utomatic Control
Mechatronics

Mark Schillinger

Safe and Dynamic Design of Experiments

Dissertation

Schriftenreihe der Arbeitsgruppe
Mess- und Regelungstechnik – Mechatronik
Department Maschinenbau

Herausgeber Oliver Nelles

UNIVERSITÄT SIEGEN

Band 4

Impressum
Prof. Dr.-Ing. Oliver Nelles
Arbeitsgruppe Mess- und Regelungstechnik
Department Maschinenbau
Universität Siegen
57068 Siegen
ISSN 2193-0538
URN urn:nbn:de:hbz:467-14768
Zugl.: Siegen, Univ., Diss., 2019

Safe and Dynamic
Design of Experiments

DISSERTATION

zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften

vorgelegt von

Mark Schillinger, M.Sc. RWTH

genehmigt durch die Naturwissenschaftlich-Technischen Fakultät

der Universität Siegen

Siegen 2019

Betreuer und erster Gutachter

Prof. Dr.-Ing. Oliver Nelles

Universität Siegen

Zweiter Gutachter

Prof. Dr.-Ing. Peter Kraemer

Universität Siegen

Tag der mündlichen Prüfung

4. Juli 2019

Preface

This thesis was created during my time as PhD student at Bosch Engineering GmbH in the group Engineering Applications and Methods (EAM1) in Abstatt, Germany. It was supervised by Prof. Dr.-Ing. Oliver Nelles, director of the work group Automatic Control – Mechatronics at the Institute of Mechanics and Control Engineering at the University of Siegen, Germany. I thank both, Bosch Engineering GmbH and the University of Siegen for making this cooperation possible.

Writing this thesis would have been impossible without the support of various people at Bosch, at the University of Siegen, and beyond. I thank Prof. Nelles for his in-depth feedback and his creative and well-versed inputs to my research project and our publications. Many thanks go to my tutor at Bosch Engineering, Dr. Benjamin Hartmann, for his trust and the numerous technical and non-technical discussions we had. I thank my supervisors Kai Grüttner and Martin Jacob for giving me enough freedom for research and supporting me throughout the whole PhD project. My colleagues at EAM1 provided a great working atmosphere which was essential for me during the good and the bad times. Thank you all! Thanks are due to Prof. Kraemer for taking the position of second examiner.

I also thank all student workers whose internship, bachelor, or master thesis I was allowed to supervise and who contributed to this thesis. These are Aleksandar Fandakov, Daniel Kuster, Marissa Füeß, Benedikt Ortelt, Bernhard Sieber, Andreas Wagenpfahl, Mariusz Francik, Patric Skalecki, Lukas Schweizer, and Christian Schlauch. The colleagues at the Bosch Corporate Sector Research and Advance Engineering generously provided answers to my more theoretical questions as well as fundamental research on which parts of this thesis rely. Among others I would like to especially thank Dr. Duy Nguyen-Tuong, Dr. Mona Meister, Jens Schreiter, and Dr. Christoph Zimmer. Additional thank goes to all colleagues at Bosch I had the pleasure to work with and who gave input to my PhD project.

Outside of Bosch I first have to mention the members of Prof. Nelles work group. Despite the large distance between Abstatt and Siegen they always gave me a warm welcome during my visits in Siegen and provided valuable feedback to my work. Many thanks go to Prof. Arif Kazi from Aalen University of Applied Science who helped me to understand the details of controller tuning in the frequency domain. Angelika Sonnenschein generously proofread the introduction and helped to improve my writing style. Thank you very much!

Finally I would like to thank my parents, my family, and my close friend Niels Ole Salscheider for their constant support not only during this PhD project. Without you I would not have found the strength to get where I am today.

Contents

List of Symbols

$\boldsymbol{\Lambda}$	diagonal matrix of length-scales
$\mathbb{N}$	set of all positive whole numbers
$\mathbb{X}$	m-dimensional input space
$\mathbb{X}_+$	feasible subspace of the input space
$\mathbb{X}_-$	infeasible subspace of the input space
$\boldsymbol{\Phi}$	data matrix for ARX parameter optimization
$\boldsymbol{\Sigma}$	covariance matrix
$\boldsymbol{\Sigma}_*$	predicted covariance matrix
$\boldsymbol{a}$	vector from the central point $\boldsymbol{c}$ to the intersection with the estimated boundary or lower limit of scalar input space
a, b	weights of autoregressive or exogenous inputs, respectively
$A(q), B(q), C(q), D(q), F(q)$	polynomials in q^{-1}
A_{pl}	plant's amplitude
$\boldsymbol{b}$	vector from the central point $\boldsymbol{c}$ to the intersection with the hypercube boundary or upper limit of scalar input space
$\boldsymbol{c}$	central point in star-shaped set or vector of measured class labels
$\mathbb{X}_{\text{conv}}, \mathbb{X}_{\text{star}}$	convex and star-shaped sets
$\mathcal{D}$	set of input-output pairs used for training
d	input dimension
d_{z}	number of supervised outputs
f	SAL's regression model of system output
$f(\boldsymbol{x}), g(\boldsymbol{x})$	latent function of system

$f_{\mathrm{NL}}, g_{\mathrm{NL}}$	stationary nonlinear part of a NARX or NFIR model
g	SAL's discriminative model
g_*	predicted discriminative function value
$\mathcal{GP}$	Gaussian Process
$G_{\mathrm{p}}(s)$	transfer function of a PID controller in parallel structure
$G_{\mathrm{s}}(s)$	transfer function of a PID controller in serial structure
H	differential entropy
$\boldsymbol{h}$	vector of measured discriminative function values
$\tilde{h}(\boldsymbol{z})$	risk function
$\boldsymbol{I}_m$	$m \times m$ identity matrix
J	loss function
$J_{\min}$	theoretical minimum of the loss function in case of SBO
$\boldsymbol{K}$	covariance matrix of the training function values
k	discrete time index
$\boldsymbol{k}_*$	vector of covariances between the latent function value $f(\boldsymbol{x}_*)$ and the training function values
k_{**}	prior variance of $f(\boldsymbol{x}_*)$, i.e. $k(\boldsymbol{x}_*, \boldsymbol{x}_*)$
$\boldsymbol{K}_*, \boldsymbol{K}_{**}$	matrices of covariances in case of multiple test points
$k(\boldsymbol{x}, \boldsymbol{x}')$	covariance function between input points $\boldsymbol{x}$ and $\boldsymbol{x}'$
$K_{\mathrm{P,p}}, K_{\mathrm{I,p}}, K_{\mathrm{D,p}}$	proportional, integral, and derivative gain of a PID-controller in parallel structure
$K_{\mathrm{P,s}}$	proportional gain of a PID-controller in serial structure
l	index of summation or discrete time delay
l_{w}	window size of moving average filter
m	number of training samples
m_*	number of test points
m_0	number of previously sampled points
$m_{\mathrm{c}}, m_{\mathrm{h}}$	number of measured class labels and discriminative function labels, respectively

$m_{\mathrm{GP}}(\boldsymbol{x})$	GP's mean function
m_{idx}	size of sparse GP's index set
$m_{\max}$	desired sample size
n	maximum delay or number of samples of a trajectory
n_*	number of points of currently planned trajectory
$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution with mean μ and variance σ^2
n_0	number of level zero upcrossings in the unit interval
$n_{\mathrm{a}}, n_{\mathrm{b}} + 1$	number of autoregressive and exogenous inputs, respectively
n_{past}	number of output samples used in extrapolation model
p	desired minimum probability of feasibility
$\boldsymbol{p}_1, \ldots, \boldsymbol{p}_4$	control points of cubic Bézier curve
q^{-1}	backward shift operator
$\boldsymbol{r}$	vector from the central point $\boldsymbol{c}$ to the original sample
$\boldsymbol{r}'$	vector from the central point $\boldsymbol{c}$ to the scaled sample
$r(k)$	reference trajectory
s	complex frequency
$\bar{u}(k)$	moving average of the actuation signal
$u(k)$	actuation signal
$w(k)$	fade in / fade out weight of moving average filter
$\dot{\boldsymbol{x}}$	vector with derivatives of inputs
$\boldsymbol{X}$	matrix of multiple input points
$\boldsymbol{x}_*$	test or planned input point
$x, \boldsymbol{x}$	scalar or multidimensional input point
$\mathcal{X}_{\mathrm{c}}$	set of candidate points
$\mathcal{X}_{\mathrm{m}}$	set of previously sampled points
$\mathcal{X}_{\mathrm{r}}$	restricted set of points with classifier output between $\phi_{\min}$ and $\phi_{\max}$
$\boldsymbol{y}$	vector of measured system outputs
y	scalar system output
y_*	predicted system output
$\bar{y}$	mean of measured outputs

$y_{*,\mathrm{lim,off}}$	limit for predicted output below which the supervising controller is switched off
$y_{*,\mathrm{lim,on}}$	limit for predicted output above which the supervising controller is switched on
y_{hardmax}	output limit above which emergency shutdown is initiated
$y_{\mathrm{lim,off}}$	limit for measured output below which the supervising controller is switched off
$y_{\mathrm{lim,on}}$	limit for measured output above which the supervising controller is switched on
y_{max}	physically allowed maximum output
y_{minmax}	lowest expected maximum output
y_{set}	setpoint for controller
$\boldsymbol{z}$	vector of supervised system outputs
Δx	extent of a scalar input space
Δy	expected range of output signal
$\Phi^{-1}(x)$	inverse cumulative distributive function of the standard normal distribution
α	line parameter
ε, ζ	zero mean Gaussian noise
$\hat{\boldsymbol{\theta}}$	optimized parameter vector
$\boldsymbol{\theta}$	ARX parameter vector or vector of GP model's hyperparameters
$\theta_{\mathrm{e}}, \theta_{\mathrm{u}}$	tuning parameters of the loss function
κ	tuning factor for Bayesian optimization
λ	length-scale or air-fuel equivalence ratio
$\boldsymbol{\mu}$	vector of mean values
μ_*	predicted mean
μ_0	prior mean
ν	confidence parameter
ξ	design of experiment
$\xi_{\mathrm{mM}}, \xi_{\mathrm{Mm}}$	minimax and maximin designs
σ_*^2	predicted variance
σ_0^2	prior variance of squared exponential kernel
σ^2, τ^2	system's noise variance

σ_{n}^2	GP model's noise variance
$\boldsymbol{\varphi}$	feature or data vector of (N)ARX model
φ_{R}	phase margin
χ	random variable
ω	angular frequency
ω_{c}	controller bandwidth
$\omega_{\mathrm{I,s}}, \omega_{\mathrm{D,s}}, \omega_{\mathrm{T,s}}$	integral, derivative, and proper differential cutoff frequencies of a PID-controller in serial structure
$\omega_{\mathrm{T,p}}$	proper differential cutoff frequency of a PID-controller in parallel structure
ϕ	classifier output
$\phi_{\mathrm{mM}}, \phi_{\mathrm{Mm}}$	minimax and maximin selection criteria

Acronyms

APRBS amplitude modulated pseudo random binary sequence

ARX autoregressive with exogenous input

BJ Box-Jenkins

DoE design of experiments

DTC deterministic training conditional

ECU engine control unit

FIR finite impulse response

GP Gaussian process

GSP global safe point

HiL hardware in the loop

HPFS high pressure fuel supply

MISO multiple-input, single-output

NARX nonlinear autoregressive with exogenous input

NFIR nonlinear finite impulse response

NRMSE normalized root mean square error

ODCM online design of experiment with constraint modeling

OE output error

OMA online measurement automation

RDE	real driving emissions
SAL	safe active learning
SBO	safe Bayesian optimization
SiL	software in the loop
SISO	single-input, single-output

Abstract

In the calibration of modern combustion engines, *stationary* model-based approaches are common. For future developments, the use of *dynamic* data-based modeling is an important and required step to further increase the efficiency and quality of the calibration process. In order to achieve high modeling quality, these models require informative measurement data. This necessitates a transient variation of the input signals during the measurement process. At the same time the safety of the system under test needs to be ensured during the measurements by supervising critical output signals.

This thesis covers different methods to tackle this task, namely for stationary and dynamic design of experiments. Therefore, two strategies are presented and analyzed: on the one hand the use of an offline generated design of experiment, on the other hand the use of active learning. The first strategy is threefold. It starts with the identification of the stationary safe system boundary. Thereafter a dynamic design of experiment is created within the stationary safe boundary. Finally, a dynamic measurement is pursued, while the system under test is safeguarded by a newly introduced supervising controller. The second strategy uses a dynamic safe active learning approach. Thereby, the dynamic model is not anymore learned offline, after the whole measurement was completed, but online, in parallel to the measurement process. Instead of a predefined design of experiment, the queried trajectories are optimized iteratively based on the learned model. The optimization also considers dynamic safety constraints of the system under test.

Special emphasis is put on the application of these strategies to real-world measurements in the combustion engine domain. The various design of experiments, measurement, and active learning algorithms are compared and their specific advantages and disadvantages are discussed.

Kurzfassung

Üblicherweise werden bei der Applikation moderner Verbrennungsmotoren *stationäre* Modelle eingesetzt. Für zukünftige Entwicklungen ist allerdings die Verwendung *dynamischer* datenbasierter Modelle ein wichtiger und notwendiger Schritt, um weitere Steigerungen der Effizienz und der Qualität des Applikationsprozesses zu erreichen. Damit diese Modelle eine gute Qualität erreichen, benötigen sie informative Messdaten. Dies erfordert die transiente Verstellung der Eingangsgrößen während des Messprozesses. Gleichzeitig muss die Sicherheit des zu vermessenden Systems während der Messungen garantiert werden, indem kritische Ausgangsgrößen überwacht werden.

In dieser Arbeit werden verschiedene Methoden für die genannte Aufgabe, nämlich die stationäre und dynamische Versuchsplanung, behandelt. Zwei verschiedene Strategien werden vorgestellt und analysiert: einerseits die Vermessung mit Hilfe eines offline erstellten Versuchsplans, andererseits die Vermessung durch aktives Lernen. Die erste Strategie ist dreigeteilt. Sie beginnt mit der Identifikation der stationär sicheren Systemgrenze, um anschließend einen dynamischen Versuchsplan innerhalb dieser Grenze zu erstellen. Im letzten Schritt wird die dynamische Messung durchgeführt, wobei das zu vermessende System durch einen erstmals vorgestellten Überwachungsregler geschützt wird. Die zweite Strategie verwendet einen Ansatz zum dynamischen sicheren aktiven Lernen. Dabei wird das dynamische Modell nicht erst offline, nach Abschluss der gesamten Messung, sondern online, parallel zur Messung, gelernt. Anstelle eines vordefinierten Versuchsplans werden die zu vermessenden Trajektorien iterativ basierend auf dem gelernten Modell optimiert. In der Optimierung werden auch die dynamischen Betriebsgrenzen des zu vermessenden Systems berücksichtigt.

Der Fokus der Arbeit liegt insbesondere auf der Anwendung der genannten Strategien an realen Messungen für Verbrennungsmotoren. Die verschiedenen Versuchsplanungs-,

Messungs- und aktiven Lernstrategien werden verglichen und ihre spezifischen Vor- und Nachteile diskutiert.

