter sets was 108, resulting from six experiments, two considered global optimization modes, three desirability modes and three folds, with each optimization contributing the single finally selected parameter set.

Parameter histograms are pooled into topical groups: Each processing module of the detector in Figure 5.1 is treated in a separate figure, as well as the Random Forest-based version of the classifier in Figure 6.1. For each such figure, binary parameters are treated differently from the other parameters: As an example, the binary parameter b^{bg} in Figure 7.13a votes by majority for median-based background elimination. Then for all other parameters in this processing module, in this case for the lengths w^{ρ} and w^{ϕ} of the employed temporal windows in (b) and (c), the histograms are regarded only over those individuals where the binary parameter b^{bg} was set to use the median. Hence the non-binary parameters are restricted to the cases deemed relevant by the binary ones. Parameters related to methods deactivated by binary parameters in the same processing module are omitted from the discussion. Note that processing modules are treated independently of each other: As an example, if a certain binary value was chosen for the background elimination module, histograms for the denoising module are not influenced by this decision, but again *all* individuals are regarded, i.e. binary decisions for previous modules are not propagated to later modules. Note that while not being captured by this type of histograms, there exists the possibility of more complex parameter interactions: As an example, denoising parameters and thresholds in time series classification can interact because stronger smoothing during denoising causes stronger smearing of peak intensities, thus requiring smaller thresholds.

Interpretation of Parameter Histograms

A general remark on how to interpret the parameter histograms to be discussed throughout this section is that conclusions can only be drawn from histograms with low entropy. If a histogram has high entropy, i.e. if its distribution is rather uniform, one can not conclude whether this is due to the parameter being irrelevant, thus requiring no optimization, or whether rather diverse values of this parameter are required to give good analysis results, thus voting for optimizing that parameter. The only information in such histograms is that all values of a parameter are about equally frequent in good individuals. In contrast to that, parameter histograms with low entropy, i.e. those indicating a tendency of agglomeration around certain parameter values, vote for looking at those values first during search. Given this general remark on histogram interpretation, detailed discussions of the parameter histograms per processing module will be provided now.

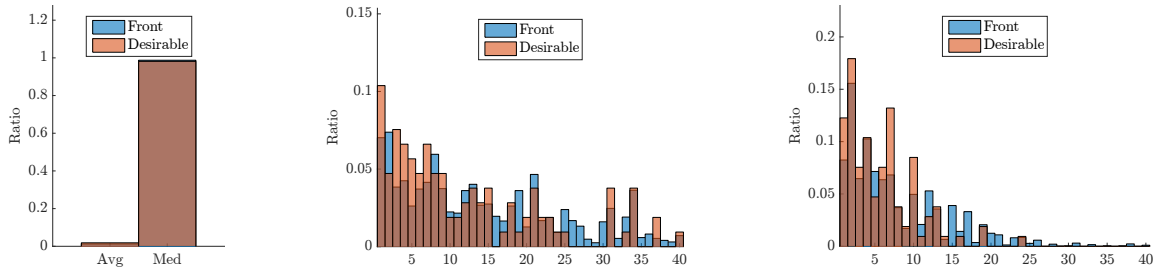


Figure 7.13: Background Elimination Parameter Histograms. The semantics of these parameters is explained in Section 5.2.4, and an analysis of the histograms is provided in the text.

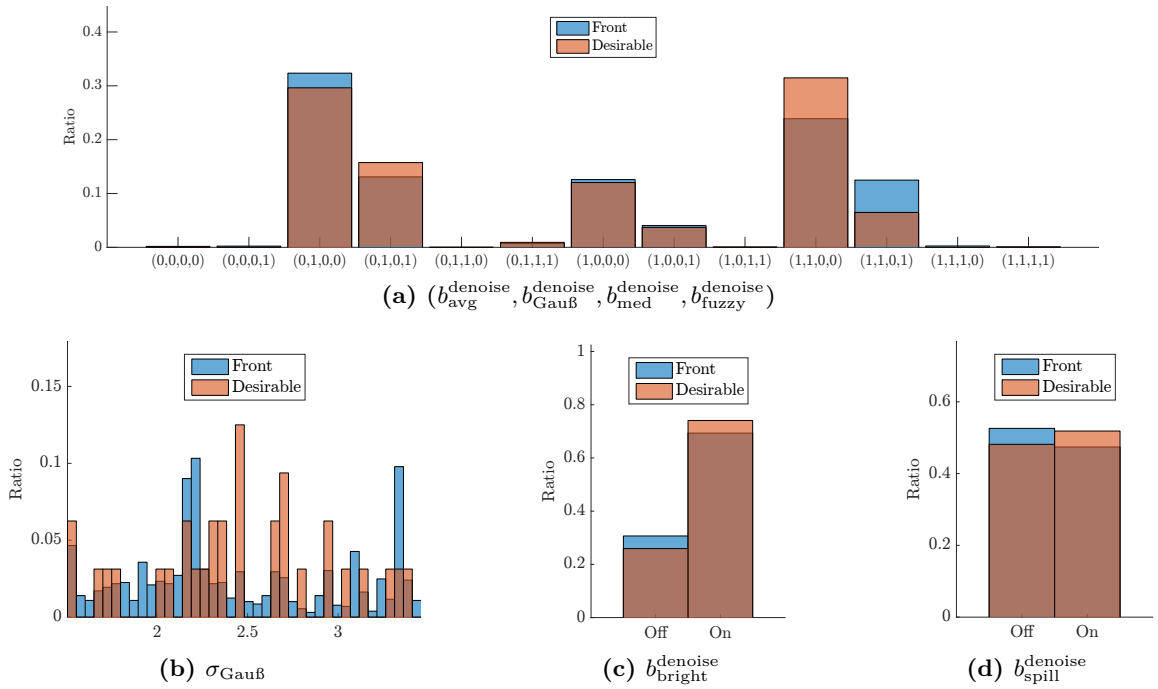
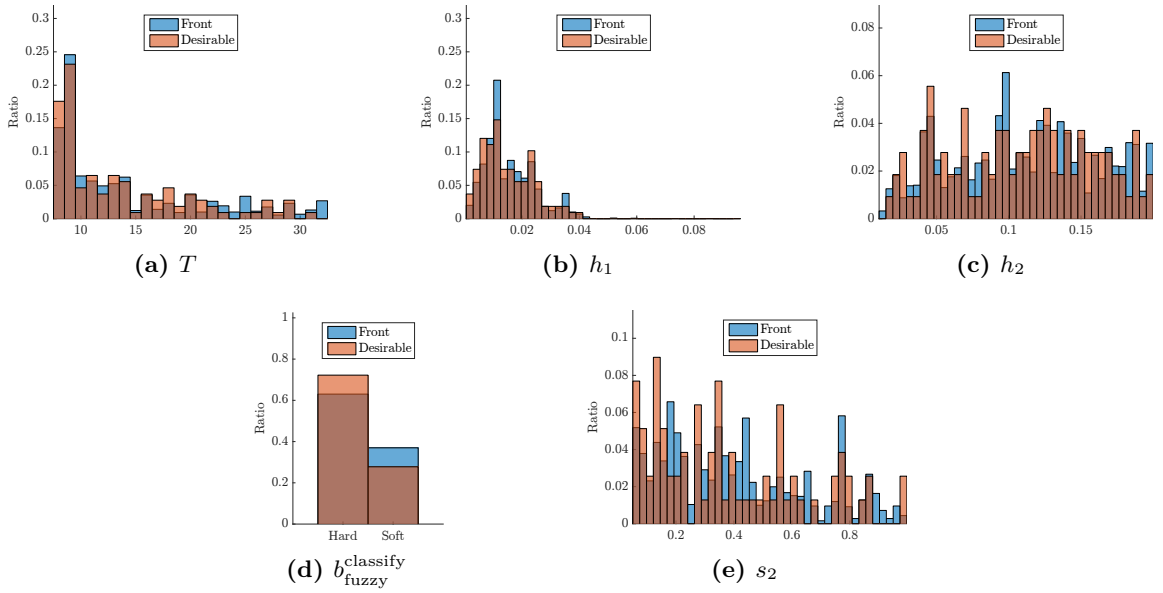


Figure 7.14: Denoising Parameter Histograms. The semantics of these parameters is explained in Section 5.3.4, and an analysis of the histograms is provided in the text. Binary vectors in (a) represent the following binary parameters in the order given here: $(b^{\text{denoise}}_{\text{avg}}\, b^{\text{denoise}}_{\text{Gauß}}\, b^{\text{denoise}}_{\text{med}}\, b^{\text{denoise}}_{\text{fuzzy}})$. Parameters of methods deactivated by the majority vote on the binary parameters in (a) are not shown.

Background Elimination

Figure 7.13 shows histograms over parameters of the background elimination module, as explained in Section 5.2.4. The vote for aggregating past and present estimates via the median over time is very clear in (a), for both, front- and desirable individuals. The lengths of the temporal windows over which the past (b) and present (c) are aggregated in the median-based parameter sets tend to agglomerate on the lower end, voting for rather short windows in both, with past windows being slightly longer.



(a) T **(b)** h_1 **(c)** h_2

(d) $b^{\text{classify}}_{\text{fuzzy}}$ **(e)** s_2

Figure 7.15: Time Series Classification Parameter Histograms. The semantics of these parameters is explained in Section 5.4.4, and an analysis of the histograms is provided in the text. Parameters of soft thresholding are not shown because this method is deactivated by the majority vote on the binary parameter in (d).

Denoising

Figure 7.14 shows histograms over parameters of the denoising module, as explained in Section 5.3.4. The most frequently occurring types of denoising methods are Gaussian smoothing alone, or in combination with averaging. The proportion of front individuals voting for Gaussian smoothing alone slightly exceeds that of desirable individuals voting for the combination with averaging. Therefore, (b)–(d) show histograms solely over individuals employing Gaussian smoothing alone. Smoothing kernel sizes in (b) are chosen primarily about the center of the examined range, and less so from the more extreme values. This indicates that the examined range is large enough and thus can provide the amount of smoothing required for the examined experiments. Among the application-specific denoising heuristics, brightness correction is enabled in the majority of cases for both, front and desirable individuals (c), whereas no conclusions can be drawn from the histogram concerning pixel overspilling compensation (d). Settling this parameter choice would require re-evaluating all individuals with the contrary choice in this parameter, while all other parameters are left unchanged, followed by comparing the obtained objective values to the original ones.

Time Series Classification

Figure 7.15 shows histograms over parameters of the time series classification module, as explained in Section 5.4.4. Concerning time series preselection parameters in (a)–(c), a tendency for smaller temporal windows within which time series are matched and classified can be determined from (a). The threshold determining the minimum required magnitude in time series intensity-differences agglomerates on the lower end of the examined range (b). Nearly no individuals with lower thresholds in the upper half of the interval occur in

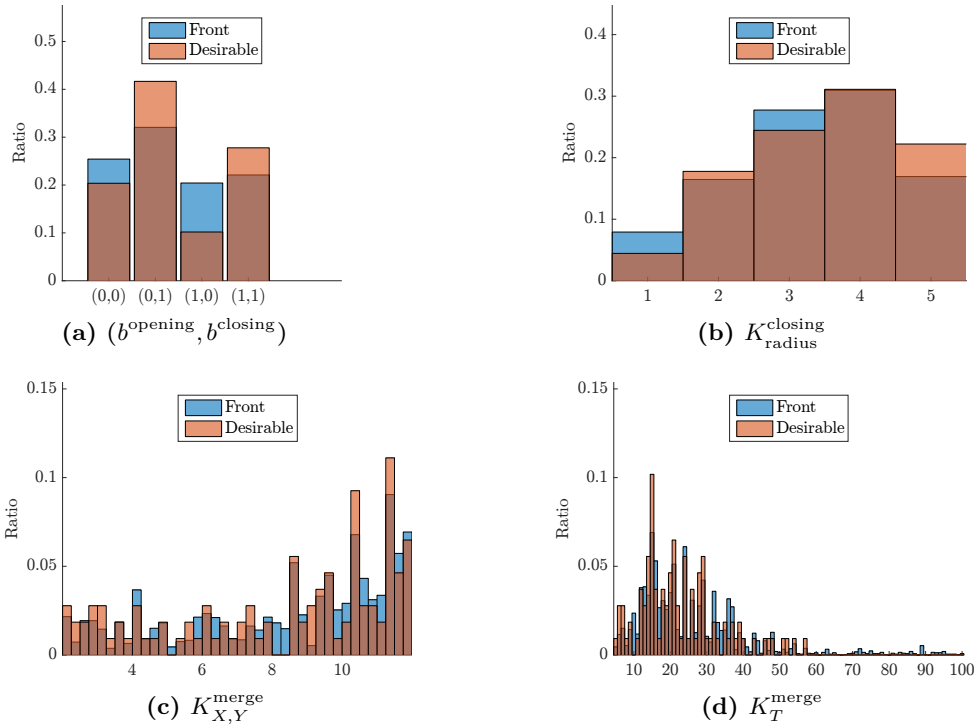


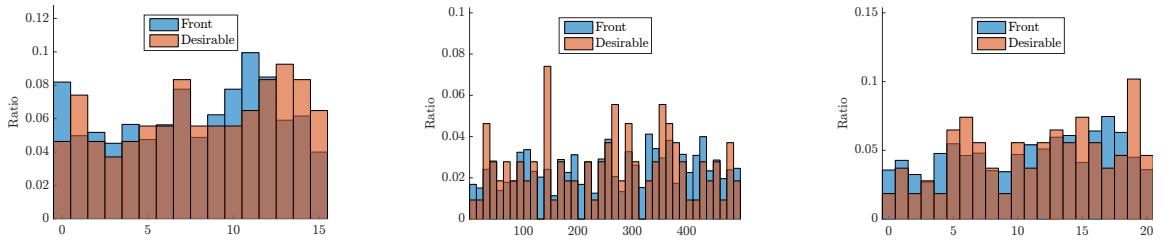
Figure 7.16: Segmentation Parameter Histograms. The semantics of these parameters is explained in Section 5.6.4, and an analysis of the histograms is provided in the text. Binary vectors in (a) represent the following binary parameters in the order given here: $(b^{\text{opening}}, b^{\text{closing}})$. The radius parameter of morphological opening is not shown because opening was deactivated by the majority vote on the binary parameters in (a).

front and desirable individuals. The vote for an upper threshold is less clear (c): Values are approximately uniformly distributed throughout the examined range, with a slight decrease in the interval overlapping with that in (b). One possible explanation for the high entropy in (c) is that an upper threshold on time series magnitude is irrelevant, which is backed by the fact that over the examined experiments, artifacts were rarely brighter than nano-objects. Hence nothing is to be gained from imposing an upper threshold.

The bottom part of Figure 7.15 is concerned with time series classification based on matching scores. A rather clear vote for hard thresholding of matching scores can be seen in (d). The distribution of the employed hard thresholds in (e) is skewed toward requiring lower matching scores, which can be due either to the low SNR in the data decreasing matching scores, or to well-performing time series preselection making structural matching scores less relevant. Both causes can apply, and they can vary by experiment.

Segmentation

Figure 7.16 shows histograms over parameters of the segmentation module, as explained in Section 5.6.4. Binary switches enabling morphological opening and closing are examined in (a). While both, front and desirable individuals vote for applying solely closing, the vote is clearer in the desirable than in the front individuals. An intuition behind the success of closing can be gained from comparing the effects of morphological operators in Figure 5.11.



(a) K **(b)** I **(c)** D

Figure 7.17: Random Forest Classifier Parameter Histograms. The semantics of these parameters is explained in Section 6.8, and an analysis of the histograms is provided in the text.

A parameter histogram for the radius of the structuring element used in closing is given in (b), voting for larger structuring elements with radii three or four.

The bottom row of Figure 7.16 is concerned with spatiotemporal clustering of polygons, which like morphological operators primarily affects the M-Rate objective. The histogram over the spatial radius within which multiple polygons are merged to a single detector response is skewed in the direction of larger values (c), hinting at a range of examined values that was too small on the upper end. Improvements in M-Rate can possibly be achieved by increasing the upper boundary of the interval examined for this parameter. Conversely, the examined range for the temporal radius of polygon clustering can be reduced (d): The vast majority of individuals agglomerates in the lower half of the range, offering a possibility for pruning the search space in future optimizations.

Random Forest Classifier

Figure 7.17 shows histograms over parameters of the Random Forest classifier, as explained in Section 6.8. Neither the number of features available for splitting at the inner nodes of trees (a), nor the number of trees in the forest (b), nor maximum allowed depth of the trees (c) clearly vote for a sweet spot. This agrees with the relative robustness of the Random Forest learner concerning parameter choices, as discussed in more detail at the end of Section 6.6.3. Not even the parameter K which trades off correlation between-, versus predictive strength of-, the individual trees makes a clear vote for certain values. Determining whether this means that the Random Forest parameters are negligible in optimization or simply uniformly distributed in front and desirable individuals could be conducted with respect to the results of the *Sequential* optimization mode: In the second optimization that is concerned solely with the classifier, one could track classifier objectives over parameters and compute their variation. If this variation is tolerable in analysis results, Random Forest parameters can simply be set to default values [Bre01; HTF09]. Note however, that in global optimization modes no acceleration is to be gained from this because classifier objectives have to be evaluated anew for each individual anyways. Whether this is done with fixed or variable Random Forest parameters is not of importance if these parameters only have small a small impact on the classifier performance of the individual.

7.6.2 Modeling Parameter Set Quality in Objective Space

After the previous section examined the front and desirable individuals in parameter space to identify systematics in what makes good parameter sets, this section takes the idea of investigating these systematics one step further. Instead of analyzing the parameter space representation of good individuals over optimizations already conducted, a predictive viewpoint is taken here: Objective values for unseen parameter sets are to be predicted by a regression model, thus exploring the systematics between parameter and objective values in a prospective way. This can be exploited in accelerating optimization, which motivates this preliminary investigation: As summarized in Section 3.3, meta-models [BLP10; KKF+11; HHL11; BBB+11; BMT+12] of response surfaces over parameter space can be computed. Each such meta-model predicts the behavior of one objective over parameter space. Meta-models can be used as cheaper-to-evaluate and faster-to-search surrogates for the actual objective functions, cf. [HHL11] for an according approach. The success and utility of meta-modeling increases with increasing predictability of objective values over parameter space, therefore the investigation in this section examines how well regression models perform in that prediction task.

Regression Method

Random Forest regression [Bre01] was selected as the learning algorithm to be used because it can handle the mixed numerical and categorical values arising in the examined parameter space. As this is a supervised method, the input consists of tuples of parameter vectors, labeled with their associated objective vectors. The labeled tuples were recorded over all runs of the *Optimization* stage, except for those with optimization mode *Sequential*, for the reason discussed at the beginning of Section 7.6. Note that in contrast to Section 7.6.1, not only the front and desirable but *all* individuals created during the optimizations are considered because the prediction model should also be capable of predicting, which combinations of parameter values are likely to be dominated, so it can avoid searching in such regions. The output of this regression is a mapping from parameter to objective space, evaluable at any point of parameter space and predicting the associated values in objective space. As the labeled data points may reside at arbitrary locations in parameter space, and as their objective values are not necessarily interpolated by the resulting regression model, this can be regarded as a scattered data approximation. For multidimensional objective spaces, multiple scalar regression models can be computed, as will be done here.

For each objective, a Random Forest regression model using 100 trees with no limit on tree depth is computed. The number of variables from which the function value is predicted is 31, consisting of the 28 parameters of the detector (cf. Section 5.7) plus three parameters of the Random Forest-based classifier (Section 6.8). The Random Forest used for regression may select from $\lfloor\sqrt{31}\rfloor = 5$ variables at each node, following Breiman's recommendation [Bre01] discussed in Section 6.6.3. In order to avoid undue optimism, this process is conducted within a ten-fold cross-validation [Koh95], cf. Section 3.9.1.

Histograms of Objectives

Before predictability of objectives is assessed in terms of the regression errors incurred in this cross-validation, the behavior of the objective functions is examined in their co-domain.