1 Introduction

This PhD project was realized in cooperation with Bosch Engineering GmbH and the Institute of Mechanics and Control Engineering at the University of Siegen. One of the main business segments of Bosch Engineering is the calibration of modern combustion engines. This is a complex task. High customer requirements, strict legal frameworks, high pressure of time, and limited availability of test vehicles demand efficient calibration methods. Physical and data-based models are already widely used in calibration tasks. For example, these models partly replace prototype vehicles, allow for the automatic optimization of calibration parameters or even serve as virtual sensor, or are used to design feed-forward controllers in the engine control unit (ECU). Physical and data-based modeling, however show different strengths and weaknesses. Physical models are compiled from physical relationships like first principles. While they can be interpreted, often reused with similar systems and provide better extrapolation behavior than data-based models, their creation requires deep system knowledge and is often very complex. Simple physical models usually run very fast, but high precision models can require significant computational resources. Data-based models, on the other hand, are created from measured data. Compared to physical models, their creation often requires less effort. Depending on the system's behavior, they can show even better simulation accuracy, as long as representative measurement data is available. Sometimes they are the only option for modeling, because the theoretical background of the system to be modeled is not yet understood.

Static data-based modeling is already frequently used in commercial calibration processes. Dynamic data-based models are less common, though. In contrast to the former, dynamic models not only describe the instantaneous relationship between one or more inputs and outputs. Rather the outputs are dependent on the development of the inputs over time. This is typically described by accounting for the time derivatives (continuous time) or differences (discrete time) in form of differential or difference equations. Most real-world systems have a dynamic behavior. Their steady-state response can often be well approximated with static models, but these models cover only a limited number of use-cases. If the dynamic behavior is very fast, it can be neglected in many applications, for example when modeling engine-out emissions. Modeling also the dynamic behavior opens up new possibilities for model-based calibration. For example, dynamic models can be used for controller tuning. Static models are naturally not sufficient for that application. Furthermore, challenging new statutory requirements like limits on real driving emissions (RDE) call for an optimization of the emissions also in transient conditions. Due to this legislation, emissions need to be evaluated not only on a small number of predefined driving cycles but in a large number of realistic driving situations. By optimizing the engine's behavior in transient conditions like acceleration or a varying accelerator pedal position, emissions, fuel consumption, or the response characteristic can be improved.

A number of different dynamic modeling algorithms have been presented in the literature and evaluated in various real-world applications in the past. They range from rather simple linear transfer-function models to advanced dynamic local model networks or Gaussian processes to very flexible but complex approaches like deep neural networks. Each modeling algorithm has its advantages and disadvantages. For example, a transfer-function model can be fitted from a small amount of measurement data, is fast to evaluate, and can easily be checked for stability. Its limited flexibility makes it suitable for linear systems only or systems which can be linearized around an operating point. Deep neural networks, on the other end of the spectrum, show great results on modeling very complex and high-dimensional data like videos, if enough computational power and training data is available. For low-dimensional problems with little training data their high number of parameters can result in overfitting. Furthermore, the network structure has to be carefully chosen, which requires expert knowledge. Thus, the modeling algorithm should be chosen based on the modeling problem to be solved, as there is no algorithm which fits all needs. In this thesis, mainly Gaussian process models and transfer function models are used.

A major obstacle in the practical application of dynamic modeling is the generation of suitable measurement data. The data-acquisition has to fulfill two main requirements:

1. The collected data should be as informative as possible for the model training

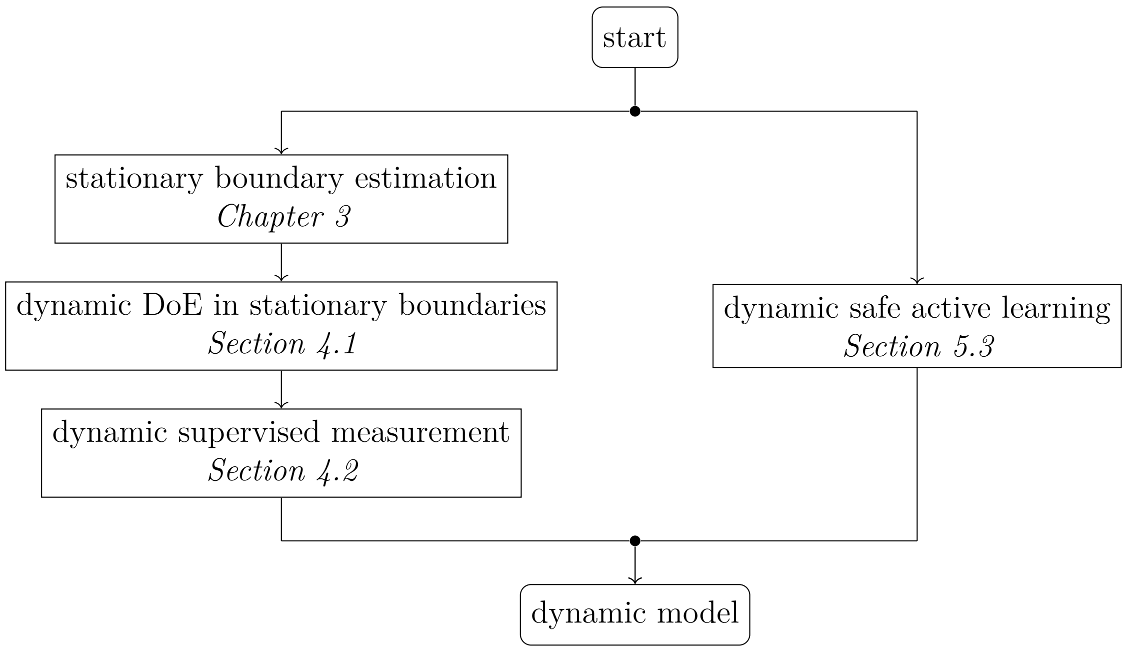
start
stationary boundary estimation
Chapter 3
dynamic DoE in stationary boundaries
Section 4.1
dynamic supervised measurement
Section 4.2
dynamic safe active learning
Section 5.3
dynamic model

Figure 1.1: Basic strategies to obtain a dynamic model proposed in this work.

algorithm.

2. While taking measurements, the system under test must not be damaged by improper excitation signals.

These challenges are addressed in the field design of experiments (DoE). By a smart selection of excitation signals, both goals can be achieved. Especially the second goal requires an iterative DoE, though. Thereby the DoE is not fixed prior to the measurement, but DoE and measurement process are linked and the DoE is adapted or even created completely online. Therefore, suitable automation becomes necessary. Especially in combination with dynamic modeling, this topic still offers open research questions. Some of these are addressed in the PhD thesis at hand. The goal is to evaluate several methods for dynamic but also stationary design of experiments and measurement in real-world applications. Thus, the focus is less on theoretical derivations but more on practical challenges. These naturally arise if methods which were evaluated only in simulation before are applied to real systems for the first time.

Figure 1.1 gives two alternative strategies to obtain a dynamic data-based model, which are presented in this thesis. Both focus on the safe generation of measurement data. The approach on the left side consists of three steps: first, the stationary boundary of the system in the input-space is determined. Methods to do so are presented in Chapter 3. In a second step, a dynamic DoE is scaled within the identified stationary boundary. The main assumption behind this step is that the stationary safe region of the input space is also dynamically safe. The method for scaling is described in Section 4.1. Finally, the actual dynamic measurement is pursued. As an additional safety-layer, the measurement is supervised by a so-called supervising controller. This step is introduced in Section 4.2. In the end, this strategy obtained the data necessary to train a dynamic model within the stationary safe input space.

The alternative approach to these three steps is dynamic safe active learning (SAL). Here, the dynamic input signals are created online. The algorithm gradually explores the input space and avoids infeasible system states. Dynamic SAL is thereby capable of not only collecting measurements in the stationary safe input space, but also in regions where stationary measurements are not feasible, but a short-time dynamic measurement is safe. The method is presented in Section 5.3. Compared to the first approach, dynamic SAL is more sophisticated and powerful, but not yet as proven and tested as the former.

The main contributions of this thesis are as follows:

- Introduction of two enhancements of the existing algorithm *online design of experiment with constraint modeling (ODCM)*: ODCM with improved boundary estimation and ODCM with discriminative risk model.
- Further development and first publication of the dynamic supervised measurement algorithm.
- Derivation of a heuristic for online hyperparameter tuning for stationary SAL.
- Development of a new safe Bayesian optimization algorithm and application to a real-world system.
- Enhancement of dynamic SAL.
- First application of ODCM with improved boundary estimation, ODCM with discriminative risk model, and stationary as well as dynamic SAL to a real-world system.

- First evaluation of ODCM and dynamic supervised measurement at a charge air system.

The thesis is structured as follows:

In Chapter 2, some fundamentals necessary for the remainder of this work are presented. The general process for data-based modeling is discussed. Some common signals used to dynamically excite a system under test are introduced. Afterwards, the important concept of different system constraints is described. The data-based modeling algorithms used in this thesis are briefly explained. Namely these are the basic concept of nonlinear models with external dynamics, transfer function models, and Gaussian process models. Finally, active learning, a method for optimized online design of experiments, is introduced.

Chapter 3 covers stationary measurement algorithms and methods for stationary boundary estimation. This topic receives some additional attention in this thesis. Albeit this area is strictly speaking dealing with stationary instead of dynamic modeling, it is an important step towards dynamic DoE, compare Figure 1.1. Furthermore, depending on the used stationary DoE method, also stationary models can be learned from the data. These may have a valuable application as well. First, general methods for modeling stationary boundaries are described and categorized. Afterwards, the online design of experiment with constraint modeling (ODCM) algorithm is introduced. It is the basis for two new enhancements, which are presented in the following: ODCM with improved boundary estimation generates new queries to improve the accuracy of the boundary estimation. ODCM with discriminative model replaces the classifier used in ODCM with a continuous model known from safe active learning (SAL) to combine the advantages of both methods.

The following Chapter 4 contains two sections. In the first, a method to scale dynamic DoEs in non-convex stationary boundaries is presented. It allows to create dynamic DoEs offline which are safe under the assumption that the stationary boundary was identified correctly and the stationary feasible input space is also dynamically feasible. The second section introduces the dynamic supervised measurement algorithm. It serves as additional safety measure while conducting dynamic experiments. Therefore it finds a trade-off between the original DoE, which was optimized to obtain maximum information, and necessary adaptions to keep safety limits. The three main components of this algorithm are presented in detail.

Chapter 5 covers different variants of the safe active learning algorithm. At the beginning, stationary SAL is introduced. Three main aspects relevant for the practical application of this algorithm are focused: ways to learn the included Gaussian process (GP) models' hyperparameters, some adaptions of the involved discriminative model and a discussion of the risk function which has to be defined for each application. The second part of this chapter deals with a special application of stationary SAL for controller parameter optimization. Additionally, a safe Bayesian optimization (SBO) approach is introduced. After a definition of the required loss and risk functions for this application, the mentioned SBO algorithm is described. Finally, SBO and SAL are compared by naming their conceptual advantages and disadvantages. The last section of this chapter presents dynamic SAL. Thereby, a main focus is put on a real-time capable implementation, which is critical for a practical application of the algorithm. As dynamic SAL is still a topic of research and more theoretical aspects are not covered in this thesis, strengths and weaknesses of its current version are discussed at the end of the chapter and proposals for future research are given.

In Chapter 6, the results of the conducted experiments are presented. At the beginning, a brief overview about the implementation of the covered algorithms for experiments with real systems is given. Subsequently, the measurements at the high pressure fuel supply (HPFS) system are presented. Most of the algorithms described in the previous chapters are tested at this system:

- ODCM,
- ODCM with improved boundary estimation,
- stationary SAL,
- ODCM with discriminative risk model,
- SAL and SBO for controller parameter tuning,
- dynamic supervised measurement, and finally
- dynamic SAL.

After a short description of the system under test, the results of the named methods are presented and discussed. In the last part of the chapter, a second application example for two of the methods is given. The charge air system is a more complex and thus a more practically relevant application. Its main functionality is presented, followed by the results obtained using ODCM and dynamic supervised measurements.

The thesis closes with a summary of the covered topics and obtained results in Chapter 7. Finally, some interesting topics for future research are listed.

2 Fundamentals

In this chapter, some fundamentals necessary in the further course of this thesis will be introduced. They are only presented in short, but references for further reading are given. Section 2.1 gives an introduction to the procedure necessary to obtain a data-based model. Afterwards, different kinds of system constraints are discussed in Section 2.3. Section 2.4 presents several types of data-based models used in this thesis. Gaussian process models and transfer function models are described. The chapter closes with Section 2.5 which gives a short introduction to active learning.

2.1 Data-based Modeling Procedure

The process of data-based modeling can be split in four basic steps (see also Figure 2.1):

1. design of experiments (DoE),
2. measuring,
3. modeling, and
4. application of the model.

In the first step, the inputs and outputs of the system to be modeled and their ranges are defined. Afterwards, suitable input points or trajectories are chosen. They should be as informative as possible, without damaging the system.

In the second step, the planned experiment is applied to the system and the inputs[I] together with the resulting outputs are measured. Therefore, links to the system like

[I]Due to the behavior of the actuators, the originally planned and the measured input signals may be different. Whether measured or planned inputs are used for modeling depends on the application of the model.

software interfaces, actuators, or sensors are required. The measurement process should be automated, allowing to conduct the measurements without human supervision. Furthermore, an automation can supervise the system and avoid critical system states.

The third step has probably received most attention in research. Numerous methods for fitting models to measured data have been developed. They range from simple linear regression to complex deep neural networks. Which modeling approach best fits the application depends on many factors, e.g. the complexity of the system, the amount of available measurement data, existing a priori knowledge, available computing power and time, or the system engineer's experience with certain methods.

Finally, the resulting model can be used for its desired purpose. This comprises, for example, the modification and export of the model to be used in hardware in the loop (HiL) or software in the loop (SiL) environments, or on special purpose hardware like engine control units (ECUs).

Depending on special requirements or a specific focus, this process can be adapted or detailed. For example, [34] expands the modeling procedure into five choices:

3a. model architecture,

3b. dynamics representation (in case of dynamic models),

3c. model order (in case of dynamic models),

3d. model structure and complexity, and

3e. model parameters.

As [61] stresses, data-based modeling can be a recursive and interactive task. For example, a dynamic DoE might need information about the feasible input space, which can be acquired in a stationary pre-identification. Other methods, like the safe active learning algorithm presented in Chapter 5, combine multiple steps: One input point is generated online, measured at the system, and included in model training. Afterwards the algorithm continues with the next point. Thereby, step one to three are merged.

$$\begin{tikzcd}\boxed{\text{design of experiments}} \arrow[r] & \boxed{\text{measuring}} \arrow[r] & \boxed{\text{modeling}} \arrow[r] & \boxed{\text{application of the model}}\end{tikzcd}$$

Figure 2.1: Procedure for data-based modeling.

2.2 Signals for Dynamic Excitation

In order to collect information about the system to be modeled, it can be excited by user-defined input signals. This is not possible for all systems. Applying special signals might be too expensive (for example in case of production machines), dangerous (e.g. in case of aircrafts), or impossible (e.g. in case of unstable systems which need a stabilizing controller). In some cases, it is at least possible to set the system's inputs indirectly, e.g. by varying the setpoint of a control loop.