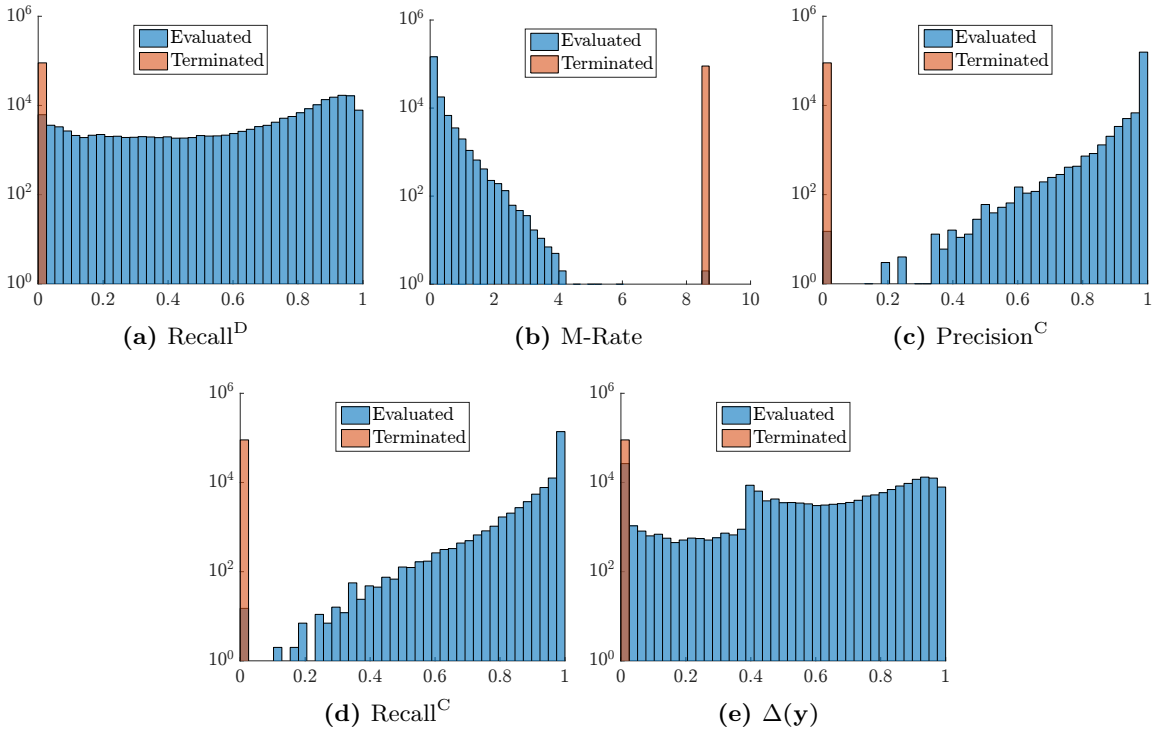


Figure 7.18: Histograms over Objectives and Desirability Index (DI). Observation frequencies of objective- and DI-values are displayed as histograms with logarithmic vertical axes. The histograms differentiate between values attained by regularly evaluated (blue) and terminated individuals (semi-transparent red). The latter are due to an early cancellation criterion (cf. Section 5.8) and are assigned the worst objective values observed over the evaluated individuals. In total, values of 270000 individuals are displayed in each histogram, among which approximately one third was terminated.

To this end, Figure 7.18 displays histograms of the functions to be modeled, with objective values on the horizontal axis and the logarithmic number of observations on the vertical. For parameter sets where the early termination criterion during matching was fulfilled (cf. Section 5.8), the function value was set to the worst value observed over the rest of the data. For example, in the case of M-Rate, it was set to the largest value of M-Rate attained by any other parameter set. In order to distinguish terminated from regularly evaluated individuals, the former are indicated by semi-transparent red bars in the histogram. The number of terminated points is 90185 and thus approximately one third of the total set of 270000 labeled input examples. This number arises from the combination of six experiments, two global optimization modes, three desirability modes and three folds of *SynOpSis* cross-validation. Each of these combinations involves an optimization evaluating 2500 individuals (cf. Section 7.3.2), thus giving the total number of data points.

Recall$^{\mathrm{D}}$ in (a) is approximately evenly distributed between 0.2 and 0.6. Below 0.2 and above 0.6 there are increases in observation frequencies, with the increase towards the higher objective values being larger in this logarithmic plot. In the last bin before the perfect Recall$^{\mathrm{D}}$ of one, frequency drops, indicating that perfect Recall$^{\mathrm{D}}$ is hard to attain, even on the training data used by the *Optimization* stage.

M-Rate in (b) is easier to perfect: The histogram looks approximately linear on a logarithmic vertical axis, with observation frequencies increasing from the worst to the best values of M-Rate. This means there is an exponential decrease in observation frequencies with worsening objective values. Furthermore, the best bin including zero M-Rate breaks the trend with an additional increase in observation frequency.

Precision^{C} and Recall^{C} in (c) and (d) show a very similar behavior, with an even more pronounced increase in observation frequencies on the end of the best objective values.

Besides the raw objective values as summarized in (a)–(d), it is worthwhile to look at the geometric mean DI $\Delta(\mathbf{y})$ of all objectives, cf. Equation (3.8). The reason is that it summarizes all four objectives in a single scalar, while incorporating application preferences: If the DI can be predicted well, a single meta-model suffices in accelerating optimizations with desirability mode *Scalarizing*. In this task, predicting a single response surface might furthermore be more exact than predicting four individual response surfaces, followed by computing the DI of the result, which will be investigated in the following. The DI exhibits the most intricate histogram, as shown in (e): Besides the 90185 terminated individuals, another 26537 evaluated individuals fall into the bin with lowest $\Delta(\mathbf{y})$. A shape similar to that of Recall^{D} in (a) can be observed between the bin involving $\Delta(\mathbf{y}) = 0.4$ and the one with $\Delta(\mathbf{y}) = 1$. The drop in observation frequencies towards $\Delta(\mathbf{y}) = 1$ is inherited from Recall^{D} because no other objective exhibits a drop towards the most desirable value.

Regression Error: Measures and Results

After examining the distributions of objective values and DI, their predictability will be investigated now by applying the regression method described above. Predictability will be measured in terms of the mean regression errors incurred within the conducted ten-fold cross-validation over the respective tuples of parameter- and objective space points.

Table 7.7 shows the mean regression errors in terms of the following measures: Firstly, the Pearson Product-Moment Correlation Coefficient (PCC) quantifies the correlation between actual function values $\mathbf{a} \in \mathbb{R}^N$ and their predictions $\mathbf{p} \in \mathbb{R}^N$ as

$$\mathrm{PCC}(\mathbf{a}, \mathbf{p}) = \frac{\mathrm{cov}(\mathbf{a}, \mathbf{p})}{\sigma(\mathbf{a})\,\sigma(\mathbf{p})}, \tag{7.9}$$

where $\mathrm{cov}(\mathbf{a}, \mathbf{p}) = \frac{1}{N-1}\sum_{i=1}^{N}(a_i - \mu(\mathbf{a}))(p_i - \mu(\mathbf{p}))$ is the sample covariance. PCC resides in $[-1, 1]$, where the extremes indicate negative and positive linear dependence, respectively, and value zero is attained for variables with no correlation. Secondly, Mean Absolute Error (MAE) is defined as

$$\mathrm{MAE}(\mathbf{a}, \mathbf{p}) = \frac{1}{N}\sum_{i=1}^{N}|a_i - p_i|. \tag{7.10}$$

It resides on the same scale as its arguments. The same holds for Root Mean Squared Error (RMSE), which is defined as

$$\mathrm{RMSE}(\mathbf{a}, \mathbf{p}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(a_i - p_i)^2}. \tag{7.11}$$

Compared to MAE, RMSE responds more strongly to outliers. In order to give a reference for the scale that MAE and RMSE reside on for the respective functions, the minimum and

Table 7.7: Regression Errors of Objectives and Desirability Index (DI). Regression errors of the optimized objectives and the DI $\Delta(\mathbf{y})$ are reported in terms of Equations (7.9) to (7.11). The topmost regression model was learned from 'All' training data, while the two models below were learned from the highest/lowest quality experiments '200 nm HQ' and '100 nm LQ', respectively. The minima and maxima reported in the 'Range'-columns indicate the scales upon which the errors in the 'MAE'- and 'RMSE'-columns reside. Higher PCC and lower MAE and RMSE indicate higher predictability, which can be exploited in meta-modeling.

		Regression Errors			Range	
Experiment	Measure	PCC	MAE	RMSE	min	max
All	Recall^{D}	0.8794	0.1347	0.1971	0	1
	M-Rate	0.8308	1.4776	2.2802	0	8.7059
	$\text{Precision}^{\text{C}}$	0.8348	0.1693	0.2607	0	1
	Recall^{C}	0.8337	0.1709	0.2591	0	1
	$\Delta(\mathbf{y})$	0.8504	0.1585	0.2128	0	1
200 nm HQ	Recall^{D}	0.8753	0.1550	0.2177	0	1
	M-Rate	0.8242	1.5889	2.3509	0	8.7059
	$\text{Precision}^{\text{C}}$	0.8278	0.1814	0.2687	0	1
	Recall^{C}	0.8261	0.1833	0.2684	0	1
	$\Delta(\mathbf{y})$	0.8528	0.1683	0.2252	0	1
100 nm LQ	Recall^{D}	0.8574	0.1306	0.1901	0	1
	M-Rate	0.7889	1.7897	2.5486	0	8.7059
	$\text{Precision}^{\text{C}}$	0.7942	0.2039	0.2894	0	1
	Recall^{C}	0.7917	0.2038	0.2868	0	1
	$\Delta(\mathbf{y})$	0.8271	0.1545	0.2033	0	1

maximum values observed in the data are given in the rightmost columns of Table 7.7. This is done for three kinds of input: In rows marked with 'All' in the 'Experiment'-column, data points from all six experiments are used, i.e. the full set of 270000 labeled input examples, as discussed in the context of objective histograms above, is used. In order to sort out the impact of mixing data points from different experiments, the rows marked with '200 nm HQ' and '100 nm LQ' in the 'Experiment'-column, exclusively regard data points from the highest and lowest quality experiment, respectively. Hence, they contain only a sixth of the data points used in the 'All'-rows. Note that the 'Range'-columns remain constant over experiments, making regression errors comparable.

One thing to note in Table 7.7 is that regression errors for Recall^{D} are lowest, indicating a higher degree of dependence between the parameter set and Recall^{D} than for the other objectives. The second lowest error is incurred by the DI, while the three remaining objectives exhibit larger errors. In the context of using a meta-model in running the *Optimization* stage with desirability mode *Scalarizing*, this suggests that it is better to use a single response surface model for the DI directly, than using separate models per objective and computing the DIs from predictions of its constituents. This bears the additional advantage of facilitating search by making it scalar. Another point in Table 7.7 can be seen from comparing regression

errors between experiments: As expected, they decrease from the lowest quality experiment 100 nm LQ to the highest quality experiment 200 nm HQ, both of which are represented with equal amounts of training data tuples. However, regression errors can be decreased further by learning the regression model from all experiments simultaneously, increasing the amount of training data tuples by factor six. Compared to regarding experiment 200 nm HQ alone, this introduces exclusively tuples from lower quality experiments to the training data, but the regression errors nevertheless decrease, as can be seen from comparing the values in the '200 nm HQ'-rows to those in the 'All'-rows.

Conclusion

As investigated in this section, successful prediction of *SynOpSis* objectives, particularly of the DI, by means of Random Forest regression is feasible: A correlation of 0.8504 between predicted and actual DI was observed in a ten-fold cross-validation over all training data. On an absolute scale, MAE = 0.1585 and RMSE = 0.2128 were attained in the co-domain of the DI, which ranges from zero to one. Predictability of Recall^{D} was slightly higher, while that of the other three objectives was lower. Creating a regression model over optimizations conducted for multiple experiments generally increased the prediction quality. This can be exploited particularly in using meta-modeling as a method to aggregate knowledge gained over multiple *PAMONO* sensor setups and experiments. Such an aggregate model can serve in accelerating future optimizations for new experimental data. Furthermore, it can be used as a container accumulating data on a central server within a distributed network of collaborating *PAMONO* sensors [LKD+14]: Each node contributes its function evaluations to the model, and during optimization it can benefit from the predictions generated by the model. Hence it exploits the knowledge gathered by all collaborating nodes, while contributing its share to it.

7.7 Cross-Experiment Generalization Performance

In the final section of this evaluation, the generalization performance of parameter sets and of classifying models is evaluated *across* the different *PAMONO* experiments from Table 7.1. These two aspects are treated separately in the following: Firstly, the generalization performance of parameter sets alone is examined, by analyzing each *PAMONO* experiment in question with detector and classifier parameter sets optimized for one of the other experiments, while the classifying model is learned from synthetic data belonging to the experiment to be analyzed. A second investigation then additionally exchanges the data source from which the classifying model is learned with synthetic data from all *other* experiments. Its goal is answering the question of how much can be learned for a new experiment from examining synthetic ground truth of previous experiments.

Cross-Experiment Generalization Performance of Parameter Sets

For the first investigation, which is concerned with the cross-experiment generalization performance of parameter sets alone, the detailed test methodology is as follows: Detector and classifier parameters optimized for each experiment in Table 7.1, are applied to all other experiments, and to the original experiment as a baseline comparison. The remainder of *SynOpSis* remains unchanged. In particular, the classifying model is learned from the

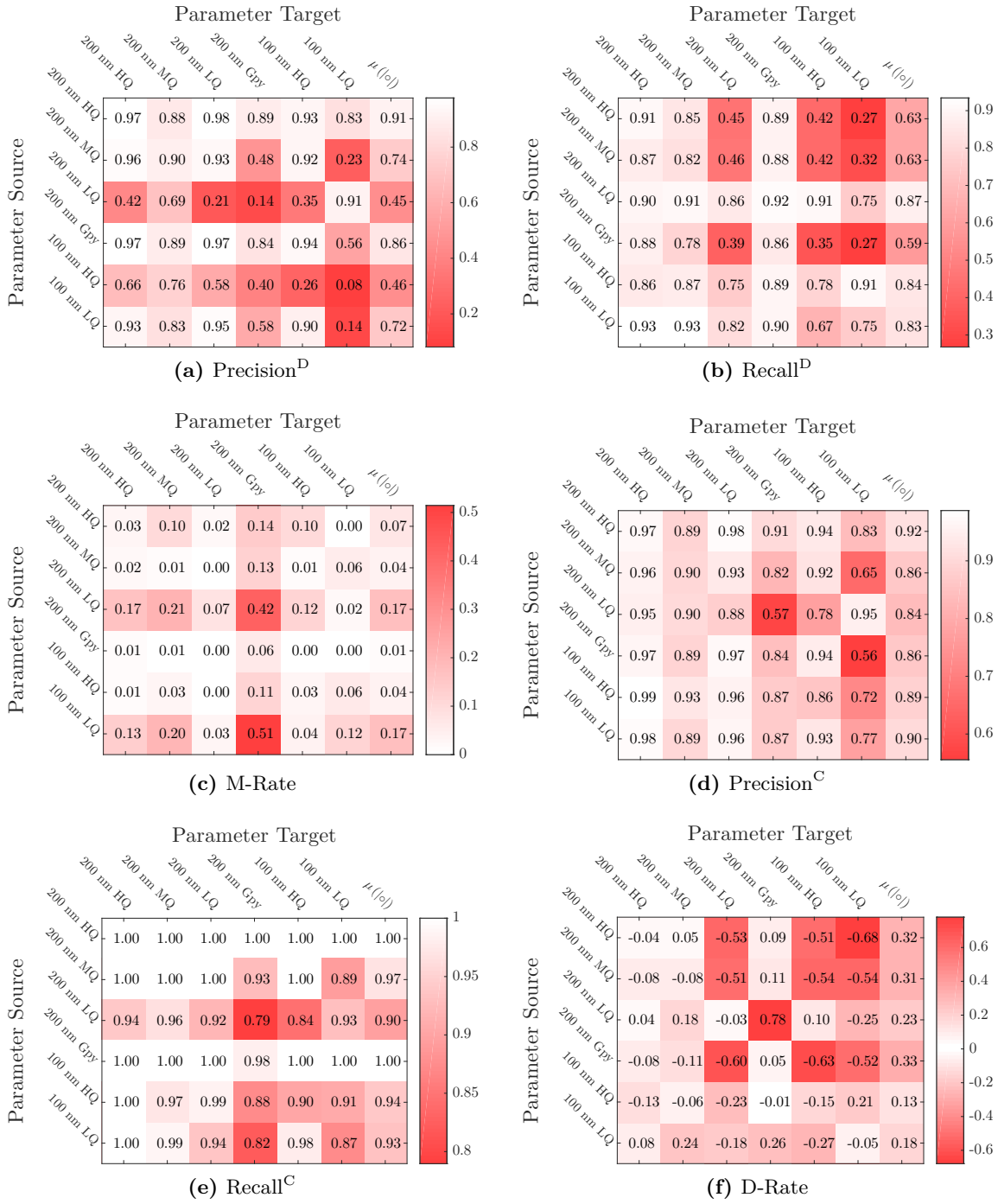


Figure 7.19: Cross-Experiment Generalization Performance – Parameters Sets. Parameter sets optimized for the experiments indicated by the row names ('Parameter Source') are applied in analyzing real sensor data from the experiments indicated by the column names ('Parameter Target'). Each matrix displays the values of a different objective or measure from Section 7.3.1. Poorer values are marked by increasing saturation of the red color underlying matrix cells. This property holds independently of whether an objective or measure is to be minimized (M-Rate), brought to zero (D-Rate) or maximized (all other). The rightmost column displays the mean absolute value of the six entries in the respective same row. It allows to quickly assess whether parameters from the experiment in that row generalize well.

synthetic ground truth created for the original experiment, however on the basis of parameter sets from a foreign experiment. In order to avoid combinatorial explosion, only parameter sets obtained with optimization mode *Global 4* and desirability mode *Scalarizing* are considered (cf. Section 7.5.4) on a single fold of cross-validation, and only the results for real sensor data are reported. The same six objectives and measures as before (cf. Section 7.3.1) are used to assess results quality. However, instead of the box plots used in Section 7.5, objectives and measures are shown in the form of the matrices in Figure 7.19, which are better suited for displaying the exhaustive *combinations* of experiments in this context.

Generally speaking, the semantics of any matrix in Figure 7.19 is as follows: Each row corresponds to the parameter source, i.e. to the experiment from which detector and classifier parameters are taken. Each column corresponds to the parameter target, i.e. to the experiment to which those parameters are applied. One exception to this rule is the '$\mu(|\circ|)$'-column, which shows the mean absolute value over all values neighboring to the left. It thus provides a summary statistic enabling easy assessment of the average merit of parameters from the experiment in the respective row for analyzing the experiments in all columns. Increasing saturation of the red color underlying matrix cells marks values that are poor in relative comparison to the other values in the regarded objective or measure. Here 'poor' means low for objectives to be maximized, high for objectives to be minimized, and far away from zero for D-Rate. Conversely, saturation decreases for better values. This coloring scheme enables to quickly see the hot spots and to assess experiment properties: Red rows correspond to experiments with parameters that do not generalize well in terms of the regarded objective or measure. Red columns indicate experiments that can not be analyzed well by parameters from other experiments.

$\text{Precision}^{\text{D}}$ as displayed in Figure 7.19a is the first measure to be regarded in detail. Values range from 0.08 to 0.98, and the main diagonal reporting the cases of applying parameters to the experiment they were originally optimized for nearly covers this entire range because $\text{Precision}^{\text{D}}$ is not an objective in this optimization. Concerning generalization performance, this result is indifferent and was reported for the sake of completeness only.

Recall^{D} in Figure 7.19b, on the other hand, allows for a statement concerning generalization performance. A prerequisite for this is the following observation: As values on the main diagonal of the matrices in Figure 7.19 were attained for applying parameters to the experiments they were originally optimized for, they can be used as a benchmark. Similarity of the values on the main diagonal to those in any rows of the matrix indicate parameter sets with good generalization performance. As an example, parameters optimized for the experiments 200 nm LQ, 100 nm HQ and 100 nm LQ generalize well in terms of Recall^{D}: Values observed for parameters from these experiment range from 0.67 to 0.93, while the overall range is 0.27 to 0.93, and the rightmost column with the mean values sets these experiments apart from the others as well. While the parameters computed for these experiments generalize well to other experiments (corresponding rows are rather white), these experiments are at the same time the hardest to be analyzed with parameters from other experiments (corresponding columns are rather red). Furthermore, these experiments exhibit the lowest minimum and median SNRs, cf. Table 7.1. Hence, concerning Recall^{D}, parameters optimized for difficult-to-analyze, low SNR experiments were found to generalize well to other experiments, while parameters optimized for the easier-to-analyze, higher SNR experiments failed to generalize to the lower SNR experiments.

M-Rate in Figure 7.19c exhibits values between zero and 0.51, with all but two values being below 0.22. For comparison, the largest value on the diagonal is 0.12. It is attained by experiment 100 nm LQ, which along with 200 nm LQ exhibits the worst generalization performance: Both corresponding rows exhibit an average M-Rate of 0.17. This is correlated with a low SNR and good generalization performance in terms of Recall^{D}, as observed in both experiments. On the parameter target side, the 200 nm Gpy experiment is hardest to analyze with parameters optimized for other experiments in terms of M-Rate: The corresponding column contains the two worst outliers, 0.42 and 0.51. The increased tendency of multiple detection of the same nano-object in experiment 200 nm Gpy is assumed to be due to the different camera and resolution: One nano-object adhesion in 200 nm Gpy covers approximately four times as many pixels as in other experiments. Hence there are four times as many pixels for the detector to respond to, which would ideally all be merged into a single detection. Using parameters optimized for smaller adhesions is likely to involve unsuitable parameters for segmentation (cf. Section 5.6.4), in particular for merging spatiotemporally adjacent detections in order to improve M-Rate.

$\text{Precision}^{\text{C}}$ in Figure 7.19d ranges from 0.56 to 0.99. The worst value on the diagonal is 0.77, while the mean $\text{Precision}^{\text{C}}$ achieved by parameter sets ranges from 0.84 to 0.92, i.e. differences are rather small. Parameters optimized for experiments 200 nm HQ and 100 nm LQ generalize best in terms of mean $\text{Precision}^{\text{C}}$. Experiments 200 nm HQ, 200 nm MQ and 200 nm LQ are most amenable to being analyzed with parameters from other experiments, i.e. the corresponding columns exhibit comparably high values.