If it is possible to use a freely chosen input signal, the generation of input signals can be distinguished in four categories. The classical approach is to use an *offline-model-free* DoE. Thereby, the input signal is generated offline, without using a model of the system. Several possible choices of signals are known from the literature. Which signal gives the best result highly depends on the application domain and the used modeling algorithm, compare [34]. In [61] a review of four common dynamic excitation signals is given. These are amplitude modulated pseudo random binary sequences (APRBS), ramp signals, multisine signals, and shifted chirps. See Figure 2.2 for some examples.

Another option are *offline-model-based* DoEs. These approaches try to optimize the input signal before the main measurement is conducted based on a model of the system. This leads to a causality dilemma, as the measurement and data-based modeling is only done to create such a model. Approaches from this category either address special usecases where another model is already available during DoE (e.g. a physical model from component design or a model of a similar system, compare [20]), or use a highly simplified model which can be created easily (e.g. a linear model with one time constant as in [26]).

The third category of DoEs are *online-model-based* approaches. They are also called active learning. Similar to the former named category, the input signals are optimized based on a model of the system. In this case though, the model is identified during the measurement and the optimization of the experimental design is done iteratively.

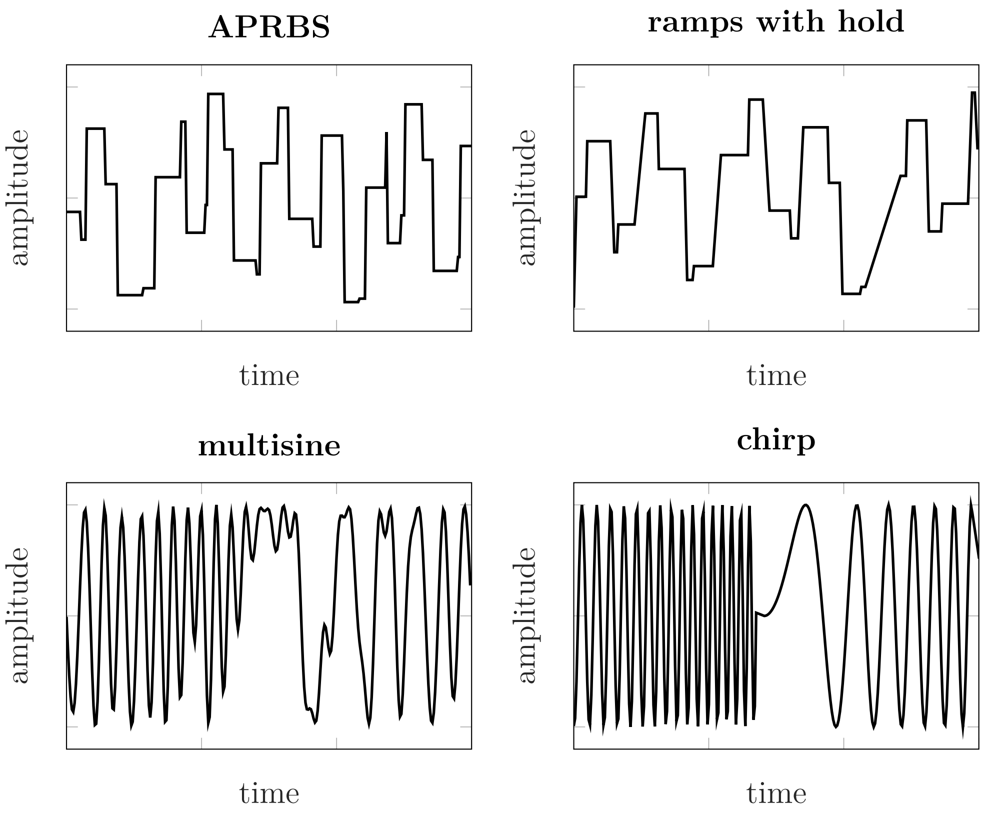

Figure 2.2: 1D-examples of four common dynamic excitation signals. APRBS features random steps and hold times. Ramps are continuous variations of the input with varying gradient and optionally hold times in between. A multisine signal is the sum of multiple sines with harmonically related frequencies, different amplitudes, and phase shifts. A chirp is a sine with varying frequency over time.

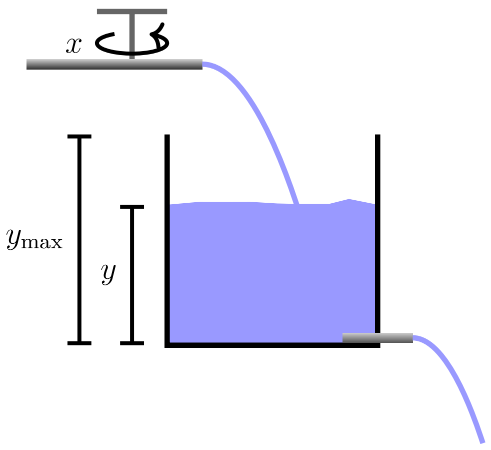

Figure 2.3: Water basin with input valve position x, output water level y and maximum allowed water level $y_{\max}$.

This way, the causality dilemma mentioned above is resolved. DoEs of this category are addressed in more detail in Section 2.5 and Chapter 5.

Finally, there are *online-model-free* DoEs. One example for this category is the dynamic online DoE presented in [61]. In this approach, the input signal is adapted to constraints of the system online, which are found during the measurement.

2.3 Constraints

When exciting a system with special signals, constraints have to be maintained. These constraints can be divided using several criteria:

- input vs. output constraints
- stationary vs. dynamic constraints
- soft vs. hard constraints

Note that this list is no conclusive enumeration.

To illustrate the different kinds of constraints, the system in Figure 2.3 is used. The figure shows a basin filled with water. Fresh water is poured into the basin through a pipe with a valve. On the bottom of the basin there is an output pipe, through which water leaves the basin. By varying the valve position x (the input), the resulting water level y (the output) can be varied. Physics tells us that the water flow leaving the basin depends on the water level [35].

The system has several constraints: The valve position can only be varied between fully open and fully closed. This is an input constraint. To prevent the basin from overflowing, the water level may not exceed a maximum value $y_{\max}$, which defines an output constraint. Another output constraint would be that the basin may never run empty. In order to conduct a safe open loop DoE-measurement, these output constraints have to be translated to input constraints which can be incorporated in the DoE. In stationary case, there is a maximum input which results in a stationary water level just below $y_{\max}$ and a minimum input resulting in a water level slightly above zero.

In the dynamic case, there is another constraint on the input: the maximum allowed input gradient. The valve's position cannot be changed arbitrarily fast. In this case, the water level is time dependent as is the maximum allowed input. If the water level is low, a wide open valve position can be chosen for a short time, before the water level raises to critical values. If the water level is already very high, the input valve may not be opened too much in order to leave enough time to close it again, before the water overflows. In this example, the maximum allowed dynamic valve position is larger or equal to the stationary allowed value. This holds true for many real-world systems. In some cases, nonetheless, the dynamic limits might be smaller than the stationary ones. This could happen, for example, if the system can oscillate and its resonant frequency is excited.

The differentiation between soft and hard constraints is sometimes a little bit fuzzy. For example we could require that the basin may in no case flow over during the measurement (hard limit), while it is ok, even though not encouraged, that it runs empty for a limited time (soft limit). This has an impact on allowed techniques to identify input constraints resulting from output constraints. For example it would be allowed to slowly close the valve until the basin runs empty to measure the minimum allowed input, while this method was not suitable to get the maximum input. In practice, a hard limit can often be made soft by defining a slightly safer value, e.g. a maximum water level below the edge of the basin.

2.4 Data-based Models

As mentioned earlier, many different data-based modeling methods have been presented in the literature. They range from simple linear regression via typical control-engineering-approaches like transfer functions to sophisticated probabilistic models and machine learning approaches like (deep) neural networks. As mentioned in Chapter 1, the modeling algorithm should be chosen based on the application, as no algorithm suits all needs. In this section, only the algorithms and aspects relevant for this thesis are presented.

Throughout this thesis, it is assumed that the output of a stationary system is generated by an unknown function (also called latent function) and additive Gaussian noise, i.e.

$$y(\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon,\ \varepsilon \sim \mathcal{N}(0, \sigma^2). \tag{2.1}$$

Thereby, $\boldsymbol{x} \in \mathbb{R}^d$ represents the d-dimensional input, $y \in \mathbb{R}$ the scalar output, and ε the zero mean Gaussian noise with variance σ^2. In case of stationary systems, the latent function f is time-invariant and only depends on the current input at the discrete time step k, namely $\boldsymbol{x}(k)$. In case of dynamic systems, f, and consequently y, also depend on all previous inputs $\boldsymbol{x}(k-l),\ l \in \mathbb{N}$:

$$y(k) = f(\boldsymbol{x}(k), \boldsymbol{x}(k-1), \boldsymbol{x}(k-2), \dots) + \varepsilon(k). \tag{2.2}$$

Note that in this thesis, $\mathbb{N}$ represents all positive whole numbers, i.e. $\mathbb{N} = \{1, 2, 3, \dots\}$. In case of systems with multiple outputs, each output is treated separately, i.e. described by its own scalar f and ε.

2.4.1 One-Step and Multi-Step Ahead Prediction

A dynamic model depending on all previous inputs as in (2.2) is in most cases not useful in practice. Thus, often only a limited number of previous inputs is used. This is possible, because in most systems the influence of previous inputs on the current output decreases for older inputs, see e.g. [34].

Additionally, previous output values can be used as inputs for the model. As most dynamic systems do not change their output dramatically within one time step, the output feedback gives the model a lot of additional information. Often this leads to good models with only a moderate number of delayed input signals. The drawback of this approach is that the model can become unstable due to the feedback loop.

When using models with output feedback for prediction, two scenarios have to be distinguished: one-step and multi-step ahead prediction. In the first case, the fed back output is measured at the real system. Thus, the model cannot wind up towards infinity and the prediction is usually more accurate compared to the second scenario. As the name suggests, the model is only capable of predicting one step in the future, though.[II] This case is often used for training the model. Multi-step ahead prediction allows to simulate the system behavior over a theoretically unlimited horizon by using predicted outputs for the feedback. Unfortunately, prediction errors can accumulate and, in worst case, make the prediction unstable. Compare Figure 2.4 for the block diagrams of feed-forward, single-step and multi-step prediction models.

2.4.2 Transfer Function Models

Transfer function models are linear, dynamic, time-invariant, scalar black-box models with concentrated parameters. They can be described most compact and universal as

$$A(q)y(k) = \frac{B(q)}{F(q)}x(k) + \frac{C(q)}{D(q)}\varepsilon(k) \tag{2.3}$$

for the single-input, single-output (SISO) case. Thereby, $A(q)$ to $F(q)$ denote polynomials in q^{-1} which is the backward shift operator defined as

$$q^{-1}x(k) = x(k-1). \tag{2.4}$$

Thus, a product of a polynomial in q^{-1} and a signal represents a weighted sum of delayed signal values, e.g.

$$A(q)y(k) = \left(\sum_{l=0}^{n} a_l q^{-l}\right) y(k) = \sum_{l=0}^{n} a_l y(k-l). \tag{2.5}$$

[II]If not the previous but only older system outputs are fed back, more than one step in the future can be predicted. Nonetheless, the number of predictable steps is limited and usually small.

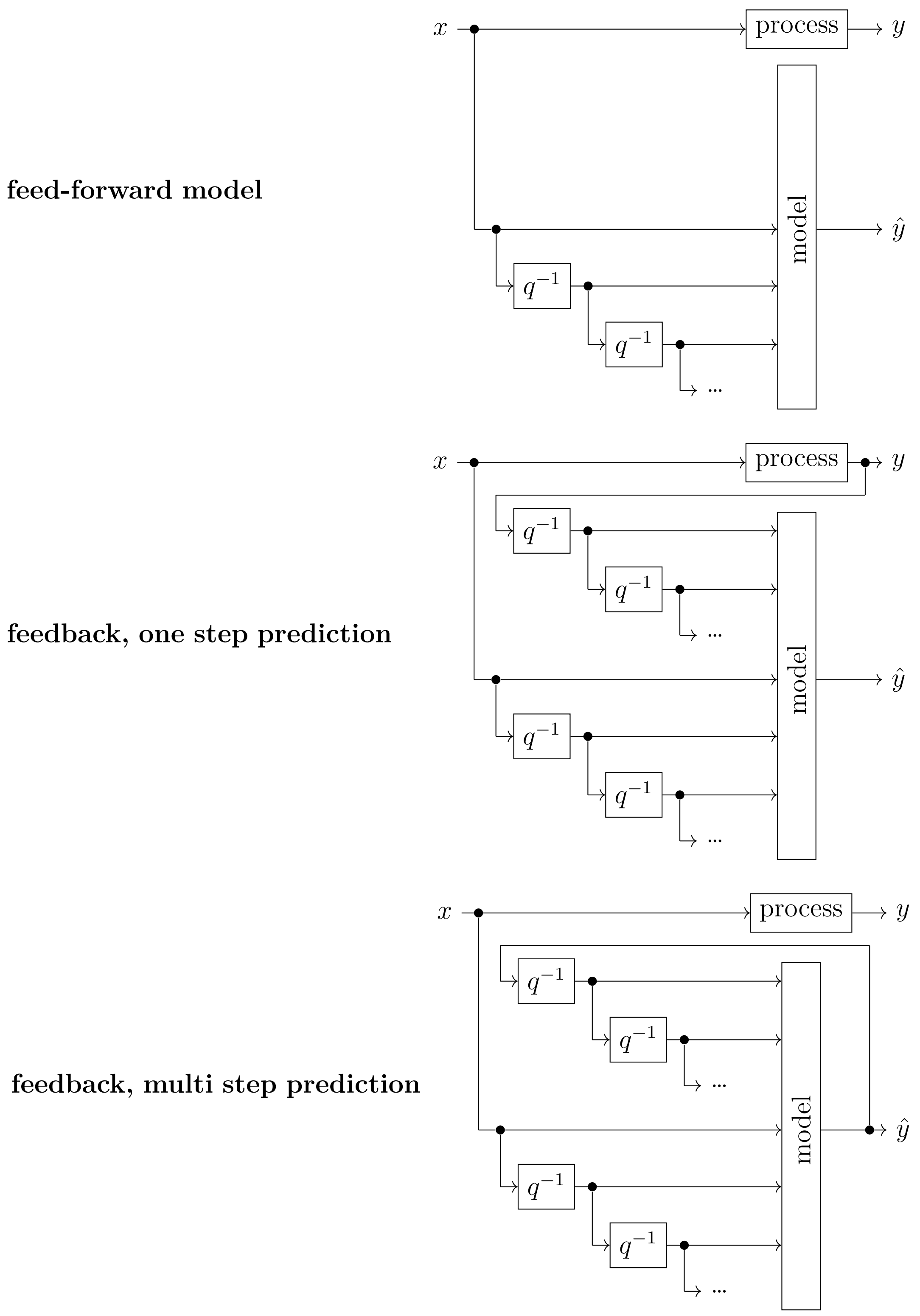

Figure 2.4: Block diagrams of feed-forward and feedback models. q^{-1} is the backward shift operator, compare (2.4).

polynomials used in (2.3)	name of model structure
B	finite impulse response (FIR)
AB	autoregressive with exogenous input (ARX)
ABC	ARMAX
AC	ARMA
ABD	ARARX
ABCD	ARARMAX
BF	output error (OE)
BFCD	Box-Jenkins (BJ)

Table 2.1: Some special cases of transfer function models. The unused polynomials are set to unity. The table was taken from [31].

Depending on which of the polynomials are present, different models are distinguished (see Table 2.1). By leaving out some of the polynomials the complexity of the model can be reduced, if the behavior of the system allows such simplifications.

In Section 4.2 an ARX model will be used. Thus, the procedure how to train this special case of a transfer function model is explained in the following, based on [31]. According to (2.3), (2.5), and Table 2.1, an ARX model can be written as

$$y(k) + a_1 y(k-1) + \dots + a_{n_\mathrm{a}} y(k-n_\mathrm{a}) = b_0 x(k) + \dots + b_{n_\mathrm{b}} x(k-n_\mathrm{b}) + \varepsilon(k) \tag{2.6}$$

$$\Leftrightarrow y(k) = b_0 x(k) + \dots + b_{n_\mathrm{b}} x(k-n_\mathrm{b}) - a_1 y(k-1) - \dots - a_{n_\mathrm{a}} y(k-n_\mathrm{a}) + \varepsilon(k), \tag{2.7}$$

where n_a is the number of autoregressive and $n_\mathrm{b}+1$ the number of exogenous inputs. When predicting using an ARX model, it is usually assumed that the noise $\varepsilon(k)$ equals zero, see [31]. This leads to

$$y_*(k) = b_0 x(k) + \dots + b_{n_\mathrm{b}} x(k-n_\mathrm{b}) - a_1 y(k-1) - \dots - a_{n_\mathrm{a}} y(k-n_\mathrm{a}) \tag{2.8}$$

$$\Leftrightarrow y_*(k) = \boldsymbol{\theta}^\mathrm{T} \boldsymbol{\varphi}(k) = \boldsymbol{\varphi}^\mathrm{T}(k) \boldsymbol{\theta} \text{ with the parameter vector} \tag{2.9}$$

$$\boldsymbol{\theta}^\mathrm{T} = [a_1, \dots, a_{n_\mathrm{a}}, b_0, \dots, b_{n_\mathrm{b}}] \text{ and the data vector} \tag{2.10}$$

$$\boldsymbol{\varphi}^\mathrm{T}(k) = [-y(k-1), \dots, -y(k-n_\mathrm{a}), x(k), \dots, x(k-n_\mathrm{b})]. \tag{2.11}$$

The task of the training algorithm is to minimize the prediction error between measured outputs $y(k)$ and predicted outputs $y_*(k)$ by tuning the parameter vector $\boldsymbol{\theta}$ on a set of m training samples. A common approach to do so is the least-squares method. It minimizes the quadratic mean of the prediction error, i.e.

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{m} \sum_{k=1}^{m} \left(y(k) - y_*(k)\right)^2 . \tag{2.12}$$

The measured data can be combined in a data matrix

$$\boldsymbol{\Phi}^{\mathrm{T}} = [\boldsymbol{\varphi}(1), \ldots, \boldsymbol{\varphi}(m)] \tag{2.13}$$

and an output vector

$$\boldsymbol{y}^{\mathrm{T}} = [y(1), \ldots, y(m)], \tag{2.14}$$

transforming (2.12) to

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left(\boldsymbol{y} - \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\theta}\right)^{\mathrm{T}} \left(\boldsymbol{y} - \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\theta}\right), \tag{2.15}$$

where the factor $\frac{1}{m}$ was omitted, as it is independent of $\boldsymbol{\theta}$ and thus does not contribute to the result of the minimization. Fortunately, this minimization problem is quadratic in $\boldsymbol{\theta}$ and can thus be solved analytically. The result is the least-squares estimate

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}. \tag{2.16}$$

ARX models and their training can be extended to multiple-input, single-output (MISO) systems. In case of d inputs, the parameter vector and the data vector are extended to

$$\boldsymbol{\theta}^{\mathrm{T}} = [a_1, \ldots, a_{n_\mathrm{a}}, b_{1,0}, \ldots, b_{1,n_\mathrm{b}}, b_{2,0}, \ldots, b_{d,n_\mathrm{b}}] \text{ and} \tag{2.17}$$

$$\boldsymbol{\varphi}^{\mathrm{T}}(k) = [-y(k-1), \ldots, -y(k-n_\mathrm{a}), x_1(k), \ldots, x_1(k-n_\mathrm{b}), x_2(k), \ldots, x_d(k-n_\mathrm{b})]. \tag{2.18}$$

The training can be done analogously to the SISO case using the least-squares method.

2.4.3 Stationary Gaussian Process Regression

Gaussian process (GP) models are a nonparametric Bayesian modeling approach. Opposed to parametric models like transfer functions, GP models represent the system's latent function $f(\boldsymbol{x})$ by a stochastic process, the eponymous Gaussian process

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m_{\mathrm{GP}}(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')) \tag{2.19}$$

with mean function $m_{\mathrm{GP}}(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$. Formally, a GP is defined as a collection of random variables, any finite number of which have a joint Gaussian distribution, see [39]. That is, for each finite number m of latent function values $\boldsymbol{f} = [f_1, \dots, f_m]$ it holds

$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.20}$$

where $\boldsymbol{\mu} \in \mathbb{R}^m$ is a vector of mean values and $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times m}$ a covariance matrix.

A priori, i.e. before taking any training data into account, a Gaussian distribution with constant mean $m_{\mathrm{GP}}(\boldsymbol{x}) = \mu_0$ and covariance function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is assumed for any set of function evaluations. This GP is called *prior*. In most cases the prior mean is set to $\mu_0 = 0$ without loss of generality, which is also presumed in the following. The process is subsequently conditioned on the training data to obtain the *posterior*. Thanks to the probabilistic framework this is possible analytically for regression models and yields the predictive distribution

$$p(y_*|\boldsymbol{x}_*, \mathcal{D}) = \mathcal{N}(\mu_*, \sigma_*^2), \text{ with } \begin{cases} \mu_* = \boldsymbol{k}_*^{\mathrm{T}}(\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}_m)^{-1} \boldsymbol{y} \\ \sigma_*^2 = \sigma_{\mathrm{n}}^2 + k_{**} - \boldsymbol{k}_*^{\mathrm{T}}(\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}_m)^{-1} \boldsymbol{k}_*, \end{cases} \tag{2.21}$$

see [39]. Thereby, $\mathcal{D} = \{\boldsymbol{x}_i, y_i | i = 1, \dots, m\}$ denotes the m input-output pairs used for training, $\boldsymbol{k}_* \in \mathbb{R}^{1 \times m}$ the vector of covariances between the latent function value $f(\boldsymbol{x}_*)$ at the test point $\boldsymbol{x}_*$ and the training function values, $\boldsymbol{K} \in \mathbb{R}^{m \times m}$ the covariance matrix of the training function values, σ_{n}^2 the noise variance, $\boldsymbol{I}_m$ the $m \times m$ identity matrix, $\boldsymbol{y} \in \mathbb{R}^{1 \times m}$ the vector of training outputs, and $k_{**} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$ the prior variance of $f(\boldsymbol{x}_*)$. $\boldsymbol{k}_*$ and $\boldsymbol{K}$ are composed using the covariance function as

$$\boldsymbol{k}_* = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_*) \\ \vdots \\ k(\boldsymbol{x}_m, \boldsymbol{x}_*) \end{bmatrix} \quad \text{and} \quad \boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_m) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_m, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_m, \boldsymbol{x}_m) \end{bmatrix}. \tag{2.22}$$

Note that in some special cases, e.g. when using GP models for classification, the posterior distribution is not Gaussian anymore. Thus, it cannot be calculated analytically in these cases. One possibility to deal with this problem is to approximate the posterior with a Gaussian distribution using *Laplace approximation*. For details see [64].

The prediction in (2.21) can also be done for multiple test points $\boldsymbol{x}_{1*}, \dots, \boldsymbol{x}_{m_**}$ at once. In this case, the vector $\boldsymbol{k}_*$ becomes a matrix $\boldsymbol{K}_* \in \mathbb{R}^{m \times m_*}$, where m_* is the number of test points. The scalar k_{**} also becomes a matrix $\boldsymbol{K}_{**} \in \mathbb{R}^{m_* \times m_*}$, containing the covariances of the test function values. The noise variance has to be added to all predicted variances, which is done by introducing the $m_* \times m_*$ identity matrix $\boldsymbol{I}_{m_*}$. This yields

$$\begin{aligned}\boldsymbol{K}_* &= \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_{1*}) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_{m_**}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_m, \boldsymbol{x}_{1*}) & \cdots & k(\boldsymbol{x}_m, \boldsymbol{x}_{m_**}) \end{bmatrix} \quad \text{and} \\ \boldsymbol{K}_{**} &= \begin{bmatrix} k(\boldsymbol{x}_{1*}, \boldsymbol{x}_{1*}) & \cdots & k(\boldsymbol{x}_{1*}, \boldsymbol{x}_{m_**}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_{m_**}, \boldsymbol{x}_{1*}) & \cdots & k(\boldsymbol{x}_{m_**}, \boldsymbol{x}_{m_**}) \end{bmatrix}\end{aligned} \tag{2.23}$$

transforming (2.21) into

$$\begin{aligned}\boldsymbol{\mu}_* &= \boldsymbol{K}_*^{\mathrm{T}}(\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}_m)^{-1} \boldsymbol{y} \\ \boldsymbol{\Sigma}_* &= \sigma_{\mathrm{n}}^2 \boldsymbol{I}_{m_*} + \boldsymbol{K}_{**} - \boldsymbol{K}_*^{\mathrm{T}}(\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}_m)^{-1} \boldsymbol{K}_*.\end{aligned} \tag{2.24}$$

Consequently, the predicted mean is a vector $\boldsymbol{\mu}_* \in \mathbb{R}^{m_*}$ and the predicted variance a matrix $\boldsymbol{\Sigma}_* \in \mathbb{R}^{m_* \times m_*}$. The latter not only contains the predicted autocovariance of the model output at each test point (on its main diagonal), but also the covariance between the outputs at the different test points.

The covariance function is usually parameterized by a set of hyperparameters. For example, when using a squared exponential covariance function

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_0^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathrm{T}} \boldsymbol{\Lambda}^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)\right), \text{ with } \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1^2, \dots, \lambda_d^2), \tag{2.25}$$

the hyperparameters are the prior variance σ_0^2, the length-scales $\lambda_1, \dots, \lambda_d$, and the

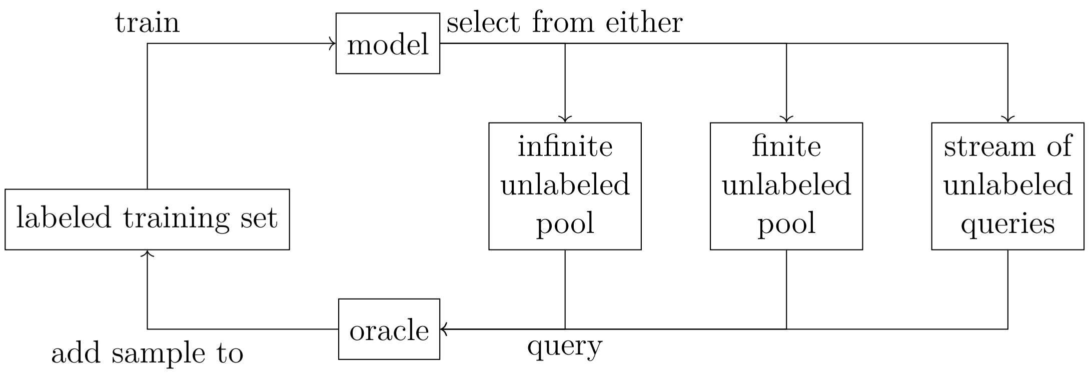

Figure 2.5: Basic process of active learning with the three main scenarios. The figure is inspired by [53].

noise variance[III] σ_n^2. They can be learned by maximizing the log marginal likelihood

$$\log p(\boldsymbol{y}|\theta) = -\frac{1}{2}\boldsymbol{y}^\mathrm{T}(\boldsymbol{K}+\sigma_\mathrm{n}^2\boldsymbol{I})^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}+\sigma_\mathrm{n}^2\boldsymbol{I}| - \frac{m}{2}\log 2\pi. \qquad (2.26)$$

More information can be found in [39].

For dynamic modeling with GP models, an external dynamics approach can be used. More specific, a nonlinear autoregressive with exogenous input (NARX) structure is utilized in this contribution. Therefore, a time-dependent feature vector $\boldsymbol{\varphi}(k)$ is constructed, using delayed system inputs x_i and outputs y:

$$\boldsymbol{\varphi}^\mathrm{T}(k) = \big[y(k-1),\dots,y(k-n),x_1(k),\dots,x_1(k-n),x_2(k),\dots,x_d(k-n)\big], \quad (2.27)$$

where n is the maximum output and input delay. Note that not all $x_i(k-l)$ and $y(k-l), i \in [1,d], l \in [0,n]$ have to be present in $\boldsymbol{\varphi}(k)$. This feature vector is used as input to a nonlinear stationary GP model. Thus, the model output becomes $y_* = f_\mathrm{NL}(\boldsymbol{\varphi}(k))$. See [34] for more details.

2.5 Active Learning

Active learning is a subfield of machine learning. Its main idea is that a modeling algorithm can pick a query from a pool, ask an oracle to label the query and thereby

[III]The noise variance is no parameter of the covariance function, but plays an analogous role. Thus, it is considered as an additional hyperparameter here, similar to [39].

improve its modeling accuracy and reduce the necessary amount of training data. This is opposed to classical generation of training data, where the queries are defined in a DoE before the labeling process is started. Active learning aims at the case where unlabeled queries are very cheap to obtain, but labeling them is expensive. By optimizing the queries during the measurement procedure, the number of required labels to obtain a desired model quality should be reduced. The basic process is shown in Figure 2.5.

In the following, the single steps are described in more detail and specified for the identification problems considered in this thesis. An unlabeled query is an input point to the system and the model. In the example from Figure 2.3 that would be a single valve position. The query is selected either from a finite or infinite size unlabeled pool, which contains e.g. some discrete valve positions or all real number valve positions between the fully closed and fully open position. The third option is the selection from a stream of queries, where the algorithm can only decide to choose or omit the current query. The oracle is an instance which labels the queries and thereby transforms them to samples, i.e. pairs of a query and the resulting system output. In the above example, the oracle would be the basin test bench, where a valve position is applied and the resulting water level is measured. The labeled training set contains all samples measured so far. The training algorithm depends on the modeling algorithm in use, compare e.g. Section 2.4.

Regarding the source of the unlabeled queries, three main scenarios are common, see [53]. In case of *membership query synthesis*, the queries are originally generated by the algorithm, which is equal to selecting from an infinite size pool. These methods are especially promising if the oracle is a machine or experiment and no human. The latter can be confused or overstrained by artificially generated queries, for example when labeling pictures, videos or speech. Thus, *pool-based sampling* methods with a finite size pool were developed. In case of *stream-based selective sampling*, the algorithm may only choose whether it wants the current query to be labeled or not. In the two previous scenarios, the queries can be chosen in an arbitrary order. Stream-based selective sampling can be especially useful in case of limited memory or processing power.