Recall^{C} in Figure 7.19e ranges from 0.79 to one, with values from 0.87 to one on the diagonal. Like with $\text{Precision}^{\text{C}}$, experiments 200 nm HQ, 200 nm MQ and 200 nm LQ can be analyzed well using other experiments as the parameter source. Applying parameters from experiment 200 nm HQ or 200 nm Gpy in analyzing other experiments achieves higher Recall^{C} than on the diagonal, which is however to be viewed with skepticism as a classifying model that predicts everything as positive trivially achieves Recall one. Parameters from experiments 200 nm LQ, 100 nm HQ and 100 nm LQ generalize worse than the others, albeit still achieving mean Recall^{C} values of 0.90, 0.94 and 0.93, respectively.

Finally, D-Rate in Figure 7.19f summarizes these results by measuring the relative deviation between the actual and reported number of nano-objects. This is of particular utility because in the investigation above, parameters that were found to generalize well in some objectives did not do so in others. D-Rate exhibits a large deviation between the matrix diagonal, where parameter source and target are identical, and the rest of the matrix. The overall range of values is from −0.68 to 0.78, while that on the diagonal is from −0.15 to 0.05. The scheme of low values in Recall^{D} propagates to D-Rate because missed nano-objects can not be recovered by classification. This can be seen from the red values in (b), converting to negatively signed red values in (f). One exception, which is at the same time the only positive among the worst D-Rate values, arises from applying the parameters of experiment 200 nm LQ to experiment 200 nm Gpy. Here the number of nano-objects in the data is overestimated by an additional ratio of 0.78. The reason is low $\text{Precision}^{\text{D}} = 0.14$ in combination with low $\text{Precision}^{\text{C}} = 0.57$ plus high M-Rate = 0.42. Concerning the other combinations of parameter source and target experiments, however, the classifier demonstrates is merit: The red values in $\text{Precision}^{\text{D}}$ (a), some of which are below 0.2, rarely propagate to red values in D-Rate (f) as this is prevented by good values in $\text{Precision}^{\text{C}}$ (d). As with Recall^{D}, experiments 200 nm

LQ (excluding one outlier), 100 nm HQ and 100 nm LQ provide parameters that generalize best in analyzing data from other experiments.

The results discussed above with respect to Figure 7.19 suggest that running the *Optimization* stage for difficult-to-analyze experiments, yields parameters that generalize well to other experiments: Parameters optimized for the 100 nm experiments yield comparably good values in Recall^{D}, $\text{Precision}^{\text{C}}$ and D-Rate over all other experiments. Parameters optimized for easier-to-analyze experiments generalize worse, particularly with respect to the difficult-to-analyze experiments. Hence, if cross-experiment generalization of parameter sets is desired, e.g. because time does not permit conducting a new run of the *Optimization* stage, difficult-to-analyze experiments should be used as the parameter sources. Good performance is achieved particularly if the parameter target experiment is easier-to-analyze.

Simply using one single parameter set, optimized for the most difficult experiment available, would render optimization for easier experiments unnecessary, if the attained performance was good enough. Conjectures about the associated impact on performance can be formed by comparing the main diagonals of the matrices in Figure 7.19 to the respective rows considered as the parameter sources. Doing so, using e.g. experiment 100 nm LQ as the parameter source, reveals that particularly for Recall^{D}, $\text{Precision}^{\text{C}}$ and Recall^{C}, parameters optimized for the respective original experiment are in some cases beaten by those obtained from generalization. However, conducting per-experiment optimization pays off by a large margin in terms of M-Rate and the summary measure D-Rate: Here, values on the diagonal are considerably closer to zero than those attained by generalizing parameters from experiment 100 nm LQ. Therefore, to ensure the highest quality analysis results, it is generally recommended to run the *Optimization* stage of *SynOpSis* separately for each experiment.

As a conclusion, however, if a new *PAMONO* experiment is conducted, and suitable parameters are to be determined, the parameters that were optimized for previous experiments are a good starting point, as demonstrated by the generalization performances in Figure 7.19. This is particularly true for parameters optimized for difficult-to-analyze previous experiments. One way of incorporating these parameters into a new optimization is adding them to the initial population in the GA.

Cross-Experiment Generalization Performance of Data Sources

In the previous investigation, the cross-experiment generalization performance of parameter sets was examined, which was conducted by applying foreign detector and classifier parameter sets in analyzing an experiment. The classifying model, however, was learned from synthetic ground truth created for the experiment under consideration. Now this investigation serves to explore, whether also the classifying model can be learned from foreign synthetic ground truth: Besides using foreign parameters as before, the classifying model is learned from all *other* experiments, and not from the experiment to be analyzed. Hence, this investigation aims at finding out how much can be learned from the ground truth of previous experiments in classifying examples from a new experiment.

The setup of this investigation is as follows: In learning a classifying model from all other experiments, the same image processing in terms of detector parameters is applied to all data entering the model, as well as to the experiment to be analyzed. Therefore, the feature vectors comprising the model and those to be classified by it are all computed based on images that underwent the same processing in terms of background elimination, denoising,

time series classification and segmentation. The previous investigation determined that the parameters optimized for experiment 100 nm LQ generalize well over the other experiments. Hence these parameters are the only parameters employed during the entire investigation. For each experiment, a classifying model is learned from the synthetic ground truth of all other experiments. This model is then used to classify the real sensor data of the experiment under consideration. Then the quality measures from Section 7.3.1 attained by this analysis are evaluated in order to assess cross-experiment generalization performance of data sources.

Corresponding results are shown in Figure 7.20, which is completely analogue to Figure 7.12b, plotting measures and objectives as attained for the real sensor data from Table 7.1 over the three folds of cross-validation. Comparing Figure 7.20 to Figure 7.12b means comparing it to the case where *SynOpSis* optimization is applied individually to each experiment, and synthetic ground truth specific to that experiment is used in learning the classifying model. Like with the matrices in Figure 7.19, the most apparent differences are severe increases in M-Rate and D-Rate. Increases in M-Rate are due to the parameters being optimized for the low SNR experiment 100 nm LQ, which requires highly sensitive parameters in order to detect the faint nano-objects in the data. The constituents of the increase in D-Rate can be seen in the other measures in Figure 7.20: $\text{Precision}^{\text{D}}$ decreases, thus there are more FP detector responses; increased M-Rate adds repeated TPs to that; finally, decreased $\text{Precision}^{\text{C}}$ means that fewer FP detector responses can be filtered out, resulting in overestimations of the numbers of nano-objects, and hence large positive values of D-Rate. Furthermore, variability over folds increases in all objectives and measures, compared to Figure 7.12b.

Note that comparing Figure 7.20 to Figure 7.12b does not isolate the effect of learning the classifying model from foreign ground truth, but regards it in conjunction with using foreign parameters for the detector and classifier. A more isolated inspection can be conducted by comparing Figure 7.20 to each bottom row of the matrices in Figure 7.19 because these rows already incorporate the loss incurred by applying parameters optimized for experiment 100 nm LQ to the other experiments. Hence the only difference between those rows and the results in Figure 7.20 is that for the latter the classifying model is learned from foreign data (and that three folds of cross-validation are shown). Therefore, in this comparison only the measures $\text{Precision}^{\text{C}}$, Recall^{C} and D-Rate, i.e. those affected by the classifier, are of interest. Losses incurred by the change in classifying models are most pronounced in $\text{Precision}^{\text{C}}$ and D-Rate, while Recall^{C} remains relatively unaffected. Concerning experiments, 200 nm Gpy and 100 nm LQ degrade the most severely, with values of D-Rate beyond one[9]. This is not surprising because the change in employed camera in experiment 200 nm Gpy and the low signal quality in experiment 100 nm LQ set these two experiments apart from the others in terms of the visual appearance of nano-objects in the images. Learning a suitable classifying model from the other experiments is not successful here, which manifests as low values in $\text{Precision}^{\text{C}}$, entailing high D-Rate. Successful learning requires experiments to be more similar: Experiments 200 nm LQ and 100 nm HQ attain values in D-Rate that are of use in practice. With reservations, this also holds for experiments 200 nm HQ and 200 nm MQ, where the benefit of not requiring synthetic ground truth that is specific to the considered experiment can be traded off for the necessity of manually sorting out FP nano-objects after classification.

[9] D-Rate values for experiment 200 nm Gpy are 0.81538, 3.14872 and 3.62051, while those for experiment 100 nm LQ are 0.16071, 0.85714 and 1.57143.

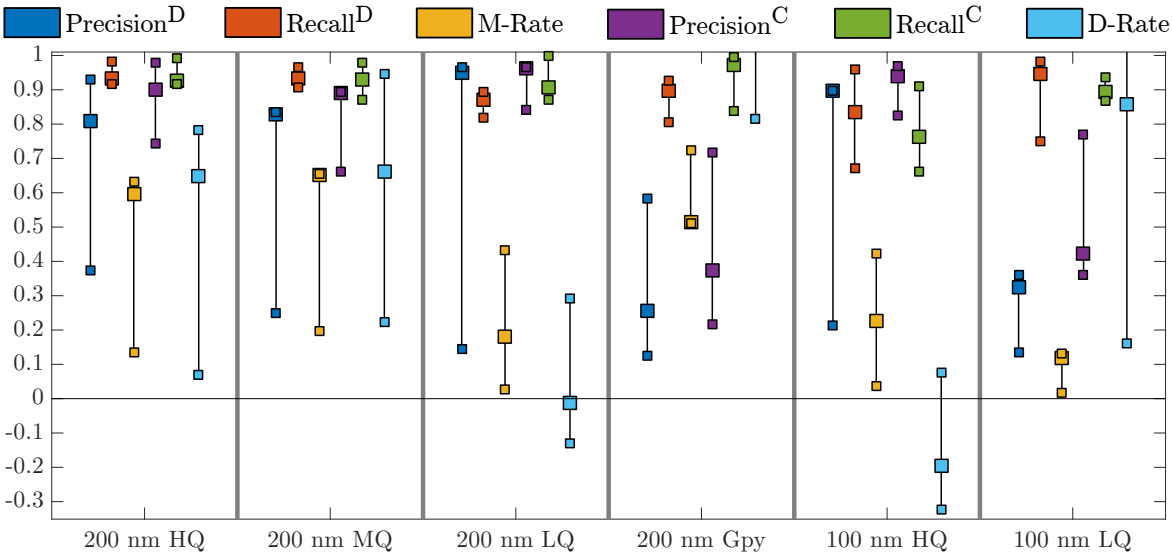


Figure 7.20: Cross-Experiment Generalization Performance – Data Sources. Results shown here are analogue to those in Figure 7.12b: Three folds of the measures and objectives from Section 7.3.1 are plotted over the real sensor data recorded in the experiments from Table 7.1. Detector and classifier parameters optimized for experiment 100 nm LQ were used in all cases, and for each considered experiment, the classifying model was learned from the synthetic ground truth of all respective *other* experiments. The two truncated D-Rate values in the context of experiment 200 nm Gpy are 3.14872 and 3.62051, while the one truncated in experiment 100 nm LQ is 1.57143. A detailed discussion of these results and comparisons to Figure 7.12b and Figure 7.19 are given in the text.

Summary

The two investigations conducted in this section evaluated the cross-experiment generalization performances attained by parameter sets alone, as well as in conjunction with the data sources used for learning the classifying model. Analysis qualities observed in both investigations are clearly beaten by running the *Optimization* stage of *SynOpSis* individually for each experiment, which is not a surprising result because parameters and the classifying model specifically target the experiment to be analyzed in the latter case. Parameters optimized for more difficult-to-analyze experiments could be identified as generalizing better than those optimized for experiments with higher SNR. Generalization performance was good enough to motivate using parameter sets optimized for other experiments in the initial populations of experiment-specific optimizations. Classifying models learned from the synthetic ground truth of foreign experiments perform acceptably, if the visual appearance of nano-objects does not differ too strongly from the experiment under consideration, which is the case for nano-objects observed in similar setups of the *PAMONO* sensor. If a loss in quality as observed in the two investigations is tolerable, the first investigation states that optimizing parameters can be omitted, while the second one states that creating synthetic ground truth for new experiments can be omitted, which are the two most time-consuming aspects of the *SynOpSis* approach.

Chapter 8

Conclusion and Future Work

Contents

In the previous chapter, the performance of the *SynOpSis* approach has been validated on real *PAMONO* sensor measurements. Its capabilities, limitations and properties have been investigated and presented quantitatively. Now Section 8.1 provides conclusions to Chapter 7 and to the overall thesis by revisiting the claims of contributions from Chapter 1 in the light of the evidence produced up to here. Section 8.2 addresses open questions and highlights possible directions for future research that can be conducted based on the findings of this thesis. Both sections share the same division into the topic of *PAMONO* data analysis, followed by considering a broader context.

8.1 Conclusion

PAMONO Data Analysis

In the course of this thesis, a parameter-optimized, model-based method for *PAMONO* data analysis was developed and evaluated. *PAMONO* indirectly detects nano-objects, e.g. biological viruses, by means of optical microscopy, exploiting the Surface Plasmon Resonance (SPR) effect (Chapter 2). It could be shown that automatic analysis of *PAMONO* sensor data is feasible in practice, achieving high quality in terms of nano-object counting (Section 7.5.5) and results specificity (Section 7.5.7). Reliable particle count estimates allow conclusions about particle concentration in the sample, due to their proportionality, as established by Gurevich et al. [GTÜ+11]. The proposed analysis process keeps up with the real-time capability of the *PAMONO* sensor itself (Section 7.5.8), providing results and enhanced images to the sensor operator while the measurement is taken. It is targeted at energy-efficient execution in portable devices [LST+13a; LSW13; LMS+14; NLE+15], thus constituting a software-backend for developing a portable *PAMONO* sensor unit. Very low Signal-to-Noise Ratios (SNRs), even below two, can be handled by the developed methods, enabling to reliably detect nano-objects down to 100 nm in diameter (Figure 7.12b).

On the methodological side, the *SynOpSis* approach (Chapter 3) demonstrates the benefit of automatic parameter optimization in configuring an image processing pipeline, using *PAMONO* as the example application scenario. *SynOpSis* utilizes data synthesis to provide ground truth for parameter optimization, the results of which are employed in real data analysis. Its ability to adapt algorithm choice and parameters to varying experimental conditions was validated in the context of the proposed detector and classifier for *PAMONO*

data analysis (Chapters 5 to 7). Achieved analysis qualities correlate with expectations drawn from the physical parameters of the sensor setup: Employing gold layers of lower quality results in larger errors in estimated nano-object counts and lower specificity (Tables 7.4 and 7.5). The establishment of this relation means taking a first step towards establishing the applicability of *SynOpSis* in finding an optimized setup of the physical parameters of the *PAMONO* sensor: The knowledge that changing a certain physical parameter leads to improved/worsened analysis quality can be used in advancing the sensor itself. More experiments concerning this topic are needed for confirmation (Section 8.2). In particular, the link between physical parameters other than gold layer quality and the obtained data quality needs to be thoroughly investigated.

Concerning techniques for *PAMONO* data analysis, the interplay of a highly sensitive detection followed by machine learning-based classification for enhancing precision was demonstrated to be successful (Section 7.5). A Random Forest learner with prior class-balancing and using the full set of presented features (Section 6.2) was determined to work best in the examined *PAMONO* experiments (Section 6.7). Optimized detector and classifier parameters as well as the classifying model determined by means of machine learning were verified to transfer well from the synthetic ground truth they are based on, to the real input data that is to be analyzed (Section 7.5). This transferability validates the proposed *PAMONO* signal model (Chapter 4) in terms of the tasks it serves in *SynOpSis*: Parameters can be optimized on synthetic data and a classifying model can be learned from it. Rephrased in terms of machine learning, this transferability means that the generalization performance of parameters and classifying model from synthetic to real data was shown to be high. Their generalization performance *across* different *PAMONO* experiments was found to be lower (Section 7.7), however still achieving serviceable results, particularly with parameters that were optimized for difficult-to-analyze experiments.

Performance estimates for analysis results on real sensor data were given, and their quality was evaluated (Section 7.5.6): For experiments with 100 nm nano-objects, estimation errors between −72% and +47% were incurred in nano-object counts, which is too large for practical purposes. Note, however that these numbers relate to errors in *performance estimates*, while the actual *analysis quality* for 100 nm nano-objects was considerably better (Table 7.4). For 200 nm experiments, errors in performance estimates ranged from −7% to +12%, allowing a lab operator to tightly assess the potential error incurred in reported nano-object counts. Given the examined setup of *SynOpSis* for *PAMONO* data analysis (Section 7.3), it can be readily applied in everyday-lab practice: No familiarity with the underlying algorithms, machine learning or optimization is assumed in the operating lab personnel. Furthermore, only minimal manual segmentation effort is required: A small number of examples of the nano-objects to be counted is needed to seed the signal model and thus enable the *Synthesis* stage to generate datasets with known ground truth that are highly similar to real data (Section 4.3). Everything else, including parameter optimization and training of a classifying model, is handled automatically by *SynOpSis*. Hence, solely domain knowledge is required, which is traded in for automatic handling of everything else.

Broader Context

Polystyrene nano-objects with 100 nm in diameter can be successfully counted using *SynOpSis* and *PAMONO*. These nano-objects are about the size of the Human Immunodeficiency Virus

(HIV), which is approximately 120 nm in diameter. HIV Virus-Like Particles (VLPs) have already been successfully counted by means of *SynOpSis* and *PAMONO* [STM+15]. VLPs are non-infectious virus envelope proteins lacking the genetic information for reproduction [GA06]. Hence they are frequently used as a safe proxy in experiments investigating virus properties other than reproduction, e.g. their binding behavior as in the case of *PAMONO*. Being able to detect HIV on the basis of the virus itself, as opposed to detecting HIV antibodies as in regular rapid HIV tests [GBP+06], may help in minimizing the diagnostic gap observed in the context of HIV: The antibody concentration required for validity of rapid HIV tests is at the earliest reached three months after an infection. During those three months, virus concentration in the affected bodily fluids is particularly high, resulting in a severely increased risk of transmitting the virus further. Narrowing this three-months diagnostic gap by detecting the high concentration of viruses instead of the low concentration of antibodies is a desirable goal. This requires further *PAMONO* experiments involving HIV or HIV VLPs within or extracted from human bodily fluids like blood. The same holds for other viruses to be detected with *PAMONO*, in order to develop rapid real-time-capable tests for the respective virus-transmitted diseases. *SynOpSis* can serve in accelerating execution of the arising large number of experiments.

In such diagnostic contexts, results specificity must be particularly high to avoid false positive experiments. A false positive experiment means that at least one virus is reported for a full dataset of *PAMONO* images that does not contain any viruses. Hence, in such a case not a single false positive classifier response is allowed over several thousands of images, in order to avoid a false positive experiment. Expressed in terms of Table 7.5 from Section 7.5.7, Specificity one is mandatory here. A step in this direction can be taken by extending the *Optimization* stage accordingly: Specificity on a virus-free time series of images can be added as a further objective or even as a hard constraint by assigning desirability zero to individuals with Specificity below one. The facilities for doing so are already present in *SynOpSis*, and a virus-free measurement is already part of the experimental protocol from Section 4.3.1. An additional such measurement should be conducted in order to assess Specificity on data not used in synthesis.

8.2 Future Work

PAMONO Data Analysis

The discussion in the previous section already touched upon some points that naturally lead up to further investigations and research. **Expanding the fields of application** of *PAMONO* is the first of these points. On the sensor side, the gold layer can be modified such that it can **distinguish multiple types of viruses**: Coating different areas on the sensor surface with different antibodies makes each such area selective for a certain type of virus. Hence virus types can be distinguished based on the position of their attachment. Furthermore, *PAMONO* experiments with **other types of nano-objects, e.g. fine dust and other particulate matter** from car or industrial exhaust, can be performed. A possible method for attaching such nano-objects to the surface is electrostatic charge. On the side of *SynOpSis*, the most obvious goal is **lowering the size of detectable nano-objects** in order to make the realm of viruses smaller than 100 nm accessible for *PAMONO*. In particular, this means pushing the breakdown point, i.e. the transition from under- to overestimation of nano-object counts

(determined in Section 7.5.5), down to smaller nano-objects or lower SNRs, respectively. A possible idea for enhancing the algorithms in this direction is merging the detector and classifier by running the latter on the per-pixel level instead of the per-polygon level. The spatial and spatiotemporal features from Section 6.2 are defined in terms of per-pixel feature maps which are good candidates for training the classifier from. Such a merged approach may be more robust to noise than the sequential, separate execution of pixel and polygon classification (Sections 5.4/5.5 and Chapter 6, respectively). The reason is that more data is considered in the merged approach: Feature vectors are no longer aggregated over polygon area, but polygons are created from a pixel classification exploiting these feature vectors. In this context, developing further features may prove beneficial. Using a GPU-accelerated Random Forest [Lib15b; Bre01] as the classifier may help in maintaining real-time capability in the merged approach.