The most interesting part of the process is the algorithm that selects the queries from the pool or stream. Settles [53] suggests a classification of the query strategies known from the literature in six categories:

Uncertainty sampling queries the instance which is least certain. This approach is comparatively simple for probabilistic models and will be discussed in more detail below.

Query-by-committee uses different models which predict the output at an instance. The instance where the committee disagrees most is queried. This approach minimizes the model's version space, i.e. the set of hypotheses consistent with the available labeled training data.

Expected model change selects the instance which would change the current model most if its label was known. The approach can be applied to all models with gradient-based training. For those models, the instance which maximizes the expected amplitude of the new gradient is chosen. Intuitively, expected model change prefers queries which likely influence the model most.

Expected error reduction chooses the instance which is likely to reduce the generalization error most. The future error is estimated on all other unlabeled instances. Another interpretation of this approach is that the expected information gain or, equivalently, the mutual information of the query is maximized.

Variance reduction aims at minimizing the output variance of the model. This approach is similar to the previous one. For some models the output variance can be calculated in closed form in contrast to the expected generalization error. Thus, variance reduction is less computationally demanding.

Density-weighted methods weight the results from a base query strategy by the average similarity to all other instances in the input distribution. The idea behind the weighting is that queries should not only be uncertain, but also representative for the underlying distribution of samples. Thereby outliers in the input distribution are avoided, which are preferred by some simpler methods.

The algorithm used in Chapter 5 uses uncertainty sampling, which is a frequent choice in the literature, according to [51]. Thus, the following description focuses on this category. The basic idea of this approach is to sample new queries where the model is least certain which output value to predict. This is rather simple in case of two-class-classification. There, the instance whose prediction is closest to the decision boundary is chosen. A more general concept is to query the next inputs $\boldsymbol{x}_{i+1}$ where the differential entropy H of the predicted output y_* is maximal, i.e.

$$\boldsymbol{x}_{i+1} = \underset{\boldsymbol{x}_* \in \mathbb{X}}{\operatorname{argmax}}\, H(y_* | \boldsymbol{x}_*, \mathcal{D}). \tag{2.28}$$

Thereby, $\mathbb{X} \subset \mathbb{R}^m$ denotes the m-dimensional input space. Entropy is a term originating from information theory, compare [54]. It describes the amount of information contained in a message. In the same publication, Shannon defined the continuous entropy of a random variable χ, which was later called differential entropy, as

$$H(\chi) = -\int_{-\infty}^{\infty} p(\chi) \log p(\chi) \mathrm{d}\chi. \tag{2.29}$$

Intuitively, the differential entropy $H(y_* | \boldsymbol{x}_*, \mathcal{D})$ is a measure for the information the sample y_* would deliver, depending on the input $\boldsymbol{x}_*$ and the previous measurement data $\mathcal{D}$. Thus, by querying at maximum entropy, the most informative measurement is conducted.

In case of Gaussian process models, the differential entropy is

$$H(y_* | \boldsymbol{x}_*, \mathcal{D}) = \frac{1}{2} \log \left(2\pi \mathrm{e} \sigma_*^2\right), \tag{2.30}$$

which is monotonically increasing in the posterior variance σ_*^2, see [51]. Thus, selecting the query with maximum entropy is equal to selecting the query with maximum variance, i.e.

$$\boldsymbol{x}_{i+1} = \underset{\boldsymbol{x}_* \in \mathbb{X}}{\operatorname{argmax}}\, \sigma_*^2(\boldsymbol{x}_*). \tag{2.31}$$

This formulation is directly applicable, as the predicted output variance σ_* can be computed using (2.21).

3 Stationary Measurements and Online Estimation of Stationary Boundaries

In this chapter, a method for stationary online design of experiments with parallel estimation of the system's boundaries in the input space is presented. The method is called *online design of experiment with constraint modeling (ODCM)* and was first introduced in [23]. ODCM delivers two results: First, space-filling measurements of the system under test are obtained. These can be used for stationary data-based modeling. Second, a classifier is provided, which can distinguish feasible from infeasible input points. This classifier can be used in a subsequent step for example to create a dynamic DoE in stationary boundaries, which is a good starting point for dynamic measurements at the system (see the following Chapter 4).

This chapter is structured as follows. Section 3.1 discusses different methods how to describe and find the stationary boundaries of a system. In Section 3.2 the aforementioned ODCM algorithm is described in more detail. Afterwards, in Section 3.3 and 3.4 two enhancements of ODCM are presented, which were developed during this research project. These are an improved boundary estimation method as well as a combination of ODCM and stationary safe active learning (SAL).

3.1 Stationary Boundary Estimation

When performing stationary measurements at a system for data-based modeling, boundary violations should be avoided. For this purpose, an estimate of the system's boundary can be used. Especially in the combustion engine calibration domain, several boundary estimation approaches have been presented. Some of them are already used in industrial applications.

These methods can be categorized using several criteria:

- Is the boundary estimation done in advance or in parallel with the measurements,
- is the approach screening or non-screening, and
- which classification algorithm is used.

Some approaches estimate the boundary in advance, e.g. [14, 25, 33, 40]. While these approaches are comparatively simple, they usually have the disadvantage of requiring additional measurement time. Often the measurements conducted for the boundary estimation cannot be used for modeling purposes, as steady state conditions were not anticipated or the measured points are not distributed spacefillingly. Online DoE methods like ODCM, SAL (see Chapter 5) or the methods proposed in [14, 27, 43, 44], to name a few, try to overcome this drawback.

Most boundary identification methods try to avoid boundary violations, but cannot guarantee an always safe operation. This problem is intrinsic, as the exact boundaries are unknown a priori (otherwise the boundary estimation would be unnecessary). Thus, a supervision of the system under test is necessary. In case of vehicle or engine measurements, this supervision is usually performed by the test bench. The behavior of the boundary estimation algorithm in case of a limit violation distinguishes screening and non-screening procedures. While the first re-plan an input point if a violation occurs, in order to get as close to the boundary as possible, the latter simply skip the point, see [44]. Screening methods might be able to give a better estimate of the boundary, but also require more time and can deteriorate a potential space-filling distribution of the originally planned points (the last argument will be discussed in more detail below). Four of the references mentioned in the previous paragraph are screening [25, 27, 40, 43], while [44] is non-screening. The references [14, 33] both use two approaches which are either screening or non-screening. ODCM is also a non-screening approach, except for ODCM with improved boundary estimation presented in Section 3.3. SAL can be screening or non-screening, depending on the implementation.

At this point the definition of space-filling distributions becomes important. As summarized in [38], different criteria for space-filling designs exist. Depending on

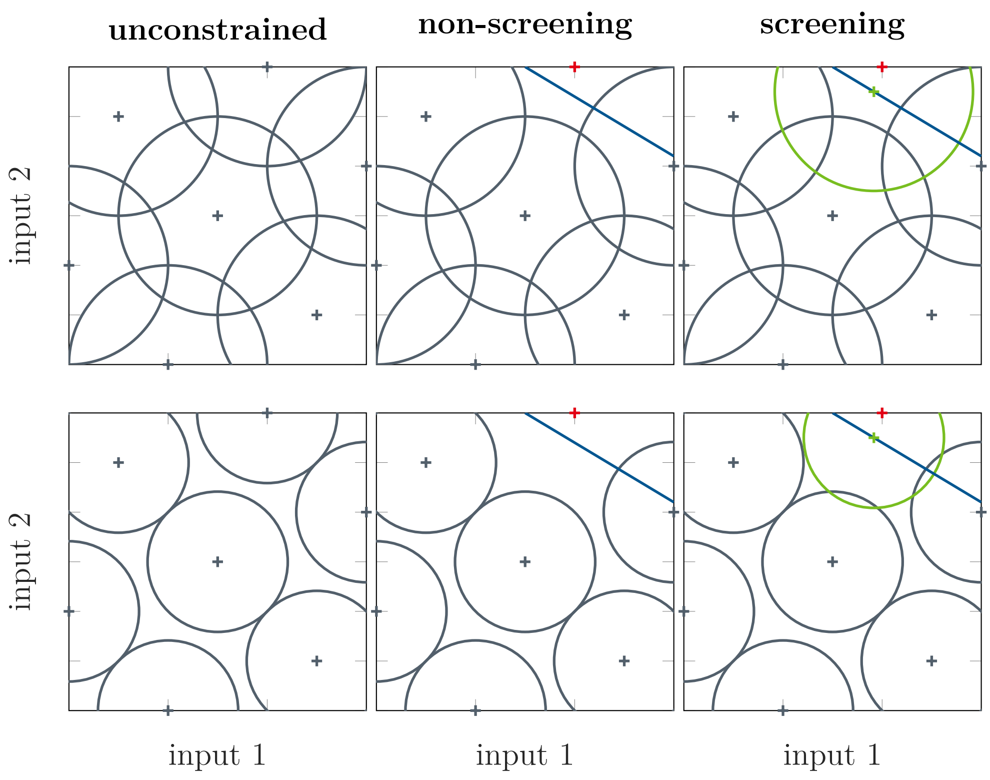

Figure 3.1: Comparison of the space-filling criteria minimax (upper plots) and maximin (lower plots) in case of non-screening (middle column) and screening (right column) boundary estimation methods. The DoEs were taken from [10]. The blue line denotes a hypothetical system boundary, the green point is a replacement for the red point which violates the boundary.

which criterion is considered, a space-filling design might benefit from either screening or non-screening procedures. This is illustrated in Figure 3.1 with the examples of the minimax and the maximin criteria. The first ($\phi_{\mathrm{mM}}(\xi)$) evaluates the distances of all points in the input space $\mathbb{X}$ to their closest selected point in the design $\xi = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{m_{\max}}\}$, i.e.

$$\phi_{\mathrm{mM}}(\xi) = \max_{\boldsymbol{x} \in \mathbb{X}} \min_{\boldsymbol{x}_i \in \xi} \|\boldsymbol{x} - \boldsymbol{x}_i\| \tag{3.1}$$

An optimal minimax design ξ_{mM} *minimizes* these distances, i.e.

$$\xi_{\mathrm{mM}} = \underset{\xi \subset \mathbb{X}}{\operatorname{argmin}}\, \phi_{\mathrm{mM}}(\xi). \tag{3.2}$$

In other words, minimax tries to cover the whole input space using balls with minimum radius, see [38] and compare the upper left plot in Figure 3.1. If a non-screening procedure is used (upper middle plot), in the presence of a boundary an uncovered area occurs and the criterion $\phi_{\mathrm{mM}}(\xi)$ deteriorates, i.e. the remaining balls would need to become a lot larger to cover the whole space again. Using a screening procedure (upper right plot), the uncovered space is smaller and the space-filling minimax criterion becomes only little worse.

The maximin criterion $\phi_{\mathrm{Mm}}(\xi)$ pursues another strategy: an optimal maximin design ξ_{Mm} *maximizes* the distance from each point in the design to its nearest neighbor, which results in

$$\phi_{\mathrm{Mm}}(\xi) = \min_{\boldsymbol{x}_i, \boldsymbol{x}_j \in \xi, \boldsymbol{x}_i \neq \boldsymbol{x}_j} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|. \tag{3.3}$$

$$\xi_{\mathrm{Mm}} = \underset{\xi \subset \mathbb{X}}{\operatorname{argmin}}\, \phi_{\mathrm{Mm}}(\xi). \tag{3.4}$$

Speaking in the ball-example used above, maximin fills the input space with balls of maximum radius without overlap, see lower left plot. The influence of screening or non-screening boundary estimation procedures is opposed to minimax, though. Non-screening procedures don't affect the criterion (lower middle plot), while a moved ball in case of a screening procedure intersects with other balls (lower right plot), leading to a worse measure ϕ_{Mm}. In both cases, the screening procedure queries the replacement point directly at the boundary on a line from the center of the input space to the infeasible point. This is only an example and usually not exactly possible in practice.

Finally, the boundary estimation methods can be categorized based on their used classification algorithm. Here, three main categories can be distinguished: one-class classification (also called hull estimation), two-class classification, and regression models. One-class classification tries to distinguish samples with certain properties from all other samples, while using only samples of one class as training data. In case of boundary estimation, only the measured feasible input points are used for training. Probably the most classical example of this category (disregarding even simpler hypercube or hyperellipsoid approaches) is the convex hull, see e.g. [3]. The convex hull is the smallest convex set which contains all points in a given set. Nonetheless, the convex hull is computationally expensive in case of a high number of inputs and naturally over- or underestimates non-convex feasible input spaces. Thus, non-convex hull algorithms were developed, see e.g. [13] for an overview. More advanced one-class classifiers are also presented in [59]. The references [14, 25, 33, 43] all use one-class classification. In [40] a mixture of convex hulls and regression models is utilized.

Two-class classifiers (also called binary classifiers) use the information which samples were measured as feasible and which were infeasible. Thereby, the quality of the boundary prediction can often be improved, as the outer feasible points do not need to be especially close to the boundary. These methods require not only the information which points were measured as feasible, but also some infeasible points. There are many binary classification algorithms known in the literature, e.g. k-nearest neighbors, Gaussian process models or support vector machines. See for example [6, 24] for details. Reference [27] and ODCM apply two-class classifiers.

Some algorithms, e.g. [14, 44] and SAL, use one or more regression models for boundary estimation. Thereby, the boundary model does not only use the information which measured points were feasible or infeasible, but also additional continuous information like system outputs or a calculated health status. Thereby, the estimation of the boundary can be improved. Especially the necessary extrapolation during an online DoE can benefit from continuous training data, reducing the number of infeasible samples. See also Chapter 5.

A comparison of different classification algorithms is given in Figure 3.2. They are all used on a set of 80 points from a Sobol sequence. The feasibility of the points is

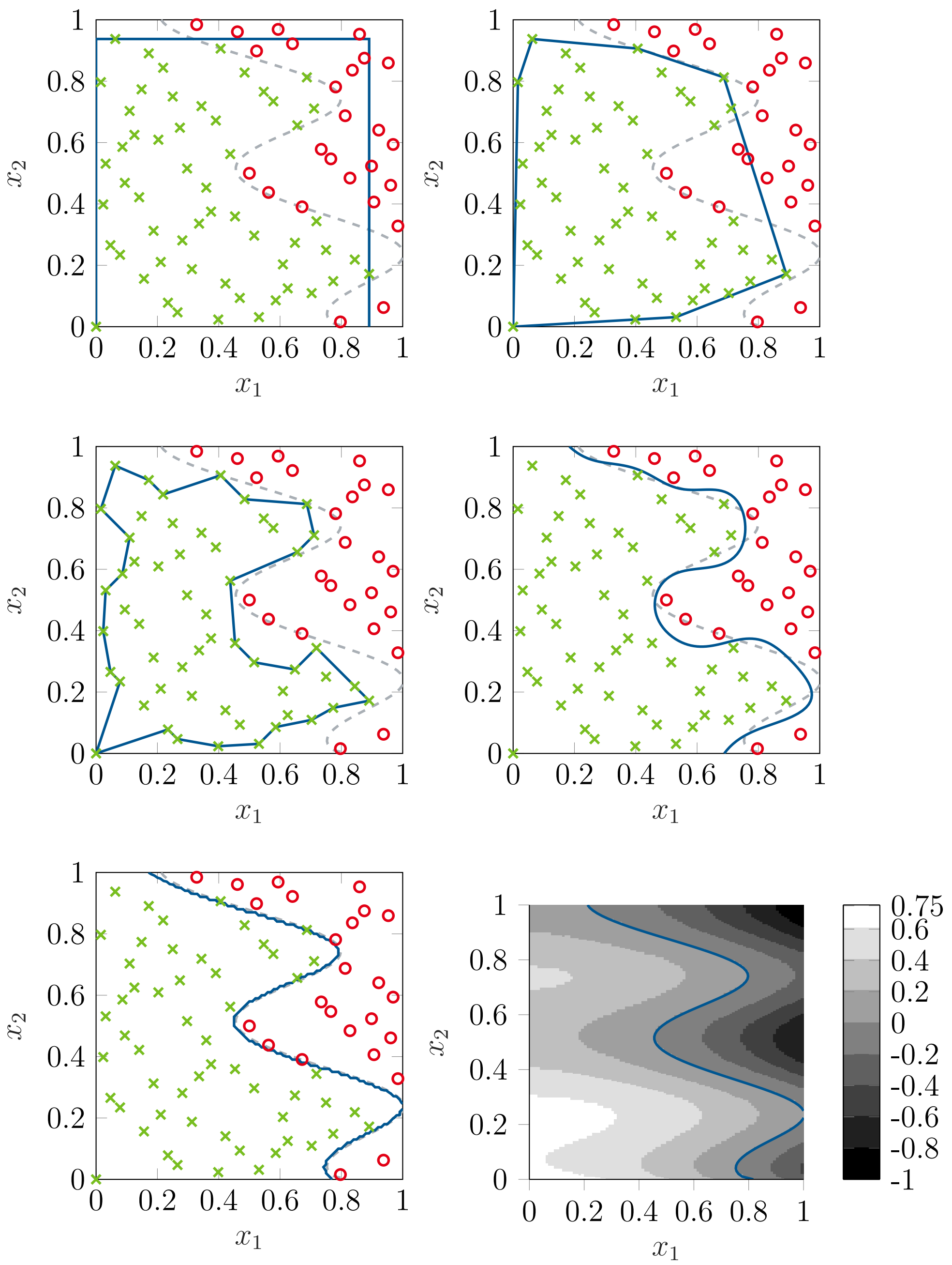

Figure 3.2: Comparison of different hulls and classifiers. Top left: hypercube, top right: convex hull, middle left: non-convex hull, middle right: two-class classifier, bottom left: regression model, bottom right: discriminative function used for the regression model. Green crosses are feasible input points, red circles denote infeasible ones. The true boundary of this artificial example is plotted as gray dashed line.

defined using the artificial discriminative function

$$g(x_1, x_2) = -\frac{2}{3}\left(\mathrm{e}^{x_1-1}\left(\frac{1}{2}\cos(4\pi x_2) + 0.5\right) + x_1^2 + \sqrt{x_2}\right) + 1. \tag{3.5}$$

This function is visualized in the bottom right plot. An input point $\boldsymbol{x} = (x_1, x_2)$ is feasible if $g(x_1, x_2) > 0$. First, a hypercube around the feasible points is drawn. As one can see in the upper left plot, this does not discriminate the feasible from the infeasible points well. In the upper right plot, a convex hull is used, which performs better than the hypercube but still does some miss-classification. In the following middle left plot an alpha-shape was used as nonconvex one-class classifier. It manages to separate feasible and infeasible points but draws a tight boundary around the feasible points. The middle right plot shows a two-class classifier, namely the Gaussian label regression used in the ODCM algorithm. This classifier, opposed to the other ones, not only uses the feasible but also the infeasible points for training. Finally the boundary predicted by a GP regression model is shown, which directly tries to model the discriminative function $g(x_1, x_2)$. This model is capable of separating feasible and infeasible points as well as the two-class classifier did, but only used feasible points as training data. In practice, the last property is an advantage, as trying to measure infeasible points requires additional measurement time and is a threat for the system under test.

3.2 Online Design of Experiments with Constraint Modeling

In this section, the ODCM method known from [23] is explained. As mentioned in the last section, ODCM is an online, non-screening boundary estimation method using a two-class classifier. As an additional benefit, not only an estimate of the stationary boundary, but also space-filling measurement points suitable for data-based modeling are obtained. Compared to classical DoE methods, ODCM reduces the number of infeasible samples during the measurement and gives a good estimate of the system's boundary.

The ODCM algorithm is presented as pseudocode in Algorithm 3.1. At the beginning a DoE plan is loaded. This plan usually features a space-filling distribution and could,

Algorithm 3.1 Pseudocode of the ODCM algorithm, adapted from [23].

```
Require: DoE plan of size $m_{\max}$,
  load DoE plan
  for $i = 1, \ldots, m_{\max}$ do
      read the $i$-th point from DoE plan
      asses point using the classifier
      if classifier output is above threshold then
          measure and label point
          update classifier
      else
          skip and label point
      end if
  end for
```

for example, be a Sobol set or a Latin hypercube. The plan does not need to be manually adapted to the expected feasible input space, as would be necessary during the traditional DoE procedure. Nonetheless, the points should be in a suitable order, e.g. with rising distance to a central point or operating point.

The planned points are read from the DoE plan one after another. Each point's feasibility is estimated by a classifier. If the predicted feasibility is above a predefined threshold, the point is sampled. If not, the point is skipped. The threshold is chosen such that a point is measured in case of doubt. Otherwise, the algorithm would falsely skip many points in the beginning, when it has seen little or no training data and its predictions are still unreliable. Thus, the threshold has to be lower than the prior mean of the GP classifier, which is usually at 0.5 when the labels 0 for infeasible and 1 for feasible points are used. As Figure 3.3 illustrates at a 1D example, the threshold has a medium influence on the algorithms performance. If it is chosen in a large range of $[0.1, 0.5]$, the input range classified as feasible varies between $[0, 0.49]$ and $[0, 0.66]$.[I] This means that the threshold does have an influence on the algorithms performance, but it is not extremely critical. In the experiments in Section 6.2.2 and 6.3.2 a threshold of 0.4 was used.

If the point was not skipped, it is measured subsequently. Thereby, the point's feasibility as well as its corresponding output values are determined. Afterwards, the classifier is retrained, incorporating the new information.

[I]Note that the input range on the right of the first point predicted as infeasible is never assumed to be feasible, because the ODCM algorithm also checks if there is a feasible straight path from the save point to a candidate. For simplicity reasons, this is omitted in Algorithm 3.1

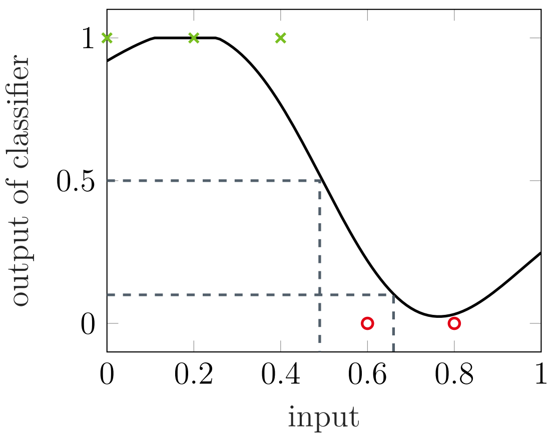

Figure 3.3: Influence of the threshold in a 1D example of ODCM. 5 equally spaced points in the range $[0, 0.8]$ were used as training data. Green crosses denote feasible samples, red circles infeasible ones. In this case a threshold varying between 0.1 and 0.5 results in an input range predicted to be feasible between $[0, 0.49]$ and $[0, 0.66]$, denoted by the dashed lines.

3.3 ODCM with improved boundary estimation

In this section, a new algorithm for ODCM with improved boundary estimation is presented. It is one contribution of this thesis and was not published before. A patent application is pending [21]. A similar approach was published in [27], which uses another classification algorithm and is targeted at an exploration of the feasible input space. The algorithm presented here is more focused on improving the boundary estimation from a previous ODCM run. The main idea is to place new measurement points near the boundary estimated by the classifier.

The improved boundary estimation algorithm is given as pseudocode in Algorithm 3.2. An illustration of one algorithmic step is shown in Figure 3.4. Before the algorithm starts, m_0 initial points have already been measured, for example during a previous ODCM run. These points are used to train a two-class classification model, for example a Gaussian process model. The model is subsequently used to classify a number of candidate points $\mathcal{X}_\mathrm{c}$, which are usually distributed spacefillingly in the input space. Thereby it is not only estimated whether the points are feasible or infeasible, but a continuous classifier output is predicted. Only those points whose classifier output is between a minimum and a maximum value are considered in the following steps. These points are close to the decision boundary of the classifier and thus not classified reliably. From these points, the point with maximum nearest

Algorithm 3.2 Pseudocode for the ODCM algorithm with improved boundary estimation.

```
Require: sample size $m_{\max}$, set of candidate points $\mathcal{X}_{\mathrm{c}} \subset \mathbb{X}$, minimum and maximum
        classifier output $\phi_{\min}$ and $\phi_{\max}$, $m_0 \geq 1$ previously sampled points $\mathcal{X}_{\mathrm{m}} \subset \mathbb{X}$
  train classifier
  for $i = m_0 + 1, \dots, m_{\max}$ do
      estimate classifier output $\phi(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{X}_{\mathrm{c}}$
      restrict to points within minimum and maximum classifier output:
          $\mathcal{X}_{\mathrm{r}} = \{\boldsymbol{x} \in \mathcal{X}_{\mathrm{c}} : \phi_{\min} < \phi(\boldsymbol{x}) < \phi_{\max}\}$
      choose point with maximum nearest neighbor distance to all previously sampled
          points: $\boldsymbol{x}_{\mathrm{opt}} = \mathrm{argmax}_{\boldsymbol{x} \in \mathcal{X}_{\mathrm{r}}} \left(\min_{\boldsymbol{x}_{\mathrm{m}} \in \mathcal{X}_{\mathrm{m}}} \|\boldsymbol{x} - \boldsymbol{x}_{\mathrm{m}}\|\right)$
      measure and label point $\boldsymbol{x}_{\mathrm{opt}}$ and add it to $\mathcal{X}_{\mathrm{m}}$
      optionally update classifier
  end for
```

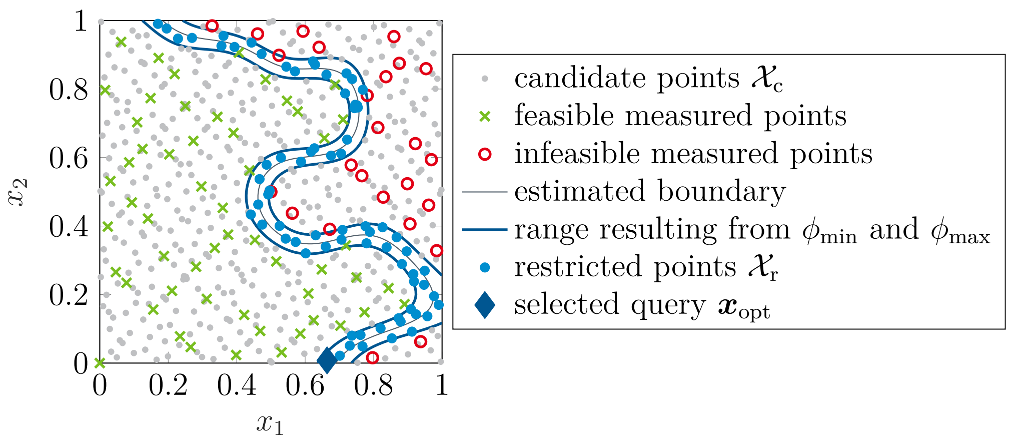

Figure 3.4: Schematic of the ODCM algorithm with improved boundary estimation using the example from Figure 3.2. Feasible and infeasible measured points form $\mathcal{X}_{\mathrm{m}}$ together.

neighbor distance to all previously measured points is chosen and measured at the system. Optionally, the classifier is updated with the newly obtained information and the algorithm continues until a predefined number of samples is reached or another stop criterion is fulfilled.

3.4 ODCM with discriminative model

As described above in Section 3.2, ODCM uses a two-class classifier to describe the boundary between feasible and infeasible points. Opposed to that, the SAL algorithm presented in Section 5.1 utilizes a so-called discriminative model, which is a health state regression of the system. As will be shown in the evaluation in Section 6.2.4, this discriminative model outperforms the two-class classifier of ODCM regarding false classifications and especially minimizes the number of infeasible samples during the measurement. The active learning approach, with a selection of the queries based on uncertainty sampling, did not lead to a better model compared to the plan-based approach of ODCM, though. On the other hand, active learning has some drawbacks in practical applications:

- The number of queries necessary to cover the complete feasible input space is determined automatically by SAL. This is an advantage in some cases, but it makes it impossible to choose a smaller number of queries and still cover the whole input space. The latter is desirable in some practical applications, for example if the available test bench time is very limited.

- It is not easily possible to adjust the placement of queries using expert knowledge. Even though not necessary in most cases, there might be special applications where a human expert wants to increase or decrease the queries' density in certain areas or mark parts of the input space as infeasible in advance.

- As discussed in Section 5.1.1, the training of the GP models' hyperparameters used in SAL is challenging. Especially the first steps, when little training samples are available, are critical. Wrongly estimated hyperparameters can lead to limit violations. Using a suitable predefined plan mitigates this problem, as the first queries can be placed in a most likely safe area. Furthermore, no extreme outliers can be queried in the beginning, if the plan is sorted properly.

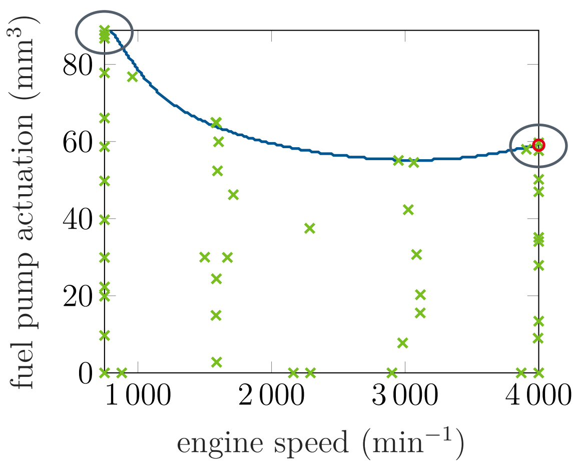

Figure 3.5: Example of a SAL measurement with too many samples relating to the system's complexity. Feasible and infeasible samples are denoted by green crosses and red circles, the estimated boundary is marked by the blue line. While the modeling error does not decrease significantly anymore after 25 samples, 50 points were measured. This leads to accumulations of samples at the boundary, highlighted by gray ellipses.