While the *PAMONO* data analysis process presented in this thesis is real-time capable, the optimization to be conducted in case of larger changes in physical sensor parameters constitutes a time-consuming offline step (Table 7.6). Hence another direction for future research is the **acceleration of optimizations** which can for example be done by using one or a combination of the following ideas:

- A lower-dimensional parameter space can facilitate optimization by reducing the number of required samples. Hence **removing irrelevant parameters** can accelerate optimization. Irrelevant/less relevant parameters were identified in Section 7.6.1, which can be exploited as follows: Boolean switches with definitive votes (e.g. 90% versus 10%) can simply be fixed at the values chosen by the majority of individuals on the Pareto front. Furthermore, parameters of the algorithms that are deactivated by those boolean switches can be removed. Finally, parameters of active algorithms that exhibit preferred values in individuals on the Pareto front can either be set constant, or their examined range can be narrowed, allowing for fine-tuning only. The set of remaining parameters can furthermore be reduced by determining, whether it has low effective dimensionality, i.e. only few parameters actually have a pronounced influence on objective values. A method to this end was proposed in [BB12]: The idea is to fit a Gaussian Process (GP) to the evaluated samples in parameter space, similar to GP-based meta-modeling [BBB+11]. Then the variables of this GP model fit can serve as measures of sensitivity of the response surface (and hence the objective) to changes along the dimensions of parameter space. Such an approach can help in answering the question whether a parameter assumes evenly distributed values in well-performing individuals because the parameter is simply irrelevant for objective values or because its optimal values are evenly distributed throughout its range. Both cases result in high entropy histograms of parameter values (cf. histograms in Section 7.6.1). Hence examining histograms alone can not answer the question of parameter irrelevance, while analysis of effective dimensionality can. The lower the number of parameters to which the response surface is sensitive, the lower the effective dimensionality of parameter space, and hence the fewer parameters are worthwhile to be optimized. With regard to *PAMONO*, the following three questions should be investigated: Firstly, does the *PAMONO* optimization problem have low effective dimensionality for a given experiment and objective? Secondly, are effective dimensions the same over all/most experiments? Thirdly, are effective dimensions the same over objectives? If all questions can be answered with

'yes', default values can be fixed for the irrelevant dimensions, and optimization can focus on the relevant dimensions only.

- **Meta-modeling** [BLP10; KKF+11; HHL11; BBB+11; BMT+12] can accelerate optimization by creating an approximate model function of the response surface to be optimized. The model function can be evaluated and optimized more quickly and easily, earning a merit if it approximates the response surface well enough to provide a good sense of the direction of improvement in parameter space. Meta-modeling can be carried out separately for each optimization, or it can be used to integrate optimization results over multiple experiments. Success of the latter is particularly likely if the questions asked at the end of the paragraph about removing irrelevant parameters can be answered with 'yes'.
  Preliminary investigations on the applicability of meta-modeling in *SynOpSis* for *PAMONO* were carried out in Section 7.6.2. In this context, Table 7.7 demonstrates that accumulating results over multiple optimizations incurs a benefit over creating a new meta-model from scratch for each optimization. This suggests suitability of meta-modeling for accumulating knowledge gained during optimizations over multiple *PAMONO* experiments. Particularly in face of the scenario of a distributed *collaborating* network of *PAMONO* sensor nodes [LKD+14], this is an important finding, which should be explored further. Part of this investigation should quantify how much can be gained from integrating optimization results of different experiments in a single meta-model. The resulting quantity is a measure of what can be gained by sensor collaboration.
  Concerning the topic of meta-modeling, it should be pointed out that its base versions only cover a single-objective setting, but that multi-objective extensions are available [KN08]. The former can be used in optimizing the Desirability Index (DI), while the latter can serve in optimizing Desirability Functions (DFs) or raw objectives. As a further preliminary result, Table 7.7 showed three cases where the DI could be predicted better than three of its four constituent objectives, hinting at single-objective meta-modeling as a good starting point.
  Note that all regression models evaluated in Table 7.7 were fit to the data *after* all optimizations entering the model were finished, and that quality was measured in a cross-validation. Hence the reported values represent the qualities those meta-models can achieve *at the end* of optimizations. Therefore, this preliminary investigation should be extended by examining the performance of single-objective and multi-objective meta-models *during* optimization, i.e. in a sequential fashion [HHL11].
  As a summary, the topic of meta-modeling in *PAMONO* poses three questions: Firstly, how well can meta-modeling approximate the response surface? Secondly, by how much can optimization be accelerated with it? Thirdly, how much can a network of cooperating sensor nodes benefit from it? The topic of collaborating sensors discussed above will be picked up again further below, providing more details.

Besides acceleration of optimizations, another topic worth investigating is whether the manual segmentation of nano-objects used to seed the signal model can be made dispensable: Towards the end of sensor prototype development, physical sensor parameters converge, and large amounts of data from previous experiments are available. In this scenario, the following ideas for **obtaining a classifying model with less or no interaction** are conceivable:

- One idea is updating an existing classifying model incrementally via **active learning** [Set10]. In such an approach, the interactive part can e.g. be crowdsourced to a platform like Amazon's 'Mechanical Turk' [BKG11; DK13].
- Furthermore, Section 7.7 demonstrated that learning a classifying model by **combining synthetic data from previous experiments** yields acceptable results if the nature of the experimental data to be analyzed does not differ too much from the nature of the data constituting the model. Hence this idea strongly benefits from converging physical parameters in *PAMONO* sensor development. It can be used as a starting point for fixing a classifying model in a non-interactive way. One key to improving cross-experiment generalization performance of such a model is simply the amount of data entering it. If the resulting model becomes too large for real-time application, the condensed k-Nearest Neighbors (k-NN) classifier from Section 5.5.3 can be used to compute models of adjustable size and application speed: The selected number of cluster centers to be computed allows control over the level of feature space detail represented in the resulting classifying model. This number of cluster centers can be adjusted to allow for real-time application of the classifier on the target system. As an example application scenario, consider the case where an embedded system, used for data analysis in a portable *PAMONO* sensor unit, requests a classifying model from a central server. Then that server can compute a condensed k-NN classifier from all currently available data, and in that computation the server can select a number of cluster centers that ensures real-time applicability of the model on the requesting embedded system. This can be done rapidly due to the streaming-capable BIRCH Meets Coresets (BICO) clustering [FGS+13] which can handle very large inputs, allowing the integration of data from many experiments. The typically unbalanced classes in the input can be balanced during clustering. k-NN can model intricate and disparate class boundaries which empowers it to integrate data from very different *PAMONO* experiments. The requirement, however, is that feature values across experiments should not overlap too much *over different* classes.

A conceptually very different approach that can be explored for the *PAMONO* data analysis task is **deep learning** [KSH12]: This technique learns not only a classifying model, but also feature extraction itself, from the data. Unlike the engineered, application-specific features used by *SynOpSis*, the feature extraction of deep learning is encoded in the front layers of the same deep Convolutional Neural Network (CNN) used as the classifying model. Hence features themselves are learned as part of the supervised learning procedure optimizing the CNN. Note that compared to *SynOpSis*, deep learning replaces/augments the tuning of high-level algorithmic parameters with the optimization of a considerably larger number of low-level parameters defining the function computed by the neural network. Deep learning techniques have recently raised the bar in per-pixel labeling of natural images [FCN+13]. However, this task typically exhibits more structure but also many more classes than *PAMONO* data analysis, so the suitability of deep learning for *PAMONO* needs careful investigation.

A further direction of future research is the **collaborative sensor network** [LKD+14] already touched upon in the context of meta-modeling. One key motivation behind this networked scenario is the desire to **conserve energy** in *PAMONO* sensors that are spatially distributed in remote locations with no energy sources available. A practical application is e.g. large-scale monitoring of viral propagation in rural environments. The network can serve

to offload [KL10; TC13] tasks from sensor nodes with lower battery level to nodes with higher battery level, thus balancing energy levels across the network. Furthermore, it can be used to offload tasks to a central server with an energy source available. The goal is to keep the sensor network running as long and as completely as possible with a single charge of batteries. Data can be transferred e.g. via the Long-Term Evolution (LTE) network, which is beneficial if the energy used for communication is lower than the energy required for performing the task locally. As an example, energy-intensive optimization tasks can be offloaded to a server that is not subject to energy-constraints. Thus optimizations[1] can be carried out on powerful GPU servers, which accelerates the adjustment of sensors to changes in their physical parameters and helps in adhering to the energy constraints imposed by remote operation. Energy conservation, however, is not the only motivation behind the networked scenario: The aspect of collaboration furthermore encompasses **sensors collaborating in terms of knowledge** about the data generated by each sensor. One way of storing and utilizing the knowledge created in collaboration is a meta-model [BLP10; KKF+11; HHL11; BBB+11; BMT+12], as already outlined above. A meta-model can accumulate knowledge from all nodes in the sensor network, and hence each node can benefit from the data provided by all other nodes. The meta-model collects optimization results over multiple experiments, and by abstracting and learning from these results it stores knowledge about the *PAMONO*-related optimization problem as gained from all collaborating sensor nodes. This knowledge can be used to predict the performance of candidate parameter sets during optimization, which is thus accelerated. It can additionally help conserving energy during optimizations carried out locally on a sensor node: If the meta-model predicts low performance with high certainty for a candidate parameter set, that set need not actually be evaluated. Hence low energy consumption is traded off for possible inaccuracies incurred by predicted objective values. Having performance of a parameter set predicted by a meta-model on a central server incurs only minor communication cost in case of the 31 parameters to be optimized for *PAMONO* data analysis. An ultimate goal is predicting suitable algorithmic parameters themselves from instance-based features of the time series of input images. Research should go into constructing suitable features to this end.

A last aspect to be mentioned in the context of *PAMONO* data analysis is the possibility of **letting *SynOpSis* results drive the improvement of the sensor**: *SynOpSis* provides a quick way of obtaining analysis results for new experimental setups of the *PAMONO* sensor, e.g. during prototype development. The performance estimates produced by *SynOpSis* can be regarded as a measure of difficulty of the analysis task. This difficulty can be used as an objective driving the alteration of the physical sensor parameters, with the goal of decreasing difficulty of analysis by providing the best-possible data. This can guide sensor development in an iterative process of improvement, converging in an optimized sensor setup, thus advancing the sensor itself. While realizing this goal is highly desirable, more experiments are needed to fully confirm and back up the applicability of *SynOpSis* in driving *PAMONO* sensor development: A first step in this direction can be taken by varying physical sensor parameters that are *known* to affect data quality. This should be done in a systematic and isolated fashion, and the correlation between the known physical parameter quality and *SynOpSis* performance estimates should be quantified. A high correlation suggests that *SynOpSis* performance estimates can be used to traverse the search space of physical sensor parameters.

[1] Seeding the signal model with nano-object examples to synthesize ground truth for optimization (Chapter 4) can be done by means of a segmentation app, operated in the remote location.

Broader Context

SynOpSis was conceptualized with regard to the abstract task description presented in Section 3.1. This description steps back from the concrete task of *PAMONO* data analysis and identifies the basic abstract building blocks required for solving tasks of this type. Chapters 4 to 6 subsequently developed concrete *PAMONO*-specific versions of these building blocks. *SynOpSis* and these building blocks are designed in a modular fashion, hence individual parts can be easily replaced or adapted with regard to different fields of application, while the overall approach of synthesis, optimization and analysis can be reused. Therefore, the remaining part of this thesis lists and examines **other possible fields of application for *SynOpSis***.

The most obvious candidate domains for adapting *SynOpSis* to are **other microscopy modalities**. Examples are the detection and classification of objects of interest in **histological slices** [HBR+08; HBR+12; ALN+12; WHS+12], **phase contrast microscopy** [PKC09; YBC+10; ALN+12], **differential interference contrast microscopy** [YBC+10] or **fluorescence microscopy** [TRS+02; Oli02; ZFS+07; JZK+07; SLN+09; ALN+12]. Many of these modalities are frequently used for cell detection, which may involve multi-class cell-classification. Possibilities for multi-class extensions of *SynOpSis* were sketched in the footnotes of Chapter 3.

Less obvious candidate domains for applying *SynOpSis* reside on a scale very different from that of microscopy: These domains are from the context of astronomy, in particular from **astroparticle physics**. Generally speaking, the task is detection and classification of astrophysical particles from extraterrestrial sources. Some of these particles serve as an indirect proof in detecting e.g. active galactic nuclei [DE13]. Hence, depending on whether one is concerned with the cause (astronomical objects) or with the effect (astroparticles), the examined scale is either considerably larger or smaller than in *PAMONO* data analysis. Three possible observation tasks for applying *SynOpSis* in astroparticle physics will be summarized now, including an identification of the required modifications. As a conclusion from this, the abstract requirements for making *SynOpSis* applicable in a new domain are listed.

The first example is constituted by the **Major Atmospheric Gamma Imaging Cherenkov Telescopes (MAGIC)** [VFB+14]: Like the *PAMONO* sensor, MAGIC delivers data with two spatial and one temporal dimension. Each event recorded by MAGIC shows an atmospheric light shower. These light showers are caused either by extraterrestrial γ-ray sources or by hadrons interacting with the atmosphere of the earth. The data analysis task is firstly detection of the spatial coordinates affected by a light shower and secondly classification of the cause: Spatiotemporal characteristics of γ-ray-initiated light showers are different from those of hadron-initiated light showers, hence the telescope can be used to survey the night sky for extraterrestrial γ-ray sources like active galactic nuclei. The composite task of detection and classification ideally matches the abstract task description of *SynOpSis* (Section 3.1). Furthermore, a *Synthesis* stage is already available: Majumdar et al. [MMB+05] developed a method for synthesizing both, γ-initiated and hadron-initiated light showers. These Monte Carlo (MC) physics simulations can be used to create synthetic ground truth for optimizing a detector and for training a classifier. Denoising and per-pixel time series classification are subtasks arising in MAGIC data analysis as well, and application-specific features for event classification have been defined in terms of the Hillas parameters [Hil85]. These prerequisites make MAGIC data analysis an ideal candidate for investigating generality of the *SynOpSis*

approach. Furthermore, it opens an alley for future research aimed at determining whether MAGIC data analysis can benefit from the capabilities of *SynOpSis*. The same holds for the **First G-APD Cherenkov Telescope (FACT)** [DT14] which, as an enhancement of MAGIC, delivers the same type of data at a different resolution.

Spatiotemporal data with a third spatial dimension is provided by the **IceCube Neutrino Observatory** [Ruh13], which is a further possible domain for applying *SynOpSis* in astroparticle physics. Its goal is estimating the spectrum of atmospheric neutrinos[2], which have to be separated from atmospheric muons, both of which trigger events in a cubic-kilometer-sized detector-array below the Amundsen–Scott South Pole Station. MC-based simulators for both, neutrinos and muons, are already available [Oli+16; CGK+92], and software for event detection and classification [Ruh13] can be integrated into *SynOpSis* by adapting the interfaces described in Sections 3.5 and 3.6.

Regarded abstractly, the **common denominator for applicability of *SynOpSis*** is a signal model for synthesis and base ideas for realizing a detector, feature extraction and a classifier. Concerning classification, the emphasis lies more on feature extraction than on the classifier itself, because the design of the latter is based on a rather general machine learning process. Due to the modular structure of *SynOpSis*, its building blocks can be filled with application-specific methods as necessary. Assessing the quality attainable by such generalizations of *SynOpSis* and comparing it to the state of the art in the respective field is left open for future research. Doing so may establish *SynOpSis* as a method for facilitating/accelerating analyses in prototypical sensor development beyond *PAMONO*.

[2]To be fully exact: Neutrino-induced muons have to be separated from atmospheric muons [Ruh13], and the neutrino-induced muons serve as an indirect proof of the neutrinos that caused them.

Appendix A

Performance Measures and Equivalences

This appendix provides formal definitions of a selection of performance measures that can be used to assess the quality of detection and classification results. In the definitions given here, only two-class classification is considered, while some of the measures have multi-class generalizations. All of the presented measures can be used in the context of classification. Some, however, are not readily applicable in detection, which is indicated in the discussion of the individual measures. Besides that, equivalences and synonyms of the respective measures are listed. Depending on application requirements, one or more of these measures may serve as objective functions in the *Optimization* stage of *SynOpSis* or other approaches.

Throughout the definitions, the abbreviations in Table A.1 are used to denote the entries of the confusion matrix of the computed prediction, where 'prediction' is a placeholder for 'detection' and 'classification', respectively.

- **Recall:**
  Recall [HG09] measures the ratio of correctly predicted positive patterns among all actually positive patterns and hence the ratio of actually positive patterns that could be found. Thus Recall is synonymously denoted as Sensitivity.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{A.1}$$

- **Precision:**
  Precision [Pow11] measures the ratio of correctly predicted positive patterns among all patterns predicted to be positive and is hence synonymously denoted as Positive Predictive Value (PPV) [AB94].

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{A.2}$$

Table A.1: Confusion Matrix Entries and Abbreviations. The table shows how the entries in the confusion matrix of a prediction are abbreviated in the definitions of performance measures. The term 'prediction' is a placeholder for 'detection' and 'classification', respectively.

		Ground Truth	
		Positive	Negative
Prediction	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

- F_β **score:**
  F_β score score [Chi92] is defined as the β-weighted harmonic mean of Precision and Recall defined above. For $\beta = 1$, it is also denoted as F_1 score, F-score, F-measure or Positive Agreement [KP03].

$$F_\beta = \frac{(1+\beta^2) \cdot \text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} = \frac{(1+\beta^2) \cdot \text{TP}}{(1+\beta^2) \cdot \text{TP} + \beta^2 \cdot \text{FN} + \text{FP}}. \tag{A.3}$$

  It considers all entries of a two-class confusion matrix except for TN, which does not arise in detection tasks [WHS+12; SLN+09]. By β it provides a possibility to balance between Precision and Recall. This makes it a possible choice for single-objective optimization.
- **Specificity:**
  Specificity [Pow11] measures the ratio of correctly predicted negative patterns among all actually negative patterns and hence the ratio of actually negative patterns that could be found:

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}. \tag{A.4}$$

  Thus Specificity is the Recall of the negative class because Equation (A.1) can be transformed into Equation (A.4) by swapping positives and negatives. Specificity can not be applied as an objective in optimizing object detection, where by task definition the number of TNs is always zero because undetected patterns do not result in a detector response [WHS+12; SLN+09]. Hence, in detection, Specificity will always be zero or, in the case of no FPs, $0/0 = \text{NaN}$ (not a number). However, the Specificity measure can be used in the context of classification, cf. e.g. Section 7.5.7.
- **Negative Predictive Value (NPV):**
  NPV [AB94] measures the ratio of correctly predicted negative patterns among all patterns predicted to be negative and is hence negative class Precision.

$$\text{NPV} = \frac{\text{TN}}{\text{TN} + \text{FN}}. \tag{A.5}$$

  NPV can not be applied as an objective in optimizing object detection for the same reason as discussed with respect to Specificity. NPV in detection degenerates to NaN in the case of no FNs and is zero otherwise.
- **Accuracy:**

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{A.6}$$

  Accuracy [Pow11] is the ratio of correctly predicted patterns among all patterns (and thus readily generalizes to more than two classes). It should not be used as the sole performance measure if the data exhibits severe class imbalance because in that case a bad performance in classifying the minority class can always be compensated for by a good performance in classifying the majority class. Hence, Accuracy can ascribe high performance to classifiers that fail in classifying the minority class.

- **Area under the ROC Curve (AUC):**
  AUC [Faw06] is defined as the integral of the Receiver Operating Characteristic (ROC) curve, which is the curve arising from plotting Recall over Fallout, where Fallout = 1 – Specificity. The ROC curve is obtained by applying a sliding threshold to classifier confidences: For each value of the threshold, a new classifier is obtained by assigning a class label to a pattern, based on whether the confidence in its class membership is above or below that threshold. Each such classifier yields one point of the ROC curve by computing its Recall and Fallout value. Computation of ROC and consequently AUC assumes a classifier that delivers confidence values. AUC can not be applied as an objective in optimizing object detection because, as discussed, Specificity is either 0 or NaN, making Fallout either 1 or NaN.

This list of performance measures is far from complete and only serves as an introduction for the two-class case. Further performance measures can be found in the cited literature, especially in [Pow11]. In addition to that, Sokolova and Lapalme systematically survey performance measures for two-class, multi-class, multi-labeled and hierarchical classification problems [SL09], while Sokolova, Japkowicz, and Szpakowicz propose additional measures [SJS06].

Appendix B

Comparison of Wavelet Bases

The wavelet bases available in the *WaveLab* package [CD95; DMS06] were examined for their suitability in the extraction of the translation-invariant (TI) time series features defined in Section 5.5.1. Suitability was evaluated in terms of Accuracy attained in classifying an unseen test set of time series, using the method proposed in Section 5.5. The examined bases were: Haar, Beylkin, Coiflet 1, Coiflet 2, Coiflet 3, Coiflet 4, Coiflet 5, Daubechies 4, Daubechies 6, Daubechies 8, Daubechies 10, Daubechies 12, Daubechies 14, Daubechies 16, Daubechies 18, Daubechies 20, Symmlet 4, Symmlet 5, Symmlet 6, Symmlet 7, Symmlet 8, Symmlet 9, Symmlet 10, Vaidyanathan, Battle 1, Battle 3, Battle 5.