- When sampling more points than necessary for the found system complexity, an accumulation of multiple queries at the same position, usually at the boundary of the feasible input space, could be observed (compare Figure 3.5). This behavior occurs due to the used squared-exponential kernel. The posterior variance predicted with this kernel raises already before the last training sample at a boundary (see Figure 3.6). Other evaluated kernels like the Matern kernel or a polynomial kernel behave similar. If the feasible input space is already covered with many samples, the variance's maximum will thus always be found at the boundary, regardless of the points already measured there. This effect can be avoided by using a predefined plan.

Taking the pros and cons of SAL and ODCM into account, a combination of the two methods offers advantages. The plan-based query selection approach of ODCM can be combined with the discriminative model of SAL. Even though this might look like a step backwards compared to pure SAL, it could combine the advantages of both methods.

The implementation of this idea is rather straight-forward. The main ODCM Algorithm 3.1 stays the same, only the classifier is replaced by the one presented in Section 5.1. In detail some fine-tuning of the parameters is necessary, which will be

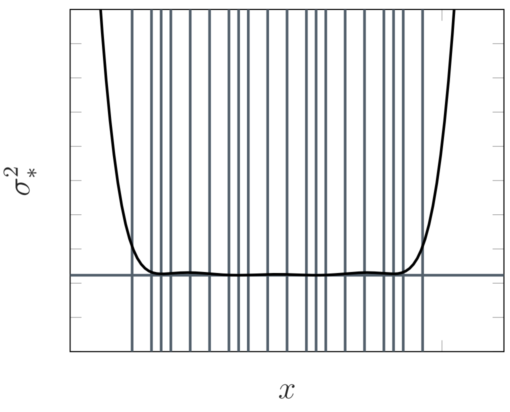

Figure 3.6: 1D example which shows how the posterior variance σ_*^2 increases already before the outmost training point if enough training data is available. The vertical lines denote the locations of the training data, the horizontal line is the minimum of the variance. If there are already samples at the boundary of the system, an uncertainty sampling based active learning algorithm would query the next point at the same redundant location.

described for the application example in Section 6.2.5.

4 Dynamic Supervised Measurement

After finishing the stationary boundary estimation, a dynamic experiment within this stationary boundary can be designed. An algorithm for this is presented in the following section. For the subsequent dynamic measurement, a supervision of critical output signals is required. Even if it is assumed that the system under test is also dynamically safe within its stationary boundary, the estimated boundary can still be partly wrong, leading to potentially dangerous limit violations without an additional supervision. Alternatively, the dynamic DoE might violate the estimated stationary boundary on purpose in order to cover a larger portion of the dynamically safe input space. Therefore, a dynamic supervised measurement algorithm is proposed in Section 4.2.

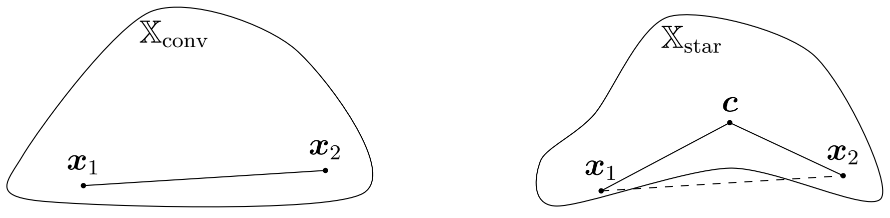

Figure 4.1: Example of a convex (left) and star-shaped (right) set. In a convex set the connection of each two points in the set is itself within the set, i.e. $\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{X}_{\text{conv}}, \alpha \in [0,1] : \alpha \boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2 \in \mathbb{X}_{\text{conv}}$. In a star-shaped set, this is no more true for all points, compare the dashed line in the right plot. Nonetheless at least one central point still fulfills the previous requirement, i.e. $\exists \boldsymbol{c} \in \mathbb{X}_{\text{star}} : \forall \boldsymbol{x} \in \mathbb{X}_{\text{star}}, \alpha \in [0,1] : \alpha \boldsymbol{c} + (1-\alpha)\boldsymbol{x} \in \mathbb{X}_{\text{star}}$.

4.1 Dynamic Design of Experiments in nonconvex stationary boundaries

As mentioned in Section 2.2, there are several common signals for dynamic excitation. This includes APRB, ramp, multisine, and chirp signals. These signals are usually generated within box-constraints on the inputs, as shown using the example of a chirp signal in the top left plot in Figure 4.2. In order to comply with the identified stationary boundary, the signals can be scaled into this boundary. Therefore, the algorithm presented in [19, 37, 62] can be used, as long as the boundary is described by a convex hull. The method can nonetheless be generalized to star-shaped (also called star-convex or radially convex) hulls, as presented in the internal report [36]. See Figure 4.1 for a definition of convex and star-shaped sets. In the following, this method will be described.

The procedure begins with a dynamic DoE in box-constraints, compare Figure 4.2 top left. Not all of the samples in this plot are stationary feasible, as they are outside the estimated boundary. All samples are now scaled towards a predefined central point. The result can be seen in the top right plot. If the feasible input space is convex, the central point can be anywhere within the feasible input space. Nonetheless, in order to get a uniform scaling, the point is usually chosen in the center of the estimated boundaries. If the feasible input space is only star-shaped, the central point has to be the star's center.

In the lower right plot in Figure 4.2 some vectors necessary for scaling a single sample are defined. All of them start at the chosen central point $\boldsymbol{c}$. Note that in this example, the central point is not in the middle of the feasible input space but a center of the star-shaped input space. A ray from the center $\boldsymbol{c}$ towards the data point to be scaled is drawn. Then, the intersections with the estimated boundary and the hypercube are calculated. This defines the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. Finally, the vector from $\boldsymbol{c}$ to the scaled point can be calculated as

$$\boldsymbol{r}' = \frac{||\boldsymbol{a}||}{||\boldsymbol{b}||}\boldsymbol{r}. \tag{4.1}$$

Depending on the boundary modeling algorithm in use, the intersection of the ray and the estimated boundary cannot be calculated analytically. This problem occurs, for example, if a GP classifier is used. To avoid it, the ray can be discretized into query points, whose feasibility is predicted by the classifier. The last feasible point on the ray is then used as intersection.

As all points are scaled, and not only those in the infeasible part of the input space, a accumulation of points near the boundary is avoided. Nonetheless, samples which were originally close to each others can be scaled to further away locations due to different scaling factors in each dimension. As the samples will be applied to the system with a fixed sampling time, this increases the signals' gradients and leads to peaks (compare the middle plots in Figure 4.4). Furthermore, oscillations in the gradient might occur. In order to reduce the peaks and keep the limits on the gradients, the scaled signal is filtered with a low pass filter. Thereby, potentially hazardous gradient peaks are avoided without major influence on the spacefilling property of the signal. The cutoff-frequency of the filter is a tuning parameter. If the cutoff frequency is too high, smoothing is weak. If is is too low, the signal is strongly deformed. The cutoff frequency can be easily tuned manually by inspecting gradient plots as shown in Figure 4.4.

The final scaled and filtered signal is shown in the lower left plot of Figure 4.2. The filtering can lead to minor violations of the feasible input space's boundary. Nonetheless these are usually small and can be accepted. Figure 4.3 gives a comparison of the signals in time domain, while Figure 4.4 shows their gradients.

4.2 Dynamic Supervised Measurement Algorithm

After the dynamic DoE has been created, the signals can be applied to the system. A supervision algorithm can be used to protect the system from limit violations. The basic idea of this algorithm is to switch on a controller in case of measured or predicted boundary violations. The algorithm was developed at Bosch Engineering and investigated in one Bachelor, three Master theses and one patent application [22]. While the first two theses [4, 16] were created before this PhD project was started, the following two [18, 55] were supervised by the author of this contribution. Their main results are presented in the following.

The first thesis [4] implemented the supervision algorithm for a high pressure fuel supply system with an automation based on the ETAS tool INCA-Flow and a linear

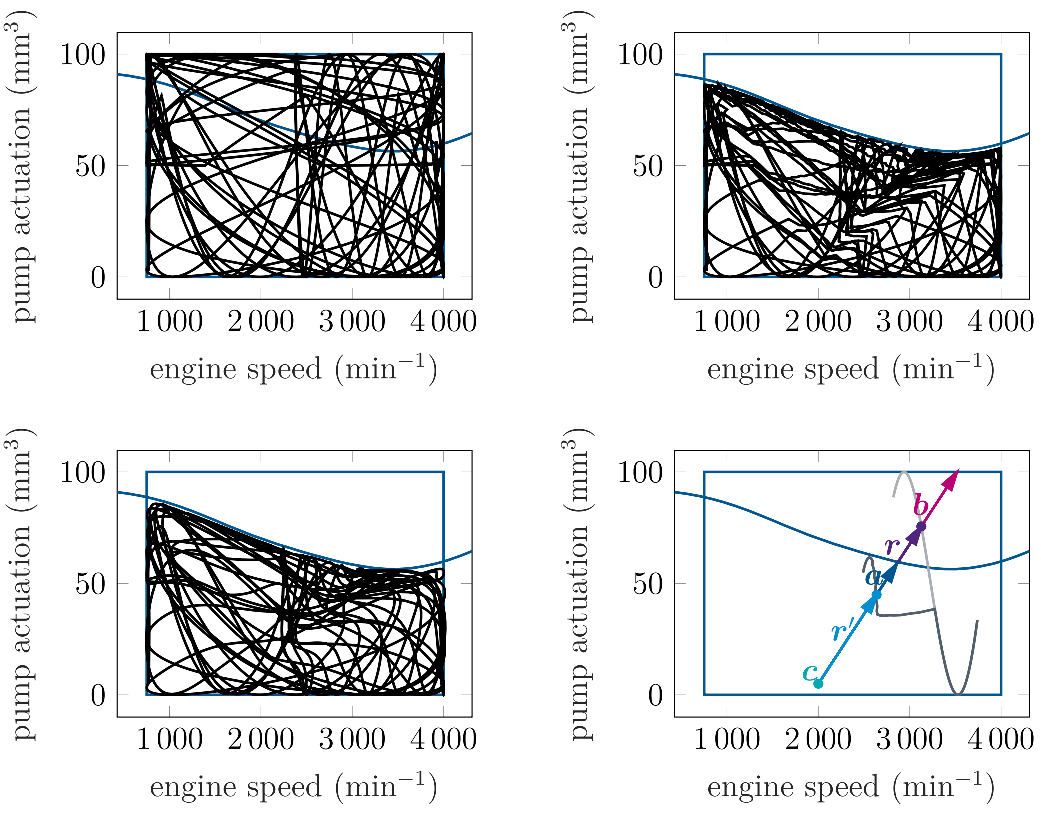

Figure 4.2: 2D example of the steps of dynamic DoE in nonconvex stationary boundaries applied to a high pressure fuel supply system. The area above the curvy blue line is infeasible, while the lower part is feasible. Top left: initial DoE in boxconstraints. Top right: DoE scaled within the boundary. Bottom left: scaled and filtered DoE to meet gradient constraints. Bottom right: detail to define the vectors used for scaling.

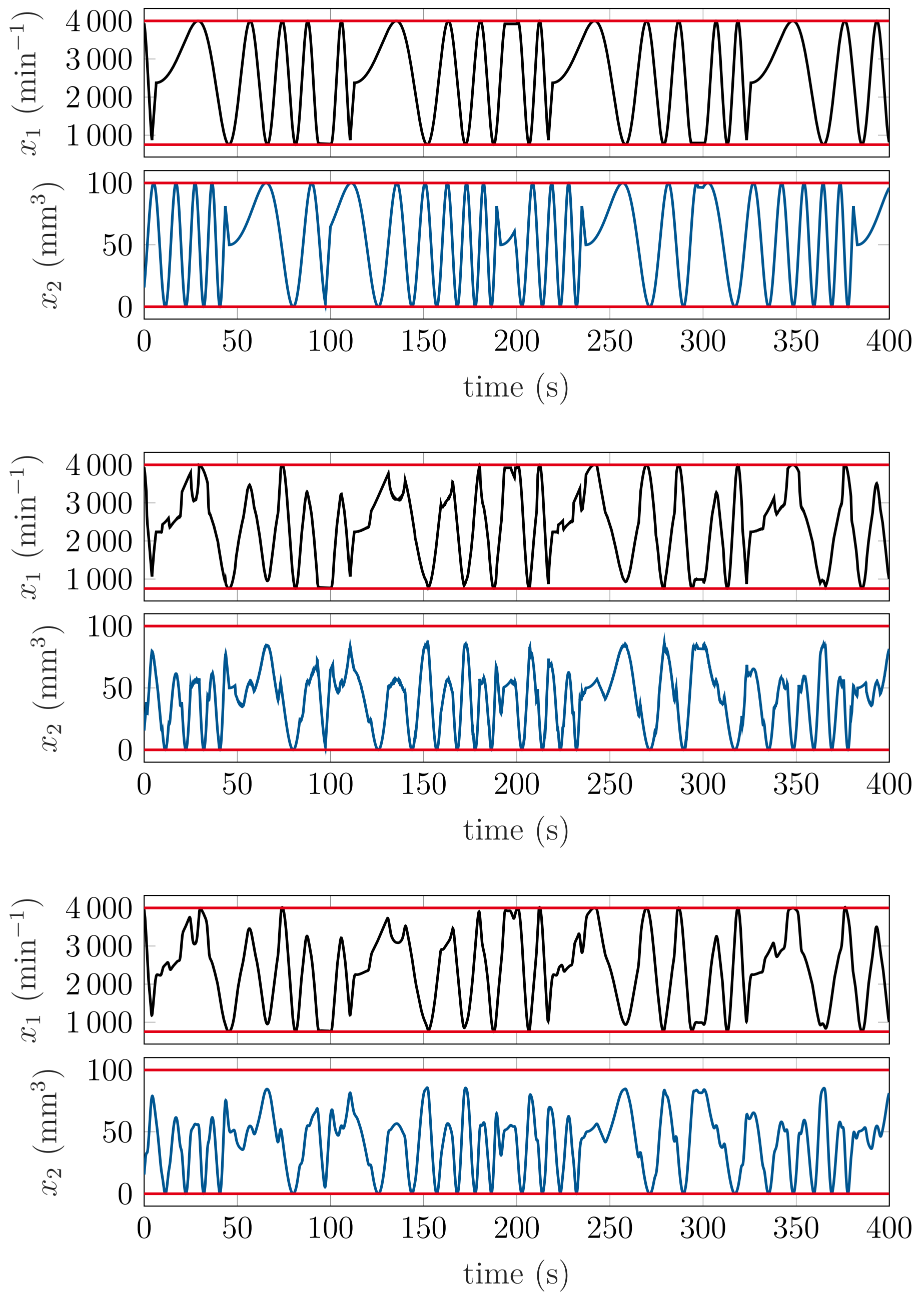

Figure 4.3: Signal amplitudes over time for the example in Figure 4.2. From top to bottom: original DoE, scaled signals, scaled and filtered signals. x_1 is the engine speed, while x_2 denotes the fuel pump actuation. The red lines describe the boxconstraints on the signals.