Table B.1 shows the results as attained for the three examined classification tasks after optimizing the feature selection and the number of regarded neighbors in k-Nearest Neighbors (k-NN). All other parameters were chosen as in the first row of Table 5.1 within the main evaluation, cf. Section 5.5.4. Variations in Accuracy over the bases are generally small within a single task, as can be seen from the standard deviations in the last row of Table B.1. The Haar basis performs best in separating the up $\cup$ down class from the background class (column 'Task 2a') and attains the highest mean Accuracy over all tasks (column 'Mean'). Furthermore, Haar-based features are fastest to compute (column 'Time (s)'). For these reasons, the Haar basis was chosen for all experiments conducted in Section 5.5 [SFL+14].

Table B.1: Comparison of Wavelet Bases. Accuracy values on an unseen test set as attained by the examined wavelet bases (column one) are shown for the three classification tasks (columns two to four). Furthermore, the mean Accuracy over the three tasks is reported (column five), and the computation times in seconds, as taken for extracting features from 100000 time series, are displayed (column six). For each numeric column, the best entries are highlighted in bold type, and the column mean value and standard deviation are provided in the last two rows. Among all examined wavelet bases, the Haar basis attains the highest mean Accuracy over all tasks, as well as the highest Accuracy for task 2a. In addition to that, Haar-based features are fastest to compute.

Wavelet Basis	Task 1	Task 2a	Task 2b	Mean	Time (s)
Haar	0.86630	**0.94989**	0.99790	**0.93803**	**3258**
Beylkin	0.86310	0.94344	0.99835	0.93496	3725
Coiflet 1	0.86510	0.94704	0.99865	0.93693	3391
Coiflet 2	0.86460	0.94749	0.99745	0.93651	3457
Coiflet 3	0.86320	0.94554	0.99760	0.93545	3664
Coiflet 4	0.86540	0.94239	0.99730	0.93503	3672
Coiflet 5	0.86530	0.94359	0.99715	0.93535	3717
Daubechies 4	0.86550	0.94599	0.99745	0.93631	3335
Daubechies 6	0.86490	0.94854	0.99835	0.93726	3357
Daubechies 8	0.86380	0.94404	**0.99895**	0.93560	3410
Daubechies 10	0.86320	0.94359	0.99805	0.93495	3493
Daubechies 12	0.86270	0.94464	0.99835	0.93523	3431
Daubechies 14	0.86460	0.94224	0.99790	0.93491	3498
Daubechies 16	0.86300	0.94389	0.99745	0.93478	3475
Daubechies 18	0.86300	0.94299	0.99790	0.93463	3663
Daubechies 20	0.86140	0.94329	0.99790	0.93420	3627
Symmlet 4	0.86580	0.94734	0.99850	0.93721	3407
Symmlet 5	0.86500	0.94254	0.99805	0.93520	3582
Symmlet 6	0.86420	0.94419	0.99730	0.93523	3605
Symmlet 7	0.86420	0.94134	0.99730	0.93428	3654
Symmlet 8	0.86360	0.94134	0.99745	0.93413	3607
Symmlet 9	0.86380	0.94569	0.99805	0.93585	3696
Symmlet 10	0.86680	0.94494	0.99730	0.93635	3716
Vaidyanathan	0.86080	0.94419	0.99730	0.93410	3739
Battle 1	**0.86700**	0.94749	0.99760	0.93736	3754
Battle 3	0.86360	0.94329	0.99700	0.93463	4052
Battle 5	0.86360	0.94194	0.99715	0.93423	4202
Column Mean	0.86420	0.94456	0.99777	0.93551	3600
Column Std. Dev.	0.00148	0.00225	0.00052	0.00112	208

Appendix C

Publications and Author's Contributions

Major parts of the contributions made in this thesis have previously been published in the context of peer-reviewed national and international conferences and journals. In the following, the corresponding publications are listed, arranged by topic. Since all publications were created in collaboration with other authors, the contributions of the author of this thesis (briefly referred to as "the author") to each publication is indicated, as required by applicable PhD regulations.

Signal Model and Synthesis. The signal model for the *PAMONO* sensor was proposed in [SLW+14]. It was devised in collaboration with one of the co-authors (A. Zybin) from ISAS Institute (Leibniz-Institut für Analytische Wissenschaften), who also conducted the *PAMONO* experiments that provided the employed data. The experimental protocol and the proposed methods to compute synthetic *PAMONO* imagery on the basis of the signal model were devised by the author of this thesis. These topics led to Chapter 4. Besides methods related to the signal model, the paper includes the following contributions by the author: An early version of the *SynOpSis* approach is presented, which automatically optimizes detector parameters. The quality of detector results is quantified in terms of two measures, one of which serves as the objective in optimization. Automatic evaluation of measures is enabled by synthetic ground truth. These aspects were developed and realized by the author. An exception is the interface to the Java Genetic Algorithms Package (JGAP) [MRK+11], which was provided by one of the co-authors (P. Libuschewski).

Detector. Methods for detecting regions in *PAMONO* data that are candidates for being affected by nano-object adhesions were presented in [SWL+11; LST+13a; LST+13b] and [SFL+14]. The real-time capable detector pipeline is described in [SWL+11] and extended in [LST+13a]. It was developed in collaboration with P. Libuschewski: *PAMONO*-specific image processing algorithms, as well as techniques for time series classification used by the detector were devised, improved and fine-tuned in collaboration, while their implementation on the Graphics Processing Unit (GPU) is due to P. Libuschewski. The work in [LST+13b] is a German short version of [LST+13a].

An alternative approach to time series classification in the detector was presented in [SFL+14]. It is based on features of translation-invariant (TI) wavelets. The presented approach was devised by the author of this thesis. This includes the design and implementation of the TI wavelet features, an according feature selection and the implementation of condensed k-Nearest Neighbors (k-NN) classification. The latter incorporates the BICO approach [FGS+13] to fast coreset-based clustering, which was provided by a co-author (H. Fichtenberger), who is responsible for everything related to BICO, including a customized

implementation. In this thesis, the detector as a whole is covered in Chapter 5, while TI wavelet-based time series classification is detailed as a part of that chapter, in Section 5.5.

Classifier. Methods for machine learning-based classification of *PAMONO* data were first proposed in [SWL+11]. The contributions of the author to this paper were as follows: A concept was designed and implemented that splits the *PAMONO* data analysis task into a highly sensitive detection of spatiotemporal nano-object candidate regions and a subsequent classification that separates these candidate regions into correct and spurious detector responses, respectively. This concept is part of Chapter 3. The machine learning process realizing the classifier part was designed and implemented by the author. It encompasses evolutionary optimization of classifier parameters and feature selection. This process, its optimization and the feature selection served as the basis for the enhanced classifier presented in Chapter 6. Details of these enhancements will be discussed in the next paragraph.

SynOpSis Approach. The *SynOpSis* approach and its constituents are covered in a range of papers [SWL+11; SLW13; LST+13a; LST+13b; SLW+14; SFL+14; STM+15]. Both, *SynOpSis* and its components underwent continuous enhancements. An early version of the optimization concept was presented as a poster [SLW13], which later became part of the full paper [SLW+14] discussed above. Hence, the author's contributions to the poster were already listed there. One exception is the parameter-optimized classifier used in the context of the poster, whereas the paper focused on the detector only.

After [SLW13] and [SLW+14] were published, the JGAP library used therein for evolutionary optimization was replaced with the Java-Based Evolutionary Computation Research System (ECJ) [Luk15]. Again, an interface by P. Libuschewski was used, which extends ECJ with functions for managing data dependencies [LMS+14]. This interface was incorporated into the final version of *SynOpSis* by the author, making it scriptable, adding *PAMONO*-specific evaluators and enabling different variants of sequential and global optimization.

Besides that, compared to the optimization approach used in [SLW+14], the version of *SynOpSis* presented in this thesis includes the following enhancements by the author: The set of optimized detector [LST+13a; LST+13b; SFL+14] parameters was reexamined. The classifier from [SWL+11] was included into the optimization and was enhanced itself as follows: New spatial and spatiotemporal features for classification were added to feature extraction in collaboration with P. Libuschewski. The Random Forest algorithm [Bre01] was added to the set of considered classifiers, and the possibility of balancing class prevalence [HBG+08] was included and evaluated. Furthermore, instead of using manually created ground truth for training as in the original paper [SWL+11], synthetic ground truth is used, which is available due to the signal model [SLW+14]. Given these enhancements of the classifier, all of which are detailed in Chapter 6, *SynOpSis* was furthermore enhanced by making the optimization multi-objective in terms of optimizing two objectives for the detector and two for the classifier. Hence automatic evaluators for the objectives were added. Besides that, methods for desirability-based [TW06] model selection and performance estimation [HTF09] were added after the optimization. Chapter 3 describes the *SynOpSis* approach including these enhancements. As parts of their merit, cross-experiment generalization performance as reported in [SWL+11] could be vastly improved, and data with smaller nano-objects, down to 100 nm in diameter, can now be analyzed reliably. Corresponding evaluations are provided in Chapter 7.

The evaluation capabilities of *SynOpSis* were furthermore used in [STM+15] to which the author contributed the analysis software created in collaboration with P. Libuschewski as described above. The paper applies *SynOpSis* for the detection of inactivated influenza A viruses and Virus-Like Particles (VLPs) of the Human Immunodeficiency Virus (HIV). VLPs are virus envelope proteins which are non-infectious because they are assembled without incorporating the genetic information necessary for reproduction [GA06]. Furthermore, *SynOpSis* served in proving the specificity of the individual virus-antibody binding events that the sensor allows to resolve. Automatic virus detection was applied, after its parameters had been optimized by *SynOpSis*, thus facilitating data analysis for the conducted experiments and making it repeatable. This was of particular importance in proving sensor specificity, as it involves comparison of results from experiments with different types of viruses.

Acronyms

k-NN k-Nearest Neighbors. 41, 66, 107–109, 113–117, 119, 125, 127, 138, 154, 157, 162–164, 171–174, 176, 246, 255, 257, 264

AABB Axis-Aligned Bounding Box. 146, 178
ABUS Automated Whole Breast Ultrasound. 30
ACO Ant Colony Optimization. 48
AdaBoost Adaptive Boosting. 30
ADASYN Adaptive Synthetic Sampling. 153–156, 175–177
AUC Area under the ROC Curve. 45, 253

BFGS Broyden-Fletcher-Goldfarb-Shanno. 24, 25, 47
BICO BIRCH Meets Coresets. 109, 113, 114, 119, 125, 246, 257
BIRCH Balanced Iterative Reducing and Clustering Using Hierarchies. 114

CADe Computer-Aided Detection. 30
CART Classification and Regression Tree. 165–167
CCD Charge-Coupled Device. 12, 15, 73, 75, 91, 98, 99, 185–187, 190
CMA-ES Covariance Matrix Adaptation Evolution Strategy. 24, 25
CNN Convolutional Neural Network. 246
CPU Central Processing Unit. 125, 222
CSV Comma-Separated Values. 170

D-Diff D-Rate Difference. 217–219
D-Rate Deviation-Rate. 192, 193, 204, 207, 209, 211–213, 215–219, 234–239
D-Summary D-Rate Summary. 208–211
DBN Deep Belief Network. 25
DF Desirability Function. 59–63, 169, 195–197, 202, 209, 245, 262
DI Desirability Index. 59, 61–63, 68, 161, 169, 196, 197, 199–202, 209, 212, 230–233, 245, 262
DOI Digital Object Identifier. 187
DWT Discrete Wavelet Transform. 109

EA Evolutionary Algorithm. 24, 26, 48
ECJ Java-Based Evolutionary Computation Research System. 258
EI Expected Improvement. 25
EM Electron Microscopy. 11
EMG Electromyogram. 150

FACT First G-APD Cherenkov Telescope. 249
FN False Negative. 36–38, 40, 43–45, 136, 137, 182, 192, 193, 203, 204, 207, 251, 252
FOCAS Faint Object Classification and Analysis System. 31
FP False Positive. 30, 36–40, 42–45, 75, 83, 84, 99, 136–138, 143, 169, 177, 192, 193, 196, 203, 204, 207, 215, 216, 219–221, 238, 251, 252
FPS Frames per Second. 186, 187, 221, 222

G-APD Geiger-Mode Avalanche Photodiode. 249
GA Genetic Algorithm. 47–52, 54, 68, 161, 169–171, 175, 178, 190, 193, 222, 237, 263, 264
GAT Generalized Anscombe Transform. 29, 91
GGA Gender-Based Genetic Algorithm. 26
GP Gaussian Process. 244
GPU Graphics Processing Unit. v, 25, 30, 50, 82, 96, 101, 104, 119, 125–127, 138, 144, 145, 152, 162, 167, 200, 222, 244, 247, 257

HIV Human Immunodeficiency Virus. 242, 243, 259
HOG Histogram of Oriented Gradients. 28

IQR Interquartile Range. 172, 173, 207, 209, 211
ISAS Leibniz-Institut für Analytische Wissenschaften. 257

JGAP Java Genetic Algorithms Package. 257, 258

KVM Kernel-Based Virtual Machine. 170

LDA Linear Discriminant Analysis. 30
LHD Latin Hypercube Design. 24, 25, 47
LTE Long-Term Evolution. 247

M-Rate Multi-Detection-Rate. 39, 40, 58, 135, 137, 191, 195, 196, 198, 200–203, 207, 209, 212, 215, 216, 218, 219, 228, 230–232, 234, 236–238
MA Memetic Algorithm. 48
MAE Mean Absolute Error. 216, 218, 219, 231–233
MAGIC Major Atmospheric Gamma Imaging Cherenkov Telescopes. 72, 248, 249
MAP Maximum a Posteriori. 167, 168
MC Monte Carlo. 72, 248, 249
MOEA Multi-Objective Evolutionary Algorithm. 26
MOGA Multi-Objective Genetic Algorithm. 47, 49–51, 54–56, 59, 61, 63, 68, 193, 194, 199
MPG Mixed-Poisson-Gaussian. 29, 31, 91
MSER Maximally Stable Extremal Regions. 28

NaCl Sodium Chloride. 186, 187
NPV Negative Predictive Value. 252
NSGA-II Non-Dominated Sorting Genetic Algorithm II. 47, 49, 50, 55–58, 193, 194
NSGA-III Non-Dominated Sorting Genetic Algorithm III. 194

OBB Oriented Bounding Box. 146, 178

PAES Pareto Archived Evolution Strategy. 49, 56
PAMONO Plasmon-Assisted Microscopy of Nano-Sized Objects. v, 2–13, 15–19, 21, 22, 24, 25, 27–30, 32–36, 38, 39, 42–44, 46–50, 59, 63, 67–69, 71–79, 82–87, 90–94, 96, 98–100, 104, 108–110, 115, 116, 119, 123, 125, 127–129, 133, 135, 138, 139, 141–143, 145, 147, 148, 150, 152–154, 157, 159, 161–163, 168–172, 176, 179, 181, 182, 184–191, 195, 200, 204–206, 209, 212, 217, 218, 220–223, 233, 237, 239, 241–249, 257, 258, 261–263
PBS Phosphate Buffered Saline. 186, 187
PCC Pearson Product-Moment Correlation Coefficient. 231, 232

POC Point of Care. 2
PPV Positive Predictive Value. 251
PSF Point Spread Function. 27, 29
PSO Particle Swarm Optimization. 48

RBF Radial Basis Function. 27, 28, 165, 171
RMSE Root Mean Squared Error. 231–233
ROC Receiver Operating Characteristic. 45, 253
ROI Region of Interest. 186, 187

SAT Propositional Satisfiability Problem. 26, 32
SMOTE Synthetic Minority Over-Sampling Technique. 153–156, 176
SMS-EMOA S-Metric Selection Evolutionary Multi-Objective Optimization Algorithm. 49, 56
SNR Signal-to-Noise Ratio. v, 4, 11, 13, 15, 27, 29–31, 86, 88–90, 96, 105, 106, 123, 124, 127, 131, 185, 187–190, 212, 214–216, 227, 235, 236, 238, 239, 241, 244
SPEA2 Strength Pareto Evolutionary Algorithm 2. 49, 56
SPOT Sequential Parameter Optimization Toolbox. 24, 25
SPR Surface Plasmon Resonance. v, 9–12, 73, 74, 105, 106, 123, 185, 241
STED Stimulated Emission Depletion. 10
STORM Stochastic Optical Reconstruction Microscopy. 10
SVM Support Vector Machine. 27, 28, 39, 41, 154, 157, 162–165, 167, 171–174, 182
SynOpSis Synthesis/Optimization/Analysis. 6–8, 18, 20–22, 25–35, 37, 41, 42, 44–48, 50, 56, 58–63, 65, 66, 68, 69, 71, 72, 75, 77–79, 82, 84, 127, 133, 135, 138, 139, 142, 144, 145, 154, 156, 158, 159, 161–163, 165, 167–177, 179, 180, 182, 184–186, 190, 193, 194, 196–200, 202–204, 206, 209, 211, 212, 215–217, 221–223, 230, 233, 237–239, 241–243, 245–249, 251, 257–259, 261

TI Translation-Invariant/Translation-Invariance. 107–111, 119, 123, 125–127, 255, 257, 258, 264
TN True Negative. 36, 37, 43–45, 137, 203, 204, 220, 251, 252
TP True Positive. 30, 36–40, 42–45, 84, 135–138, 143, 183, 191–193, 203, 204, 238, 251, 252

U-NSGA-III Unified Non-Dominated Sorting Genetic Algorithm III. 194

VLP Virus-Like Particle. 185, 243, 259
VM Virtual Machine. 170
VST Variance Stabilizing Transform. 29, 31, 91

Mathematical Symbols

General Rules

N Scalar integers representing cardinalities, e.g. the total number of elements in a set, a vector or a matrix dimension are indicated by upper case letters.

r Any other scalars, including integers, real numbers and class labels, are indicated by lower case letters.

$\mathbf{v}$ Vectors are indicated by bold type lower case letters.

v_i Vector components v_i are indicated by lower case letters equal to the letter of the full vector $\mathbf{v}$ they originate from. Indexing is done with subscripts.

$\mathbf{M}$ Matrices are indicated by bold type upper case letters.

$m_{i,j}$ Matrix entries $m_{i,j}$ are indicated by lower case letters equal to the letter of the matrix $\mathbf{M}$ they originate from. Indexing is done with double subscripts.

S Sets are indicated by upper case letters, exactly like scalar integers representing cardinalities (cf. first point). Confusion of the two should be precluded by the respective context of appearance.

s_i Set elements s_i are indicated by lower case letters equal to the letter of the set S they originate from. Indexing is done with subscripts. Confusion with vector components should be precluded by the respective context of appearance.

$\mathbf{x}^a$ Different instances of a symbol are distinguished by superscripts, e.g. $\mathbf{x}^a$ and $\mathbf{x}^b$ are two possibly different vectors. Subscripts and superscripts are combinable, e.g. x_i^a denotes the ith component of vector $\mathbf{x}^a$. Confusion with exponents should be precluded by the respective context of appearance.

α Functions are indicated by Greek upper or lower case letters. This includes those upper case Greek letters that exist as well in the Latin alphabet, i.e. the letters A, B, E, Z, H, I, K, M, N, O, P, T, Y, X.

In rare cases, exceptions to these rules are made. These serve to follow established conventions of mathematical nomenclature. Duplicate symbol names are avoided wherever possible. In case they could not be avoided without compromising the wish for intuitive names, all meanings are given in the following list, separated with (a) and (b). These cases are rare and it should always be clear from the context of symbol usage, which meaning is intended.

Parameters Being Optimized by SynOpSis

All parameters that are optimized by *SynOpSis* are summarized in respective sections in the text. For the parameters of the pattern detector, cf. Table 5.6 in Section 5.7. For the parameters of the classifier, cf. Table 6.2 in Section 6.8.

Functions (Greek Letters)

$A(x, y, t)$, $A\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ Artifacts signal component in *PAMONO* sensor data, evaluated at location x, y and time t. The properties of this component are similar to those of the target patterns signal $T(x, y, t)$, but $A(x, y, t)$ contains the non-target patterns.

$B(x,y)$, $B\colon \mathbb{N}_{>0}^2 \to \mathbb{R}$ Background signal component in *PAMONO* sensor data, evaluated at location x, y. This component is modeled as being constant over time. Any deviation from the constancy assumption is regarded as an artifact and hence belongs to the $A(x,y,t)$ signal.

$\Gamma(x,y,t)$, $\Gamma\colon \mathbb{N}_{>0}^3 \to \{0,1\}$ Binary class mask, ascribing each spatiotemporal coordinate x, y, t either to the positive class (up/down signals caused by nano-object adhesions) indicated by value one, or to the negative class (background, no nano-object adhesion) indicated by value zero.

$\gamma(x,y)$, $\gamma\colon \mathbb{N}_{>0}^2 \to \{0,1\}$ Abbreviation to denote one 2-D spatial image from the class mask Γ at a certain point in time t, i.e. $\gamma(\circ,\circ) = \Gamma(\circ,\circ,t)$ for some t.

$\Delta(\mathbf{y})$, $\Delta\colon \mathbb{R}^O \to]0,1]$ Desirability Index (DI) of objective values $\mathbf{y}$ from objective space.

$\delta(y_i)$, $\delta\colon \mathbb{R} \to]0,1]$ Desirability Function (DF) of objective value y_i.

$\zeta_{\text{isLarge}}^{l_1,l_2}(d)$, $\zeta_{\text{isLarge}}^{l_1,l_2}\colon \mathbb{R} \to [0,1]$ Membership of absolute intensity difference d in the fuzzy set of large values (used in fuzzy denoising).

$\zeta_{\text{isPos}}^{s_1,s_2}(m)$, $\zeta_{\text{isPos}}^{s_1,s_2}\colon [0,1] \to [0,1]$ Membership of matching score m in the fuzzy set of values possibly affected by a nano-object adhesion (used in fuzzy classification).

$\zeta_{\text{fringe}}^{s_1,s_2}(m)$, $\zeta_{\text{fringe}}^{s_1,s_2}\colon [0,1] \to [0,1]$ Membership of matching score m in the fuzzy set of values in the fringe of a nano-object adhesion (used in fuzzy classification).

$\zeta_{\text{time}}^{s_1,s_2}(m)$, $\zeta_{\text{time}}^{s_1,s_2}\colon [0,1] \to [0,1]$ Fuzzy rule realizing a temporal dilation of matching scores m (used in fuzzy classification).

$\zeta_{\text{up}\cup\text{down}}^{s_1,s_2}(m)$, $\zeta_{\text{up}\cup\text{down}}^{s_1,s_2}\colon [0,1] \to [0,1]$ Fuzzy rule for the class of up/down signals (used in fuzzy classification).

$\zeta_{\text{background}}^{s_1,s_2}(m)$, $\zeta_{\text{background}}^{s_1,s_2}\colon [0,1] \to [0,1]$ Fuzzy rule for the class of background signals (used in fuzzy classification).

$\mathbf{H}^\sigma(x,y)$, $\mathbf{H}^\sigma\colon \mathbb{N}_{>0}^2 \to \mathbb{R}^{2\times 2}$ Hessian matrix of $\eta(x,y)$ on scale σ, i.e. computed by convolving $\eta(x,y)$ with Gaussian derivative kernels of standard deviation σ.

$\eta(x,y)$, $\eta\colon \mathbb{N}_{>0}^2 \to \mathbb{R}$ Shorthand notation for a 2-D image $(T\cdot A)(x,y,t_c)$ at a certain temporal coordinate t_c.

$I(x,y,t)$, $I\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ Composite *PAMONO* sensor signal, evaluated at location x, y and time t. This is the signal that is measured by the sensor. It consists of several components, as described by Equation (4.1).

$I^{T=1}(x,y,t)$, $I^{T=1}\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ Composite *PAMONO* sensor signal during background measurement, evaluated at location x, y and time t. The target patterns component $T(x,y,t)$ is constantly one (neutral in the multiplicative signal model in Equation (4.1)) because during background measurements there are no target patterns in the flow cell of the sensor.

$\widetilde{I}(x,y,t)$, $\widetilde{I}\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ Composite synthetic *PAMONO* sensor signal, evaluated at location x, y and time t. Ground truth pattern locations and classification are known for this signal.

$\iota(x,y)$, $\iota\colon \mathbb{N}_{>0}^2 \to \mathbb{R}$ Single image $I(\circ,\circ,t_c)$ at a given time t_c.

$\kappa_{\text{avg}}(x,y)$, $\kappa_{\text{avg}}\colon \mathbb{N}_{>0}^2 \to \mathbb{R}$ 2-D averaging filter kernel.

$\kappa_{\text{Gauß}}(x,y \mid \sigma_{\text{Gauß}})$, $\kappa_{\text{Gauß}}\colon \mathbb{N}_{>0}^2 \to \mathbb{R}$ 2-D Gauß filter kernel with standard deviation $\sigma_{\text{Gauß}}$.

$\kappa_{\text{SE}}(x,y)$, $\kappa_{\text{SE}}\colon \mathbb{N}_{>0}^2 \to \{0,1\}$ Structuring element used in morphological operators.

$\Lambda(k,S)$ denotes the k-th largest element in the ordered multiset S.

$M(x,y,t)$, $M\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ Matching score in template matching as attained at spatiotemporal coordinate (x,y,t).

$\mu(\circ)$ Mean value of an arbitrary number of numerical input values (represented by the wildcard symbol "$\circ$"), cf. Equation (5.16).

$N(x,y,t),\ N\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ Noise component in *PAMONO* sensor data, at location x, y and time t.

$\xi(\mathbf{f}) = c_i,\ \xi\colon \mathbb{R}^F \to \{c_1,\ldots,c_C\}$ Classifying model, mapping a feature vector $\mathbf{f}$ from the feature space $\mathbb{R}^F$ to a class label c_i from the set of labels $\{c_1,\ldots,c_C\}$.

$P(\mathbf{f}),\ P\colon \mathbb{R}^F \to [0,1]$ Probability of observing feature vector $\mathbf{f}$.

$P(c_k),\ P\colon \{c_1,\ldots,c_C\} \to [0,1]$ Probability of observing class label c_k (prior).

$P(\mathbf{f} \mid c_k),\ P\colon \mathbb{R}^F \times \{c_1,\ldots,c_C\} \to [0,1]$ Conditional probability of observing feature vector $\mathbf{f}$, given that class label c_k has been observed (likelihood).

$P(c_k \mid \mathbf{f}),\ P\colon \{c_1,\ldots,c_C\} \times \mathbb{R}^F \to [0,1]$ Conditional probability of observing class label c_k, given that feature vector $\mathbf{f}$ has been observed (posterior probability).

$\rho(x,y,t_c),\ \rho\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ The past signal at time t_c, cf. Definition 5.1 for details.

$\sigma(\circ)$ Standard deviation of an arbitrary number of numerical input values (represented by the wildcard symbol "$\circ$"), cf. Equation (5.17).

$T(x,y,t),\ T\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ Target patterns signal component in *PAMONO* sensor data, evaluated at location x, y and time t. This is the desired signal of the measurement because it can serve as an indirect proof for nano-objects attaching to the sensor surface.

$\widetilde{T}(x,y,t),\ \widetilde{T}\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ Synthetic target patterns signal component in *PAMONO* sensor data, evaluated at location x, y and time t. Ground truth is known for this signal component.

$\phi(x,y,t_c),\ \phi\colon \mathbb{N}_{>0}^3 \to \mathbb{R}$ The present signal at time t_c, cf. Definition 5.1 for details.

$\varphi_i(\mathbf{x}) = y_i,\ \varphi_i\colon \mathbb{P}^P \to \mathbb{R}$ The ith objective function (synonymously: fitness function) assigning parameter space point $\mathbf{x} \in \mathbb{P}^P$ its value y_i in the ith objective, cf. also $\mathbf{y}$.

$\omega(x,y),\ \omega\colon \mathbb{N}_{>0}^2 \to \mathbb{R}$ Result of denoising $\iota(x,y)$.

Other Symbols

$\circ$ is the wildcard symbol for function arguments, i.e. a placeholder for the set of all possible values of a function argument. As an example, $I(\circ,\circ,t)$ denotes the 2-D projection of I, for which the third component has the value t.

$a \in \mathbb{R}$ The number of detector responses is allowed to be at most a times the number of ground truth target patterns in the early cancellation criterion for parameter sets in optimization, cf. Section 5.8.

$C \in \mathbb{N}_{>0}$ Number of classes.

c_i Name of ith class.

E (a) Elite size in a Genetic Algorithm (GA), then $E \in \mathbb{N}_{>0}$.
(b) Number of examples in a dataset, then $E \in \mathbb{N}_{\geq 0}$.

$\mathbf{E} \in \mathbb{N}_{\geq 0}^{C \times C}$ Confusion matrix.

$e_{i,j} \in \mathbb{N}_{\geq 0}$ Confusion matrix entry equaling the number of examples that were predicted to belong to class c_i, while ground truth assigns them to class c_j.

$\mathbf{F} \in \mathbb{R}^{E \times F}$ Feature matrix with E examples as row vectors in an F-dimensional feature space.

$f_{e,i} \in \mathbb{R}$ Entry of $\mathbf{F}$ corresponding to the value of feature i as observed in the eth example.

$\widehat{\mathbf{F}} \in \mathbb{R}^{E \times F}$ Normalization result of feature matrix $\mathbf{F}$.

$\widehat{f_{e,i}} \in \mathbb{R}$ Entry of $\widehat{\mathbf{F}}$ corresponding to the *normalized* value of feature i as observed in the eth example.

$F \in \mathbb{N}_{>0}$ Number of features and hence dimension of feature space $\mathbb{R}^F$.

$\mathbf{f} \in \mathbb{R}^F$ Feature vector from the feature space $\mathbb{R}^F$.

G (a) Number of generations in a genetic algorithm, then $G \in \mathbb{N}_{>0}$.
(b) Number of target patterns in the ground truth, then $G \in \mathbb{N}_{\geq 0}$.

K_m Number of cluster centers per class as computed by the k-Means algorithm during TI wavelet feature-based time series classification in Section 5.5.

K_n Number of neighbors regarded in k-NN during TI wavelet feature-based time series classification in Section 5.5.

$L^{X,Y,T} = \{m_1, \ldots, m_{XYT}\}$ Local neighborhood multiset of matching scores from M, occurring in a local cuboid with side-lengths X, Y, T in the horizontal, vertical and temporal directions. The cuboid is placed with its center at the spatiotemporal coordinate (x_c, y_c, t_c) for which $L^{X,Y,T}$ is computed. This coordinate is omitted from the symbol $L^{X,Y,T}$ to avoid notational clutter.

$m \in \mathbb{R}$ Shorthand notation for an individual matching score in template matching, i.e. $m = M(x_c, y_c, t_c)$ for a certain spatiotemporal coordinate (x_c, y_c, t_c).

$\mathbb{N}_{>0}$ Set of natural numbers excluding zero.

$\mathbb{N}_{\geq 0}$ Set of natural numbers including zero.

$O \in \mathbb{N}_{>0}$ Number of objectives, i.e. the dimensionality of objective space $\mathbb{R}^O$.

$o_i \in \mathbb{R}$ Offset for normalizing the ith feature.

$\mathbb{P}^P$ Parameter space with P dimensions. Note the mixed nature of $\mathbb{P}^P$: Each dimension may be from a different set, e.g. boolean, ordered/unordered discrete values or real values. The $\mathbb{P}$ is therefore a placeholder for other sets.

P Number of parameters, i.e. the dimensionality of parameter space.

$p \in \{c_1, \ldots, c_C\}$ Predicted class label.

$\mathbb{R}$ Set of real numbers.

$S \in \mathbb{N}_{>0}$ (a) Population size in a Genetic Algorithm (GA).
(b) Number of scales regarded in TI wavelet decomposition.

$s_i \in \mathbb{R}$ Scale factor for normalizing the ith feature.

$T \in \mathbb{N}_{>0}$ (a) Length of a time series that has been written as a vector, e.g. $\mathbf{v} \in \mathbb{R}^T$.
(b) Tournament size in a Genetic Algorithm (GA).

$\mathbf{t} \in \mathbb{R}^T$ Ideal template time series, written as a vector.

t (a) True class label, then $t \in \{c_1, \ldots, c_C\}$.
(b) Temporal coordinate, then $t \in \mathbb{N}_{>0}$.

$\mathbf{v} \in \mathbb{R}^T$ An observed time series, written as a vector.

$\widehat{\mathbf{W}} \in \mathbb{R}^{S \times T}$ TI wavelet coefficient table for a time series $\mathbf{v} \in \mathbb{R}^T$ over $S = \log(T)$ scales.

$\mathbf{W} \in \mathbb{R}^{S \times T}$ $\mathbf{W}$ is obtained from $\widehat{\mathbf{W}}$ by taking the absolute values of all coefficients in $\widehat{\mathbf{W}}$, followed by sorting each scale (row) separately by descending absolute coefficient value, making $\mathbf{W}$ invariant to circular shifts in the underlying time series $\mathbf{v}$.

$w_{s,t} \in \mathbb{R}_{\geq 0}$ Entry of TI wavelet coefficient table $\mathbf{W}$, yielding the tth-largest coefficient on scale s.

$\mathbf{x} \in \mathbb{P}^P$ Point in parameter space.

$x_i \in \mathbb{P}$ Value of the ith parameter. $\mathbb{P}$ is a placeholder for the set from which x_i may be selected.

$x \in \mathbb{N}_{>0}$ Horizontal spatial coordinate.

$\mathbf{y} \in \mathbb{R}^O$ Point in objective space, obtained as $(\varphi_1(\mathbf{x}) \ldots \varphi_O(\mathbf{x}))$.

$y_i \in \mathbb{R}$ Value of the ith objective, obtained as $\varphi_i(\mathbf{x})$.

$y \in \mathbb{N}_{>0}$ Vertical spatial coordinate.

Bibliography

[AB94] D. Altman and J. Bland. “Statistics Notes: Diagnostic Tests 2: Predictive Values”. In: *BMJ* 309.6947 (1994), p. 102 (Cited on pages 251 sq.).

[Abb73] E. Abbe. “Beiträge zur Theorie des Mikroskops und der mikroskopischen Wahrnehmung”. In: *Archiv für Mikroskopische Anatomie* 9.1 (1873), pp. 413–418 (Cited on pages 9, 27, 29).

[AH01] S. Aksoy and R. M. Haralick. “Feature Normalization and Likelihood-Based Similarity Measures for Image Retrieval”. In: *Pattern Recognition Letters* 22.5 (2001), pp. 563–582 (Cited on pages 157 sq.).

[ALN+12] C. Arteta, V. Lempitsk, J. A. Noble, and A. Zisserman. “Learning to Detect Cells Using Non-Overlapping Extremal Regions”. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. 2012, pp. 348–356 (Cited on pages 2 sq., 27 sqq., 39, 248).

[Alp97] E. Alpaydin. “Voting Over Multiple Condensed Nearest Neighbors”. In: *Artificial Intelligence Review* 11.1–5 (1997), pp. 115–132 (Cited on page 109).

[Ang05] F. Angiulli. “Fast Condensed Nearest Neighbor Rule”. In: *Proceedings of the 22nd International Conference on Machine Learning (ICML)*. 2005, pp. 25–32 (Cited on page 109).

[ARR+07] N. Agrawal, G. P. Rangaiah, A. K. Ray, and S. K. Gupta. “Design Stage Optimization of an Industrial Low-Density Polyethylene Tubular Reactor for Multiple Objectives Using NSGA-II and Its Jumping Gene Adaptations”. In: *Chemical Engineering Science* 62.9 (2007), pp. 2346–2365 (Cited on pages 49, 56).

[AST09] C. Ansótegui, M. Sellmann, and K. Tierney. “A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms”. In: *Principles and Practice of Constraint Programming*. Springer, 2009, pp. 142–157 (Cited on pages 23, 26, 48 sq.).

[AV07] D. Arthur and S. Vassilvitskii. “`k-Means++`: The Advantages of Careful Seeding”. In: *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2007, pp. 1027–1035 (Cited on page 114).

[BB00] Y. M. Blanter and M. Buttiker. “Shot Noise in Mesoscopic Conductors”. In: *Physics Reports* 336.1 (2000), pp. 1–166 (Cited on pages 15, 29, 75).

[BB12] J. Bergstra and Y. Bengio. “Random Search for Hyper-Parameter Optimization”. In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 281–305 (Cited on pages 47, 49 sq., 244).

[BBB+11] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2011, pp. 2546–2554 (Cited on pages 3, 23, 25, 46, 48, 63, 229, 244 sq., 247).

[BFS+84] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. CRC Press, 1984 (Cited on pages 165 sq.).

[BKG11] M. Buhrmester, T. Kwang, and S. D. Gosling. "Amazon's Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data?" In: *Perspectives on Psychological Science* 6.1 (2011), pp. 3–5 (Cited on pages 127, 246).

[BKL+13] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. "Mixed-Integer Nonlinear Optimization". In: *Acta Numerica* 22 (2013), pp. 1–131 (Cited on page 49).

[BL03] D. C. Banks and S. Linton. "Counting Cases in Marching Cubes: Toward a Generic Algorithm for Producing Substitopes". In: *Proceedings of the 14th IEEE Visualization Conference (VIS 03)*. 2003, pp. 51–58 (Cited on pages 128–131).

[BLP10] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. "The Sequential Parameter Optimization Toolbox". In: *Experimental Methods for the Analysis of Optimization Algorithms*. 2010, pp. 337–362 (Cited on pages 23 sq., 48, 229, 245, 247).

[BM91] B. Bourguignon and D. L. Massart. "Simultaneous Optimization of Several Chromatographic Performance Goals Using Derringer's Desirability Function". In: *Journal of Chromatography A* 586.1 (1991), pp. 11–20 (Cited on page 59).

[BMT+12] B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs. "Resampling Methods for Meta-Model Validation with Recommendations for Evolutionary Computation". In: *Evolutionary Computation* 20.2 (2012), pp. 249–275 (Cited on pages 23, 25, 48, 229, 245, 247).

[BN07] E. G. Bekele and J. W. Nicklow. "Multi-Objective Automatic Calibration of SWAT Using NSGA-II". In: *Journal of Hydrology* 341.3 (2007), pp. 165–176 (Cited on pages 49, 56).

[BNE07] N. Beume, B. Naujoks, and M. Emmerich. "SMS-EMOA: Multiobjective Selection Based on Dominated Hypervolume". In: *European Journal of Operational Research* 181.3 (2007), pp. 1653–1669 (Cited on pages 49, 56).

[Bre01] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32 (Cited on pages 25, 41, 142, 157, 165 sqq., 172, 228 sq., 244, 258).

[Bre96] L. Breiman. "Bagging Predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140 (Cited on page 165).

[Bru07] T. Bruckhaus. "The Business Impact of Predictive Analytics". In: *Knowledge Discovery and Data Mining. Challenges and Realities*. Information Science Reference, 2007, pp. 114–138 (Cited on page 126).

[BSB+04] P. M. Boltovets, B. A. Snopok, V. R. Boyko, T. P. Shevchenko, N. S. Dyachenko, and Y. M. Shirshov. "Detection of Plant Viruses Using a Surface Plasmon Resonance via Complexing with Specific Antibodies". In: *Journal of Virological Methods* 121.1 (2004), pp. 101–106 (Cited on page 10).

[BSP+02] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. "A Racing Algorithm for Configuring Metaheuristics". In: *Genetic and Evolutionary Computation Conference (GECCO)*. Vol. 2. 2002, pp. 11–18 (Cited on pages 23, 48).

[Bur98] C. J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". In: *Data Mining and Knowledge Discovery* 2.2 (1998), pp. 121–167 (Cited on pages 163, 165, 171).

[BYB+10] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. "F-Race and Iterated F-Race: An Overview". In: *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, 2010, pp. 311–336 (Cited on pages 23 sq., 48).

[CBH+02] N. V. Chawla, K. W. Bowyer, L. Hall, and W. P. Kegelmeyer. "SMOTE: Synthetic Minority Over-Sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357 (Cited on pages 153 sqq.).

[CD95] R. R. Coifman and D. L. Donoho. *Translation-Invariant De-Noising*. Springer, 1995 (Cited on pages 107, 109 sq., 255).

[CGK+92] J.-N. Capdevielle, P. Grieder, J. Knapp, P. Gabriel, H. J. Gils, D. Heck, H. J. Mayer, J. Oehlschläger, H. Rebel, G. Schatz, et al. *The Karlsruhe Extensive Air Shower Simulation Code CORSIKA*. Tech. rep. Kernforschungszentrum Karlsruhe GmbH (Germany). Institut für Kernphysik, 1992 (Cited on page 249).

[Chi92] N. Chinchor. "MUC-4 Evaluation Metrics". In: *Proceedings of the Fourth Message Understanding Conference*. 1992, pp. 22–29 (Cited on pages 45, 252).

[CKB+05] J. W. Chung, S. D. Kim, R. Bernhardt, and J. C. Pyun. "Application of SPR Biosensor for Medical Diagnostics of Human Hepatitis B Virus (hHBV)". In: *Sensors and Actuators B: Chemical* 111 (2005), pp. 416–422 (Cited on page 10).

[CM02] D. Comaniciu and P. Meer. "Mean Shift: A Robust Approach Toward Feature Space Analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 24.5 (2002), pp. 603–619 (Cited on page 28).

[CMO97] R. E. Caflisch, W. J. Morokoff, and A. B. Owen. *Valuation of Mortgage Backed Securities Using Brownian Bridges to Reduce Effective Dimension*. Department of Mathematics, University of California, Los Angeles (UCLA), 1997 (Cited on page 50).

[COM15] COMSOL. *COMSOL Multiphysics*. Commercial Software. 2015. URL: `https://www.comsol.com/` (Cited on page 72).

[Con99] W. J. Conover. *Practical Nonparametric Statistics*. Wiley, 1999 (Cited on page 23).

[CS00] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000 (Cited on page 171).

[CWG01] M. K. Cheezum, W. F. Walker, and W. H. Guilford. "Quantitative Comparison of Algorithms for Tracking Single Fluorescent Particles". In: *Biophysical Journal* 81.4 (2001), pp. 2378–2388 (Cited on pages 4, 187, 190, 215).

[Dan98] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1998 (Cited on page 48).

[DB05] M. Dorigo and C. Blum. "Ant Colony Optimization Theory: A Survey". In: *Theoretical Computer Science* 344.2 (2005), pp. 243–278 (Cited on page 48).

[DE13] M. Doert and M. Errando. "High Confidence AGN Candidates Among Unidentified Fermi-Lat Sources via Statistical Classification". In: *Proceedings of the International Cosmic Ray Conference (ICRC)*. 2013 (Cited on pages 2, 248).

[Deb01] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001 (Cited on pages 21, 26, 48, 54 sq.).

[DK07] K. Deb and S. Karthik. "Dynamic Multi-Objective Optimization and Decision-Making Using Modified NSGA-II: A Case Study on Hydro-Thermal Power Scheduling". In: *Evolutionary Multi-Criterion Optimization*. 2007, pp. 803–817 (Cited on pages 49, 56).

[DK13] A. Donath and D. Kondermann. "Is Crowdsourcing for Optical Flow Ground Truth Generation Feasible?" In: *Computer Vision Systems*. Springer, 2013, pp. 193–202 (Cited on pages 127, 246).

[DK77] K. Dines and A. C. Kak. "Constrained Least Squares Filtering". In: *IEEE Transactions on Acoustics, Speech and Signal Processing* 25.4 (1977), pp. 346–350 (Cited on page 34).

[DMS06] D. Donoho, A. Maleki, and M. Shahram. *Wavelab 850. Software Toolkit for Time-Frequency Analysis*. Stanford University, Department of Statistics. 2006. URL: `http://statweb.stanford.edu/~wavelab/` (Cited on page 255).

[Dom99] P. Domingos. "Metacost: A General Method for Making Classifiers Cost-Sensitive". In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1999, pp. 155–164 (Cited on page 153).

[DPA+02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197 (Cited on pages 49 sq., 55 sq., 58, 193 sq.).

[DPM00] K. Deb, A. Pratap, and S. Moitra. "Mechanical Component Design for Multiple Objectives Using Elitist Non-Dominated Sorting GA". In: *Parallel Problem Solving from Nature (PPSN VI)*. 2000, pp. 859–868 (Cited on pages 49, 56).

[DS80] G. C. Derringer and D. Suich. "Simultaneous Optimization of Several Response Variables". In: *Journal of Quality Technology* 12.4 (1980), pp. 214–219 (Cited on page 61).

[DT05] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. 2005, pp. 886–893 (Cited on page 28).

[DT14] W. Duivesteijn and J. Thaele. *Understanding Where Your Classifier Does (Not) Work – The SCaPE Model Class for Exceptional Model Mining*. Tech. rep. TU Dortmund University, 2014 (Cited on pages 2, 249).

[EMY+08] D. Erickson, S. Mandal, A. H. J. Yang, and B. Cordovez. "Nanobiosensors: Optofluidic, Electrical and Mechanical Approaches to Biomolecular Detection at the Nanoscale". In: *Microfluidics and Nanofluidics* 4.1–2 (2008), pp. 33–52 (Cited on page 2).

[FA91] W. T. Freeman and E. H. Adelson. "The Design and Use of Steerable Filters". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 13.9 (1991), pp. 891–906 (Cited on page 148).

[Faw06] T. Fawcett. "An Introduction to ROC Analysis". In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874 (Cited on pages 45, 253).

[FCN+13] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. "Learning Hierarchical Features for Scene Labeling". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35.8 (2013), pp. 1915–1929 (Cited on page 246).

[FGS+13] H. Fichtenberger, M. Gillé, M. Schmidt, C. Schwiegelshohn, and C. Sohler. "BICO: BIRCH Meets Coresets for k-means Clustering". In: *Algorithms – ESA 2013*. Springer, 2013, pp. 481–492 (Cited on pages 107, 109, 113 sq., 246, 257).

[Fle08] T. Fletcher. *Support Vector Machines Explained*. University College London (UCL). 2008. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.164.3457` (Cited on page 165).

[FM06] H. Faraji and W. J. MacLean. "CCD Noise Removal in Digital Images". In: *IEEE Transactions on Image Processing* 15.9 (2006), pp. 2676–2685 (Cited on pages 15, 29, 75, 187).

[FS97] Y. Freund and R. E. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139 (Cited on pages 28, 30, 167).

[GA06] E. V. L. Grgacic and D. A. Anderson. "Virus-Like Particles: Passport to Immune Recognition". In: *Methods* 40.1 (2006), pp. 60–65 (Cited on pages 185, 243, 259).

[GBG05] C. Guria, P. K. Bhattacharya, and S. K. Gupta. "Multi-Objective Optimization of Reverse Osmosis Desalination Units Using Different Adaptations of the Non-Dominated Sorting Genetic Algorithm (NSGA)". In: *Computers & Chemical Engineering* 29.9 (2005), pp. 1977–1995 (Cited on pages 49, 56).

[GBP+06] J. L. Greenwald, G. R. Burstein, J. Pincus, and B. Branson. "A Rapid Review of Rapid HIV Antibody Tests". In: *Current Infectious Disease Reports* 8.2 (2006), pp. 125–131 (Cited on page 243).

[GE03] I. Guyon and A. Elisseeff. "An Introduction to Variable and Feature Selection". In: *The Journal of Machine Learning Research* 3 (2003), pp. 1157–1182 (Cited on pages 159 sq.).

[GK79] K. C. Gowda and G. Krishna. "The Condensed Nearest Neighbor Rule Using the Concept of Mutual Nearest Neighborhood". In: *IEEE Transactions on Information Theory* 25.4 (1979), pp. 488–490 (Cited on page 109).

[GTÜ+11] E. L. Gurevich, V. Temchura, K. Überla, and A. Zybin. "Analytical Features of Particle Counting Sensor Based on Plasmon Assisted Microscopy of Nano Objects". In: *Sensors and Actuators B: Chemical* 160.1 (2011), pp. 1210–1215 (Cited on pages 152, 241).

[GW07] R. C. Gonzalez and R. E. Woods. *Digital Image Processing 3rd Edition*. Prentice Hall, 2007 (Cited on pages 31, 91–94, 115, 128, 148).

[Han06] N. Hansen. "The CMA Evolution Strategy: A Comparing Review". In: *Towards a New Evolutionary Computation*. Springer, 2006, pp. 75–102 (Cited on page 24).

[Har65] J. Harrington. "The Desirability Function". In: *Industrial Quality Control* 21.10 (1965), pp. 494–498 (Cited on pages 59 sq., 195).

[HBG+08] H. He, Y. Bai, E. A. Garcia, and S. Li. "ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning". In: *IEEE International Joint Conference on Neural Networks (IJCNN)*. 2008, pp. 1322–1328 (Cited on pages 153, 155, 258).

[HBK10] B. Huang, B. Buckley, and T.-M. Kechadi. "Multi-Objective Feature Selection by Using NSGA-II for Customer Churn Prediction in Telecommunications". In: *Expert Systems with Applications* 37.5 (2010), pp. 3638–3646 (Cited on pages 49, 56).

[HBR+08] J. W. Han, T. Breckon, D. Randell, and G. Landini. "Radicular Cysts and Odontogenic Keratocysts Epithelia Classification Using Cascaded Haar Classifiers". In: *Proceedings of the Twelfth Annual Conference of Medical Image Understanding and Analysis*. 2008, pp. 54–58 (Cited on pages 2, 27 sq., 30, 72, 167, 248).

[HBR+12] J. W. Han, T. Breckon, D. Randell, and G. Landini. "The Application of Support Vector Machine Classification to Detect Cell Nuclei for Automated Microscopy". In: *Machine Vision and Applications* 23.1 (2012), pp. 15–24 (Cited on pages 2, 15, 19, 27, 32, 39, 72, 248).

[HCL03] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. *A Practical Guide to Support Vector Classification*. Department of Computer Science, National Taiwan University. (Last updated 2010). 2003 (Cited on pages 157 sq., 163, 165, 171).

[HG09] H. He and E. A. Garcia. "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (2009), pp. 1263–1284 (Cited on pages 26, 38, 44, 144, 152–155, 176, 251).

[HHL+09] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. "ParamILS: An Automatic Algorithm Configuration Framework". In: *Journal of Artificial Intelligence Research* 36.1 (2009), pp. 267–306 (Cited on pages 23, 48).

[HHL11] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523 (Cited on pages 23, 25, 48, 229, 245, 247).

[Hil85] A. M. Hillas. "Cerenkov Light Images of EAS Produced by Primary Gamma". In: *Proceedings of the International Cosmic Ray Conference (ICRC)*. 1985, pp. 445–448 (Cited on page 248).

[HK73] J. E. Hopcroft and R. M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231 (Cited on page 137).

[HM04] S. Har-Peled and S. Mazumdar. "On Coresets for k-means and k-median Clustering". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*. 2004, pp. 291–300 (Cited on pages 107, 114).

[HOT06] G. E. Hinton, S. Osindero, and Y.-W. Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (2006), pp. 1527–1554 (Cited on page 25).

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009 (Cited on pages 24, 27, 30, 41, 64–67, 113, 153, 157, 162 sq., 165 sqq., 171 sq., 228, 258).

[Hu62] M.-K. Hu. "Visual Pattern Recognition by Moment Invariants". In: *IRE Transactions on Information Theory* 8.2 (1962), pp. 179–187 (Cited on pages 31, 145, 147).

[HW94] S. W. Hell and J. Wichmann. "Breaking the Diffraction Resolution Limit by Stimulated Emission: Stimulated-Emission-Depletion Fluorescence Microscopy". In: *Optics Letters* 19.11 (1994), pp. 780–782 (Cited on page 10).

[HWM05] H. Han, W.-Y. Wang, and B.-H. Mao. "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning". In: *Advances in Intelligent Computing*. Springer, 2005, pp. 878–887 (Cited on page 153).

[JB99] M. Jansen and A. Bultheel. "Multiple Wavelet Threshold Estimation by Generalized Cross Validation for Images with Correlated Noise". In: *IEEE Transactions on Image Processing* 8.7 (1999), pp. 947–953 (Cited on page 29).

[JD13] H. Jain and K. Deb. "An Improved Adaptive Approach for Elitist Nondominated Sorting Genetic Algorithm for Many-Objective Optimization". In: *Evolutionary Multi-Criterion Optimization*. 2013, pp. 307–321 (Cited on page 194).

[JJN+91] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3.1 (1991), pp. 79–87 (Cited on pages 28, 182).

[JK09] I.-J. Jeong and K.-J. Kim. "An Interactive Desirability Function Method to Multiresponse Optimization". In: *European Journal of Operational Research* 195.2 (2009), pp. 412–426 (Cited on page 59).

[JRZ14] M. Janidarmian, K. Radecka, and Z. Zilic. "Automated Diagnosis of Knee Pathology Using Sensory Data". In: *2014 EAI Fourth International Conference on Wireless Mobile Communication and Healthcare (MobiHealth)*. 2014, pp. 95–98 (Cited on page 150).

[JS97] I. M. Johnstone and B. W. Silverman. "Wavelet Threshold Estimators for Data with Correlated Noise". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 59.2 (1997), pp. 319–351 (Cited on page 29).

[JSW98] D. R. Jones, M. Schonlau, and W. J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13.4 (1998), pp. 455–492 (Cited on pages 24 sq.).

[JT81] J. Jarvis and J. Tyson. "FOCAS: Faint Object Classification and Analysis System". In: *The Astronomical Journal* 86 (1981), pp. 476–495 (Cited on pages 27, 31, 145, 147).

[JZK+07] S. Jiang, X. Zhou, T. Kirchhausen, and S. T. Wong. "Detection of Molecular Particles in Live Cells via Machine Learning". In: *Cytometry Part A* 71.8 (2007), pp. 563–575 (Cited on pages 2, 27–30, 167, 187, 248).

[KBM+09] S. Kannan, S. Baskar, J. D. McCalley, and P. Murugan. "Application of NSGA-II Algorithm to Generation Expansion Planning". In: *IEEE Transactions on Power Systems* 24.1 (2009), pp. 454–461 (Cited on pages 49, 56).

[KC99] J. Knowles and D. Corne. "The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimisation". In: *Evolutionary Computation*. Vol. 1. 1999 (Cited on pages 49, 56).

[Kel99] C. T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics (SIAM), 1999 (Cited on pages 24, 47).

[Ken10] J. Kennedy. "Particle Swarm Optimization". In: *Encyclopedia of Machine Learning*. Springer, 2010, pp. 760–766 (Cited on page 48).

[KKF+11] W. Konen, P. Koch, O. Flasch, T. Bartz-Beielstein, M. Friese, and B. Naujoks. "Tuned Data Mining: A Benchmark Study on Different Tuners". In: *Genetic and Evolutionary Computation Conference (GECCO)*. Vol. 11. 2011, pp. 1995–2002 (Cited on pages 23 sqq., 48, 229, 245, 247).

[KL03] S. S. Keerthi and C.-J. Lin. "Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel". In: *Neural Computation* 15.7 (2003), pp. 1667–1689 (Cited on page 171).

[KL10] K. Kumar and Y.-H. Lu. "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" In: *Computer* 43.4 (2010), pp. 51–56 (Cited on page 247).

[KN08] J. Knowles and H. Nakayama. "Meta-Modeling in Multiobjective Optimization". In: *Multiobjective Optimization*. Springer, 2008, pp. 245–284 (Cited on page 245).

[Koh95] R. Kohavi. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 14. 2. 1995, pp. 1137–1145 (Cited on pages 46, 64, 113, 116, 144, 165, 197 sq., 223, 229).

[KP03] H. L. Kundel and M. Polansky. "Measurement of Observer Agreement". In: *Radiology* 228.2 (2003), pp. 303–308 (Cited on page 252).

[Kre71] E. Kretschmann. "Determination of Optical Constants of Metal by Excitation of Surface Plasmons". In: *Zeitschrift für Physik* 241.4 (1971), pp. 313–324 (Cited on page 12).

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2012, pp. 1097–1105 (Cited on page 246).

[Kun07] L. I. Kuncheva. "A Stability Index for Feature Selection". In: *Artificial Intelligence and Applications*. 2007, pp. 421–427 (Cited on pages 113, 116, 119 sqq.).

[Lan06] G. Landini. "Quantitative Analysis of the Epithelial Lining Architecture in Radicular Cysts and Odontogenic Keratocysts". In: *Head & Face Medicine* 2.4 (2006), pp. 1–9 (Cited on pages 145 sq.).

[Lea06] E. G. Learned-Miller. "Data Driven Image Models Through Continuous Joint Alignment". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 28.2 (2006), pp. 236–250 (Cited on pages 33, 72).

[LEC+07] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. "An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation". In: *Proceedings of the 24th International Conference on Machine Learning (ICML)*. 2007, pp. 473–480 (Cited on page 25).

[Lib15a] P. Libuschewski. Feature Extraction on GPUs. Unpublished Program Code. 2015 (Cited on pages 144 sq.).

[Lib15b] P. Libuschewski. Random Forest Evaluator for GPUs. Unpublished Program Code. 2015 (Cited on pages 144, 162, 167, 222, 244).

[Lin94] T. Lindeberg. "Scale-Space Theory: A Basic Tool for Analyzing Structures at Different Scales". In: *Journal of Applied Statistics* 21.1–2 (1994), pp. 225–270 (Cited on page 148).

[LKD+14] P. Libuschewski, D. Kaulbars, B. Dusza, D. Siedhoff, F. Weichert, H. Müller, C. Wietfeld, and P. Marwedel. "Multi-Objective Computation Offloading for Mobile Biosensors via LTE". In: *2014 EAI Fourth International Conference on Wireless Mobile Communication and Healthcare (MobiHealth)*. 2014, pp. 226–229 (Cited on pages 2, 49, 233, 245 sq.).

[LL03] H.-T. Lin and C.-J. Lin. *A Study on Sigmoid Kernels for SVM and the Training of Non-PSD Kernels by SMO-Type Methods*. Tech. rep. National Taiwan University, 2003 (Cited on page 171).

[Llo82] S. P. Lloyd. "Least Squares Quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137 (Cited on page 114).

[LLS08] D. Li, H. Luo, and Z. Shi. "Redundant DWT Based Translation Invariant Wavelet Feature Extraction for Face Recognition". In: *19th International Conference on Pattern Recognition (ICPR)*. 2008, pp. 1–4 (Cited on page 109).

[LMS+14] P. Libuschewski, P. Marwedel, D. Siedhoff, and H. Müller. "Multi-Objective Energy-Aware GPGPU Design Space Exploration for Medical or Industrial Applications". In: *2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*. 2014, pp. 637–644 (Cited on pages 49 sq., 241, 258).

[Low04] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision (IJCV)* 60.2 (2004), pp. 91–110 (Cited on page 93).

[LST+13a] P. Libuschewski, D. Siedhoff, C. Timm, A. Gelenberg, and F. Weichert. "Fuzzy-Enhanced, Real-Time Capable Detection of Biological Viruses Using a Portable Biosensor". In: *Proceedings of the International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSIGNALS)*. 2013, pp. 169–174 (Cited on pages 5, 10 sq., 22, 25, 68, 82, 96, 101, 104, 106, 125, 241, 257 sq.).

[LST+13b] P. Libuschewski, D. Siedhoff, C. Timm, and F. Weichert. “Mobile Detektion Viraler Pathogene Durch Echtzeitfähige GPGPU-Fuzzy-Segmentierung”. In: *Bildverarbeitung für die Medizin 2013*. Springer, 2013, pp. 326–331 (Cited on pages 5, 10 sq., 22, 68, 101, 104, 125, 257 sq.).

[LSW13] P. Libuschewski, D. Siedhoff, and F. Weichert. “Energy-Aware Design Space Exploration for GPGPUs”. In: *Computer Science – Research and Development* 29.3–4 (2013), pp. 171–176 (Cited on pages 49, 241).

[Luk13] S. Luke. *Essentials of Metaheuristics (Second Edition)*. George Mason University, 2013. URL: `http://cs.gmu.edu/~sean/book/metaheuristics/` (Cited on pages 26, 48–53, 55 sq., 178, 193).

[Luk15] S. Luke. *The ECJ Owner's Manual*. George Mason University, 2015. URL: `https://cs.gmu.edu/~eclab/projects/ecj/` (Cited on page 258).

[LW02] A. Liaw and M. Wiener. “Classification and Regression by RandomForest”. In: *R News* 2.3 (2002), pp. 18–22 (Cited on pages 165, 167, 172).

[LWT12] P. Libuschewski, F. Weichert, and C. Timm. “Parameteroptimierte und GPGPU-basierte Detektion viraler Strukturen innerhalb Plasmonen-unterstützter Mikroskopiedaten”. In: *Bildverarbeitung für die Medizin 2012*. Springer, 2012, pp. 237–242 (Cited on page 96).

[LWZ09] X.-Y. Liu, J. Wu, and Z.-H. Zhou. “Exploratory Undersampling for Class-Imbalance Learning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 39.2 (2009), pp. 539–550 (Cited on page 153).

[Mal99] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1999 (Cited on pages 29, 31).

[Mar63] D. W. Marquardt. “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. In: *Journal of the Society for Industrial & Applied Mathematics* 11.2 (1963), pp. 431–441 (Cited on page 47).

[MBC79] M. D. McKay, R. J. Beckman, and W. J. Conover. “Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code”. In: *Technometrics* 21.2 (1979), pp. 239–245 (Cited on pages 24, 47).

[MBW+12] M. A. Mayer, A. Borsdorf, M. Wagner, J. Hornegger, C. Y. Mardin, and R. P. Tornow. “Wavelet Denoising of Multiframe Optical Coherence Tomography Data”. In: *Biomedical Optics Express* 3.3 (2012), pp. 572–589 (Cited on page 187).

[MCU+04] J. Matas, O. Chum, M. Urban, and T. Pajdla. “Robust Wide-Baseline Stereo from Maximally Stable Extremal Regions”. In: *Image and Vision Computing* 22.10 (2004), pp. 761–767 (Cited on page 28).

[Mie08] G. Mie. “Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen”. In: *Annalen der Physik* 330.3 (1908), pp. 377–445 (Cited on pages 9, 13).

[MKB09] P. Murugan, S. Kannan, and S. Baskar. “NSGA-II Algorithm for Multi-Objective Generation Expansion Planning Problem”. In: *Electric Power Systems Research* 79.4 (2009), pp. 622–628 (Cited on pages 49, 56).

[MM97] O. Maron and A. W. Moore. “The Racing Algorithm: Model Selection for Lazy Learners”. In: *Lazy Learning*. Springer, 1997, pp. 193–225 (Cited on page 23).

[MMB+05] P. Majumdar, A. Moralejo, C. Bigongiari, O. Blanch, and D. Sobczynska. “Monte Carlo Simulation for the MAGIC Telescope”. In: *Proceedings of the International Cosmic Ray Conference (ICRC)*. 2005, pp. 41–44 (Cited on pages 33, 72, 248).

[MMB+14a] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coello. “A Survey of Multiobjective Evolutionary Algorithms for Data Mining: Part I”. In: *IEEE Transactions on Evolutionary Computation* 18.1 (2014), pp. 4–19 (Cited on pages 23, 26, 48 sq.).

[MMB+14b] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coello. “A Survey of Multiobjective Evolutionary Algorithms for Data Mining: Part II”. In: *IEEE Transactions on Evolutionary Computation* 18.1 (2014), pp. 20–35 (Cited on pages 23, 26, 48 sq.).

[MMR+01] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. “An Introduction to Kernel-Based Learning Algorithms”. In: *IEEE Transactions on Neural Networks* 12.2 (2001), pp. 181–201 (Cited on pages 27 sq., 41, 157, 163 sq.).

[MNK11] T. Melange, M. Nachtegael, and E. E. Kerre. “Fuzzy Random Impulse Noise Removal from Color Image Sequences”. In: *IEEE Transactions on Image Processing* 20.4 (2011), pp. 959–970 (Cited on pages 95 sqq.).

[Mos89] P. Moscato. “On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms”. In: *Caltech Concurrent Computation Program, C3P Report* 826 (1989), pp. 1–67 (Cited on page 48).

[MRE09] J. Mairhofer, K. Roppert, and P. Ertl. “Microfluidic Systems for Pathogen Sensing: A Review”. In: *Sensors* 9.6 (2009), pp. 4804–4823 (Cited on page 2).

[MRK+11] K. Meffert, N. Rotstan, C. Knowles, and U. Sangiorgi. *JGAP – Java Genetic Algorithms and Genetic Programming Package*. Open-Source Software. 2011. URL: `http://jgap.sourceforge.net/` (Cited on page 257).

[MSB+13] W. K. Moon, Y.-W. Shen, M. S. Bae, C.-S. Huang, J.-H. Chen, and R.-F. Chang. “Computer-Aided Tumor Detection Based on Multi-Scale Blob Detection Algorithm in Automated Breast Ultrasound Images”. In: *IEEE Transactions on Medical Imaging* 32.7 (2013), pp. 1191–1200 (Cited on pages 2, 27, 30, 148 sqq.).

[MSB95] F. Murtagh, J.-L. Starck, and A. Bijaoui. “Multiresolution in Astronomical Image Processing: A General Framework”. In: *International Journal of Imaging Systems and Technology* 6.4 (1995), pp. 332–338 (Cited on pages 27, 31 sq., 91).

[MT01] K. Ma and X. Tang. “Translation-Invariant Face Feature Estimation Using Discrete Wavelet Transform”. In: *Wavelet Analysis and Its Applications*. Springer, 2001, pp. 200–210 (Cited on page 109).

[MT06] J. Mehnen and H. Trautmann. "Integration of Expert's Preferences in Pareto Optimization by Desirability Function Techniques". In: *Proceedings of the Fifth CIRP International Seminar on Intelligent Computation in Manufacturing Engineering (CIRP ICME '06)*. 2006, pp. 293–298 (Cited on page 59).

[MTL78] R. McGill, J. W. Tukey, and W. A. Larsen. "Variations of Box Plots". In: *The American Statistician* 32.1 (1978), pp. 12–16 (Cited on pages 170, 206).

[MTT07] J. Mehnen, H. Trautmann, and A. Tiwari. "Introducing User Preference Using Desirability Functions in Multi-Objective Evolutionary Optimisation of Noisy Processes". In: *IEEE Congress on Evolutionary Computation (CEC)*. 2007, pp. 2687–2694 (Cited on page 59).

[MZ92] S. Mallat and S. Zhong. "Characterization of Signals from Multiscale Edges". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 14.7 (1992), pp. 710–732 (Cited on page 29).

[NLE+15] O. Neugebauer, P. Libuschewski, M. Engel, H. Müller, and P. Marwedel. "Plasmon-Based Virus Detection on Heterogeneous Embedded Systems". In: *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*. 2015, pp. 48–57 (Cited on pages 16, 49, 241).

[NM65] J. A. Nelder and R. Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (1965), pp. 308–313 (Cited on page 48).

[NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science & Business Media, 2006 (Cited on page 47).

[NW10] M. J. Nasse and J. C. Woehl. "Realistic Modeling of the Illumination Point Spread Function in Confocal Scanning Optical Microscopy". In: *Journal of the Optical Society of America A* 27.2 (2010), pp. 295–302 (Cited on pages 27, 29).

[Oli+16] A. Olivas et al. *IceCube Simulation Documentation*. University of Wisconsin-Madison. 2016. URL: `http://wiki.icecube.wisc.edu/index.php/Simulation_Documentation_Wiki/` (Cited on page 249).

[Oli02] J. C. Olivo-Marin. "Extraction of Spots in Biological Images Using Multiscale Products". In: *Pattern Recognition* 35.9 (2002), pp. 1989–1996 (Cited on pages 2, 15, 19, 27, 29 sqq., 248).

[Pat05] P. Pattnaik. "Surface Plasmon Resonance". In: *Applied Biochemistry and Biotechnology* 126.2 (2005), pp. 79–92 (Cited on page 12).

[PKC09] J. Pan, T. Kanade, and M. Chen. "Learning to Detect Different Types of Cells under Phase Contrast Microscopy". In: *Microscopic Image Analysis with Applications in Biology (MIAAB)*. 2009, pp. 1–8 (Cited on pages 2, 27 sq., 38, 72, 248).

[PMM+94] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk. "Reducing Misclassification Costs". In: *Proceedings of the Eleventh International Conference on Machine Learning (ICML)*. 1994, pp. 217–225 (Cited on page 153).

[PN06] S. H. R. Pasandideh and S. T. A. Niaki. "Multi-Response Simulation Optimization Using Genetic Algorithm Within Desirability Function Framework". In: *Applied Mathematics and Computation* 175.1 (2006), pp. 366–382 (Cited on page 59).

[Pow11] D. Powers. "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation". In: *Journal of Machine Learning Technologies* 2 (2011), pp. 37–63 (Cited on pages 37, 45, 219, 251 sqq.).

[Pru15] Prudsys AG. *Data Mining Cup (DMC)*. International Student Competition. 2015. URL: `http://www.data-mining-cup.de/` (Cited on page 24).

[RBZ06] M. J. Rust, M. Bates, and X. Zhuang. "Sub-Diffraction-Limit Imaging by Stochastic Optical Reconstruction Microscopy (STORM)". In: *Nature Methods* 3.10 (2006), pp. 793–796 (Cited on page 10).

[RC06] M. Reyes-Sierra and C. C. Coello. "Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art". In: *International Journal of Computational Intelligence Research* 2.3 (2006), pp. 287–308 (Cited on page 48).

[Ris01] I. Rish. "An Empirical Study of the Naive Bayes Classifier". In: *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*. Vol. 3. 22. 2001, pp. 41–46 (Cited on page 167).

[RN03] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 2nd ed. Prentice Hall, 2003 (Cited on pages 157, 161, 167 sq.).

[RSV+13] T. Ruhe, M. Schmitz, T. Voigt, and M. Wornowizki. "DSEA: A Data Mining Approach to Unfolding". In: *Proceedings of the International Cosmic Ray Conference (ICRC)*. 2013 (Cited on page 2).

[Rud99] G. Rudolph. *Evolutionary Search under Partially Ordered Sets*. Tech. rep. Department for Computer Science, LS11, University of Dortmund, Dortmund, Germany, Technical Report CI-67/99, 1999 (Cited on pages 51, 55).

[Ruh13] T. Ruhe. "Data Mining on the Rocks. A Measurement of the Atmospheric Muon Neutrino Flux using IceCube in the 59-String Configuration and a Novel Data Mining Based Approach to Unfolding". PhD thesis. TU Dortmund University, 2013 (Cited on pages 2, 249).

[RW95] J. C. Russ and R. P. Woods. *The Image Processing Handbook*. CRC Press, 1995 (Cited on page 27).

[SD14] H. Seada and K. Deb. *U-NSGA-III: A Unified Evolutionary Algorithm for Single, Multiple, and Many-Objective Optimization*. Tech. rep. Computational Optimization and Innovation Laboratory (COIN), 2014 (Cited on page 194).

[SD15] H. Seada and K. Deb. "U-NSGA-III: A Unified Evolutionary Optimization Procedure for Single, Multiple, and Many Objectives: Proof-of-Principle Results". In: *Evolutionary Multi-Criterion Optimization*. 2015, pp. 34–49 (Cited on page 194).

[Set10] B. Settles. *Active Learning Literature Survey*. Tech. rep. University of Wisconsin-Madison, 2010 (Cited on page 246).

[SFL+14] D. Siedhoff, H. Fichtenberger, P. Libuschewski, F. Weichert, C. Sohler, and H. Müller. "Signal/Background Classification of Time Series for Biological Virus Detection". In: *Pattern Recognition*. Ed. by X. Jiang, J. Hornegger, and R. Koch. Vol. 8753. Lecture Notes in Computer Science. Springer, 2014, pp. 388–398 (Cited on pages 5, 100, 107, 110, 117, 120 sq., 255, 257 sq.).

[SH97] T. Scheffer and R. Herbrich. “Unbiased Assessment of Learning Algorithms”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 1997 (Cited on pages 63, 67 sq., 197).

[SHM+12] J. Schlenke, L. Hildebrand, J. Moros, and J. J. Laserna. “Adaptive Approach for Variable Noise Suppression on Laser-Induced Breakdown Spectroscopy Responses Using Stationary Wavelet Transform”. In: *Analytica Chimica Acta* 754 (2012), pp. 8–19 (Cited on page 187).

[SJS06] M. Sokolova, N. Japkowicz, and S. Szpakowicz. “Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation”. In: *AI 2006: Advances in Artificial Intelligence*. Springer, 2006, pp. 1015–1021 (Cited on page 253).

[SL09] M. Sokolova and G. Lapalme. “A Systematic Analysis of Performance Measures for Classification Tasks”. In: *Information Processing & Management* 45.4 (2009), pp. 427–437 (Cited on pages 44, 127, 253).

[SLN+09] I. Smal, M. Loog, W. Niessen, and E. Meijering. “Quantitative Comparison of Spot Detection Methods in Live-Cell Fluorescence Microscopy Imaging”. In: *IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI)*. 2009, pp. 1178–1181 (Cited on pages 2, 4, 27, 29 sq., 36 sq., 137, 187, 190, 203, 216, 248, 252).

[SLW+14] D. Siedhoff, P. Libuschewski, F. Weichert, A. Zybin, P. Marwedel, and H. Müller. “Modellierung und Optimierung eines Biosensors zur Detektion viraler Strukturen”. In: *Bildverarbeitung für die Medizin 2014*. Springer, 2014, pp. 108–113 (Cited on pages 5, 33, 71, 73, 115, 257 sq.).

[SLW13] D. Siedhoff, P. Libuschewski, and F. Weichert. *Knowledge Extraction and Application in Biosensor Data Analysis*. Poster Presentation at Interdisciplinary College 2013 (IK 2013). 2013 (Cited on pages 5, 258).

[SMB98] J.-L. Starck, F. D. Murtagh, and A. Bijaoui. *Image Processing and Data Analysis: The Multiscale Approach*. Cambridge University Press, 1998 (Cited on pages 29, 91).

[SP94] M. Srinivas and L. M. Patnaik. “Genetic Algorithms: A Survey”. In: *Computer* 27.6 (1994), pp. 17–26 (Cited on page 48).

[SS04] A. J. Smola and B. Schölkopf. “A Tutorial on Support Vector Regression”. In: *Statistics and Computing* 14.3 (2004), pp. 199–222 (Cited on page 163).

[SSK+13] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. “Real-Time Human Pose Recognition in Parts from Single Depth Images”. In: *Communications of the ACM* 56.1 (2013), pp. 116–124 (Cited on pages 33, 73).

[STM+15] V. Shpacovitch, V. Temchura, M. Matrosovich, J. Hamacher, J. Skolnik, P. Libuschewski, D. Siedhoff, F. Weichert, P. Marwedel, H. Müller, K. Überla, R. Hergenröder, and A. Zybin. “Application of Surface Plasmon Resonance Imaging Technique for the Detection of Single Spherical Biological Submicrometer Particles”. In: *Analytical Biochemistry: Methods in the Biological Sciences* 486 (2015), pp. 62–69 (Cited on pages 2, 5, 13, 15, 185, 243, 258 sq.).

[SWL+11] D. Siedhoff, F. Weichert, P. Libuschewski, and C. Timm. "Detection and Classification of Nano-Objects in Biosensor Data". In: *Microscopic Image Analysis with Applications in Biology (MIAAB)*. 2011, pp. 1–6 (Cited on pages 5, 82, 142, 145 sq., 257 sq.).

[Sze06] R. Szeliski. "Image Alignment and Stitching: A Tutorial". In: *Foundations and Trends® in Computer Graphics and Vision* 2.1 (2006), pp. 1–104 (Cited on page 78).

[SZS+14a] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data 100nm_27sep13_exp2*. SFB 876 Project B2. 2014. DOI: 10.15467/e9ofalaebk (Cited on page 187).

[SZS+14b] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data 100nm_27sep13_exp3*. SFB 876 Project B2. 2014. DOI: 10.15467/e9ofomedxc (Cited on page 187).

[SZS+14c] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data 200nm_10apr13*. SFB 876 Project B2. 2014. DOI: 10.15467/e9ofqnvl6o (Cited on page 187).

[SZS+14d] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data 200nm_11apr13_1*. SFB 876 Project B2. 2014. DOI: 10.15467/e9ofylrdvk (Cited on page 187).

[SZS+14e] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data 200nm_11apr13_2*. SFB 876 Project B2. 2014. DOI: 10.15467/e9ofxjfh8g (Cited on page 187).

[TC13] A. Toma and J.-J. Chen. "Computation Offloading for Frame-Based Real-Time Tasks with Resource Reservation Servers". In: *2013 25th Euromicro Conference on Real-Time Systems (ECRTS)*. 2013, pp. 103–112 (Cited on page 247).

[THJ+04] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. "Support Vector Machine Learning for Interdependent and Structured Output Spaces". In: *Proceedings of the 21st International Conference on Machine Learning (ICML)*. 2004, pp. 104–111 (Cited on pages 28, 39).

[Tim12] C. Timm. "Resource-Efficient Processing and Communication in Sensor/Actuator Environments". PhD thesis. TU Dortmund University, 2012 (Cited on page 49).

[TM09] H. Trautmann and J. Mehnen. "Preference-Based Pareto Optimization in Certain and Noisy Environments". In: *Engineering Optimization* 41.1 (2009), pp. 23–38 (Cited on page 59).

[TRS+02] D. Thomann, D. R. Rines, P. K. Sorger, and G. Danuser. "Automatic Fluorescent Tag Detection in 3D with Super-Resolution: Application to the Analysis of Chromosome Movement". In: *Journal of Microscopy* 208.1 (2002), pp. 49–64 (Cited on pages 2, 27, 29 sq., 149, 187, 248).

[TW06] H. Trautmann and C. Weihs. "On the Distribution of the Desirability Index using Harrington's Desirability Function". In: *Metrika* 63.2 (2006), pp. 207–213 (Cited on pages 60 sqq., 195 sq., 258).

[VFB+14] T. Voigt, R. Fried, M. Backes, and W. Rhode. "Gamma-Hadron-Separation in the MAGIC Experiment". In: *Data Analysis, Machine Learning and Knowledge Discovery*. 2014, pp. 115–124 (Cited on pages 2, 248).

[VJ01] P. Viola and M. Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2001, pp. 511–518 (Cited on pages 27, 30).

[VS91] L. Vincent and P. Soille. "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 13.6 (1991), pp. 583–598 (Cited on page 27).

[Wah89] F. M. Wahl. *Digitale Bildsignalverarbeitung: Grundlagen, Verfahren, Beispiele*. Springer, 1989 (Cited on pages 92 sq.).

[WGS+10] C. Wolf, D. Gaida, A. Stuhlsatz, S. McLoone, and M. Bongards. "Organic Acid Prediction in Biogas Plants Using UV/vis Spectroscopic Online-Measurements". In: *Life System Modeling and Intelligent Computing*. Springer, 2010, pp. 200–206 (Cited on page 25).

[WGT+10] F. Weichert, M. Gaspar, C. Timm, A. Zybin, E. Gurevich, M. Engel, H. Müller, and P. Marwedel. "Signal Analysis and Classification for Surface Plasmon Assisted Microscopy of Nanoobjects". In: *Sensors and Actuators B: Chemical* 151.1 (2010), pp. 281–290 (Cited on pages 10 sq.).

[WHS+12] S. Wienert, D. Heim, K. Saeger, A. Stenzinger, M. Beil, P. Hufnagl, M. Dietel, C. Denkert, and F. Klauschen. "Detection and Segmentation of Cell Nuclei in Virtual Microscopy Images: A Minimum-Model Approach". In: *Nature Scientific Reports* 2 (2012), pp. 1–7 (Cited on pages 2, 27 sq., 36 sq., 137, 203, 248, 252).

[WMC10] Q. Wu, F. Merchant, and K. Castleman. *Microscope Image Processing*. Academic Press, 2010 (Cited on page 85).

[WNC07] J. Wang, P. Neskovic, and L. N. Cooper. "Improving Nearest Neighbor Rule with a Simple Adaptive Distance Measure". In: *Pattern Recognition Letters* 28.2 (2007), pp. 207–213 (Cited on page 163).

[WSP+10] S. Wang, X. Shan, U. Patel, X. Huang, J. Lu, J. Li, and N. Tao. "Label-Free Imaging, Detection, and Mass Measurement of Single Viruses by Surface Plasmon Resonance". In: *Proceedings of the National Academy of Sciences* 107.37 (2010), pp. 16028–16032 (Cited on pages 33, 72).

[Wu04] F.-C. Wu. "Optimization of Correlated Multiple Quality Characteristics Using Desirability Function". In: *Quality Engineering* 17.1 (2004), pp. 119–126 (Cited on page 59).

[WW98] J. Weston and C. Watkins. *Multi-Class Support Vector Machines*. Tech. rep. Royal Holloway University of London, 1998 (Cited on page 163).

[YBC+10] Z. Yin, R. Bise, M. Chen, and T. Kanade. "Cell Segmentation in Microscopy Imagery Using a Bag of Local Bayesian Classifiers". In: *IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI)*. 2010, pp. 125–128 (Cited on pages 2, 27 sq., 248).

[Zad65] L. A. Zadeh. “Fuzzy Sets”. In: *Information and Control* 8.3 (1965), pp. 338–353 (Cited on page 95).

[ZBM+07] A. Zybin, D. Boecker, V. Mirsky, and K. Niemax. “Enhancement of the Detection Power of Surface Plasmon Resonance Measurements by Optimization of the Reflection Angle”. In: *Analytical Chemistry* 79.11 (2007), pp. 4233–4236 (Cited on page 12).

[ZBT07] E. Zitzler, D. Brockhoff, and L. Thiele. “The Hypervolume Indicator Revisited: On the Design of Pareto-Compliant Indicators via Weighted Integration”. In: *Evolutionary Multi-Criterion Optimization*. 2007, pp. 862–876 (Cited on page 55).

[ZDT00] E. Zitzler, K. Deb, and L. Thiele. “Comparison of Multiobjective Evolutionary Algorithms: Empirical Results”. In: *Evolutionary Computation* 8.2 (2000), pp. 173–195 (Cited on pages 51, 55).

[ZFS+07] B. Zhang, J. Fadili, J. L. Starck, and J. C. Olivo-Marin. “Multiscale Variance-Stabilizing Transform for Mixed-Poisson-Gaussian Processes and Its Applications in Bioimaging”. In: *IEEE International Conference on Image Processing (ICIP)*. 2007 (Cited on pages 2, 27, 29 sqq., 248).

[Zha12] Z. Zhang. “Microsoft Kinect Sensor and Its Effect”. In: *IEEE MultiMedia* 19.2 (2012), pp. 4–10 (Cited on page 73).

[ZKG+10] A. Zybin, Y. A. Kuritsyn, E. L. Gurevich, V. V. Temchura, K. Überla, and K. Niemax. “Real-Time Detection of Single Immobilized Nanoparticles by Surface Plasmon Resonance Imaging”. In: *Plasmonics* 5.1 (2010), pp. 31–35 (Cited on pages 2, 9 sq., 13, 74, 87, 124, 185, 215).

[ZLT01] E. Zitzler, M. Laumanns, and L. Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK). 2001 (Cited on pages 49, 56).

[ZLX09] Y. Zhou, Y. Li, and S. Xia. “An Improved KNN Text Classification Algorithm Based on Clustering”. In: *Journal of Computers* 4.3 (2009), pp. 230–237 (Cited on pages 107, 109, 113).

[ZRL97] T. Zhang, R. Ramakrishnan, and M. Livny. “BIRCH: A New Data Clustering Algorithm and Its Applications”. In: *Data Mining and Knowledge Discovery* 1.2 (1997), pp. 141–182 (Cited on page 114).

[ZSS+17] A. Zybin, V. Shpacovitch, J. Skolnik, and R. Hergenröder. “Optimal Conditions for SPR-Imaging of Nano-Objects”. In: *Sensors and Actuators B: Chemical* 239 (2017). (Available online 22 July 2016), pp. 338–342 (Cited on pages 73, 185, 187).