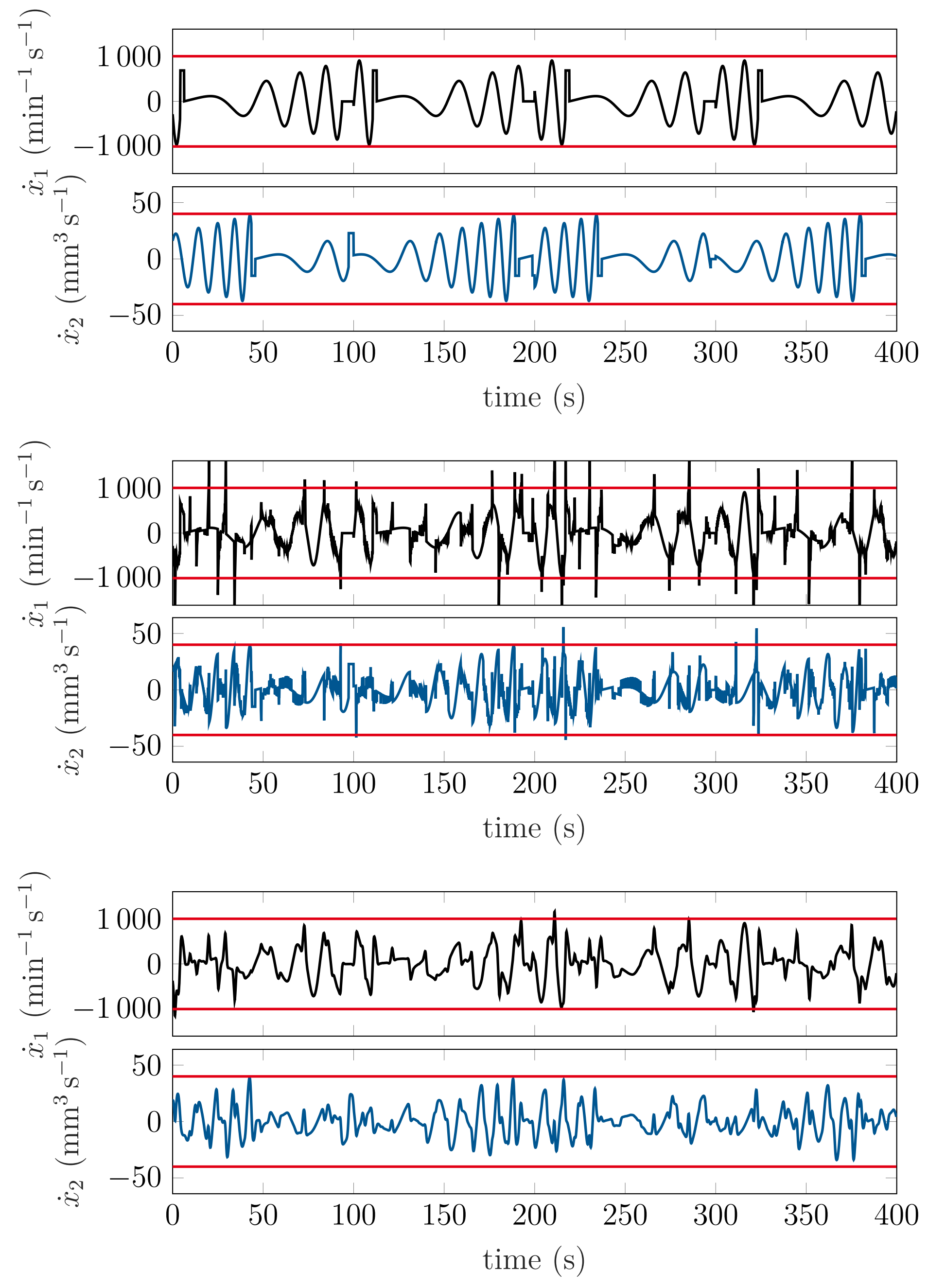

Figure 4.4: Signal gradients for the example of Figure 4.2 and 4.3. From top to bottom: original DoE, after scaling, and after filtering. $\dot{x}_1$ is the derivative of the engine speed, while $\dot{x}_2$ denotes the derivative of the fuel pump actuation. The red lines describe the boxconstraints on the signals' gradients.

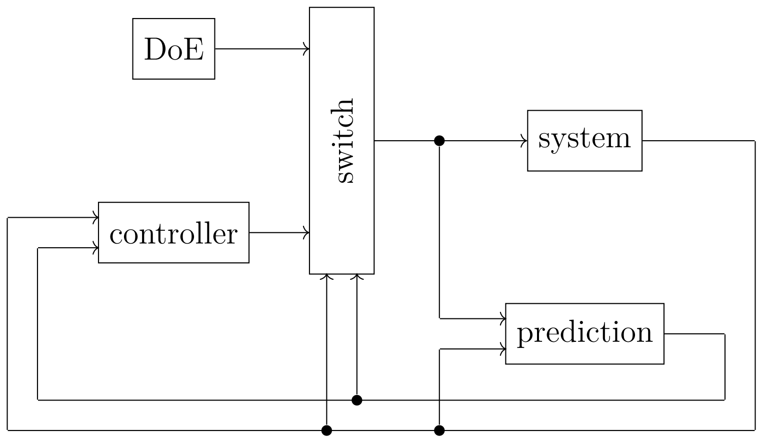

Figure 4.5: Basic structure of the supervision system.

extrapolation model. See Appendix A for a brief description of INCA-Flow and other software products used in this thesis. In [16] the method was adapted to another high pressure fuel supply system and the automation was rebuild using Matlab and INCA-MCE due to performance issues at small sampling times. Furthermore, an ARX model was used as prediction model in the algorithm. The next thesis [55] covered a simulation of a system with large time constants, implemented a gain-scheduling controller and evaluated methods for automated controller tuning. Based on this work, [18] applied the supervision algorithm to a charge air system subject to constant engine speed and implemented the algorithm in the online measurement automation (OMA) framework, compare also Section 6.1.

The supervision system consists of three main parts:

- the controller,
- the prediction model, and
- the switching logic.

The structure is presented in Figure 4.5. During normal operation, the DoE is directly applied to the system. Thus, the desired open loop measurement is possible. The system's outputs are supervised by the switching logic. Additionally, future outputs are predicted based on the past and future inputs applied to the system and past outputs. The predicted output is also considered in the switching logic. If a limit violation or a predicted future limit violation is detected, the control loop is closed and the controller is switched on. The controller can act on the current and predicted system output, depending on its design. This structure can be interpreted as a switching control, compare [30], or override control, see [7], where one controller is replaced by the DoE as feed-forward controller.

In the following subsections, these three parts are discussed in more detail.

4.2.1 Supervising Controller

The controller's task is to keep the system near its boundary, if the DoE would result in a limit violation. To achieve this goal, basically any controller structure could be used, depending on the system's behavior. Nonetheless, there are special requirements for the controller in this application. While a superb reference tracking is not necessary and neither is a perfect disturbance rejection, the controller should be easy to tune and robust. Overshoots should be avoided. As the controller is used during the measurement of the system, usually no sophisticated system model is available for tuning. This eliminates many complex control structures like model-based or optimal control, as they require a good model of the system.

Due to similar considerations, a PID controller was selected in [4]. This kind of controller is easy to tune and quite robust. In [55] it was extended to a PID gain scheduling controller to cope with stronger nonlinearities in the system. A concept for tuning the controller was also developed and tested in simulation. PID controllers are probably the most widely used controllers in industrial practice. Many different methods for tuning these controllers were published, see e.g. [2]. The concept in [55] features a frequency domain tuning method as described in [48]. In the following it is presented as one possibility to tune the supervising controller. Note that a suitable controller structure and tuning method always depend on the system to be controlled and are not generally applicable to any system.

In Figure 4.6 a scalar linear PID controller in serial and parallel structure is shown, compare [32]. The parallel structure is usually chosen for implementing a controller, while the serial structure is better suited for the frequency domain tuning method described in the following. Thus, it is necessary to transform the controller from the serial to the parallel structure. The parallel structure features three tuning parameters: The proportional gain $K_{\mathrm{P,p}}$, the integral gain $K_{\mathrm{I,p}}$ and the derivative gain $K_{\mathrm{D,p}}$. The derivative part of the serial structure is extended to be proper. Thus,

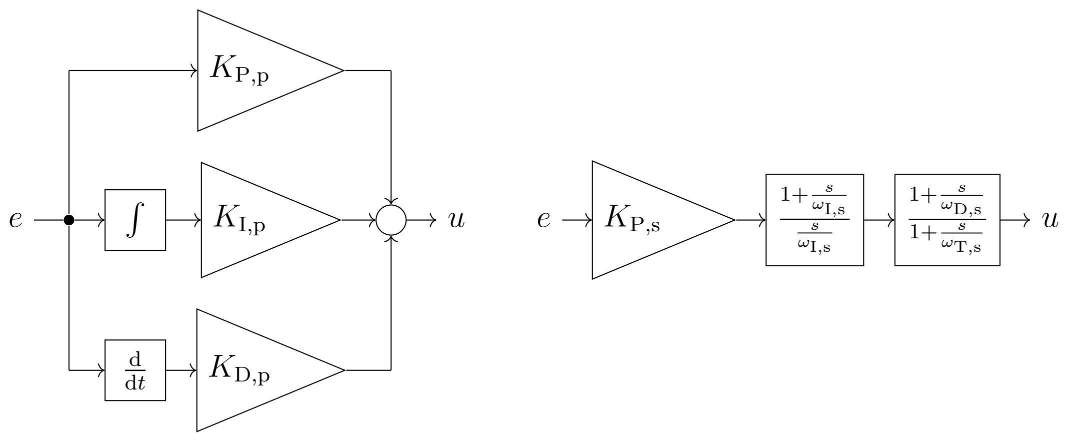
e
$\int$
$\frac{\mathrm{d}}{\mathrm{d}t}$
$K_{\mathrm{P,p}}$
$K_{\mathrm{I,p}}$
$K_{\mathrm{D,p}}$
u
e
$K_{\mathrm{P,s}}$
$\frac{1+\frac{s}{\omega_{\mathrm{I,s}}}}{\frac{s}{\omega_{\mathrm{I,s}}}}$
$\frac{1+\frac{s}{\omega_{\mathrm{D,s}}}}{1+\frac{s}{\omega_{\mathrm{T,s}}}}$
u

Figure 4.6: Ideal parallel (left) and proper serial (right) structure of a PID controller.

this representation features four parameters, which are the proportional gain $K_{\mathrm{P,s}}$ and the cutoff frequencies of the integral, differential, and proper differential parts $\omega_{\mathrm{I,s}}, \omega_{\mathrm{D,s}}$, and $\omega_{\mathrm{T,s}}$. If the parallel PID is extended similarly, its transfer function can be calculated as

$$G_{\mathrm{p}}(s) = K_{\mathrm{P,p}} + K_{\mathrm{D,p}} \frac{s}{1+\frac{s}{\omega_{\mathrm{T,p}}}} + K_{\mathrm{I,p}} \frac{1}{s}. \tag{4.2}$$

The serial structure's transfer function is

$$G_{\mathrm{s}}(s) = K_{\mathrm{P,s}} \cdot \frac{1+\frac{s}{\omega_{\mathrm{I,s}}}}{\frac{s}{\omega_{\mathrm{I,s}}}} \cdot \frac{1+\frac{s}{\omega_{\mathrm{D,s}}}}{1+\frac{s}{\omega_{\mathrm{T,s}}}}. \tag{4.3}$$

Both representations are equivalent and can be transformed into one another. The parameters of the parallel structure can be calculated from those of the serial structure as

$$K_{\mathrm{P,p}} = K_{\mathrm{P,s}} \left(1 + \frac{\omega_{\mathrm{I,s}}}{\omega_{\mathrm{D,s}}} - \frac{\omega_{\mathrm{I,s}}}{\omega_{\mathrm{T,s}}}\right), \tag{4.4}$$

$$K_{\mathrm{I,p}} = K_{\mathrm{P,s}} \omega_{\mathrm{I,s}}, \tag{4.5}$$

$$K_{\mathrm{D,p}} = K_{\mathrm{P,s}} \frac{\omega_{\mathrm{T,s}} - \omega_{\mathrm{I,s}} - \omega_{\mathrm{D,s}} \left(1 - \frac{\omega_{\mathrm{I,s}}}{\omega_{\mathrm{T,s}}}\right)}{\omega_{\mathrm{D,s}} \omega_{\mathrm{T,s}}}, \text{ and} \tag{4.6}$$

$$\omega_{\mathrm{T,p}} = \omega_{\mathrm{T,s}}. \tag{4.7}$$

As mentioned above, the serial structure is well-suited for analysis and design in the frequency domain. Figure 4.8 shows a bode plot of the PID controller's transfer

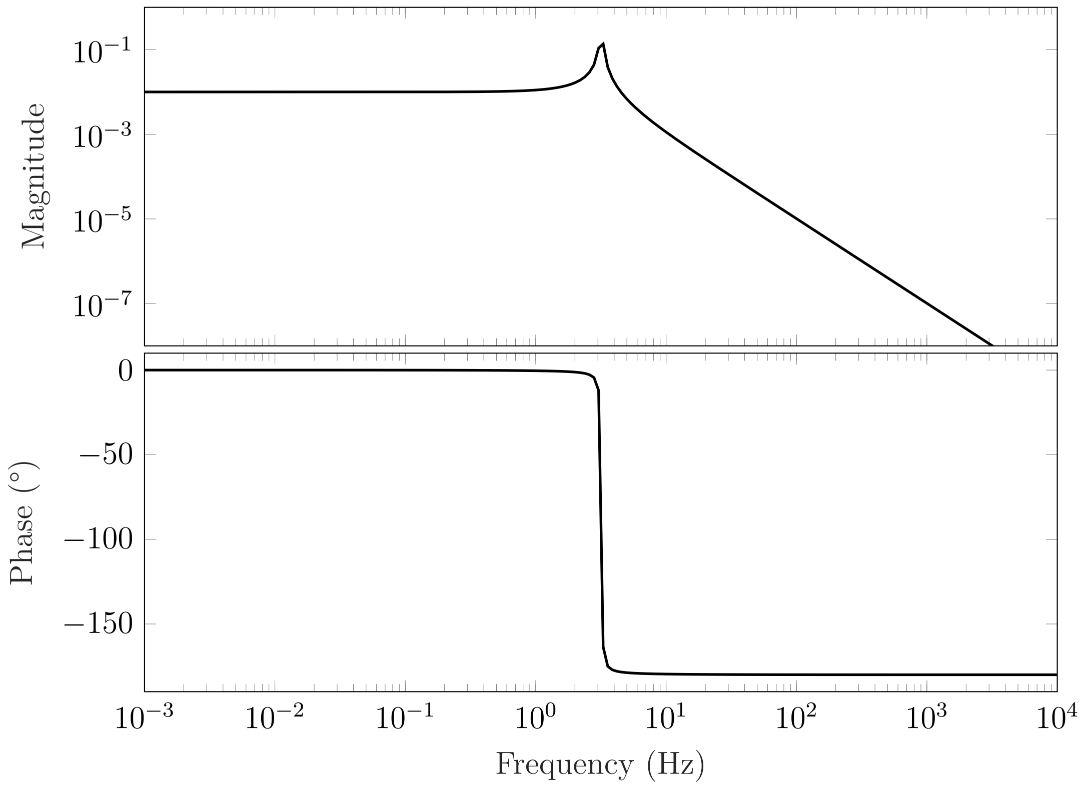

Figure 4.7: Bode plot of an example plant.

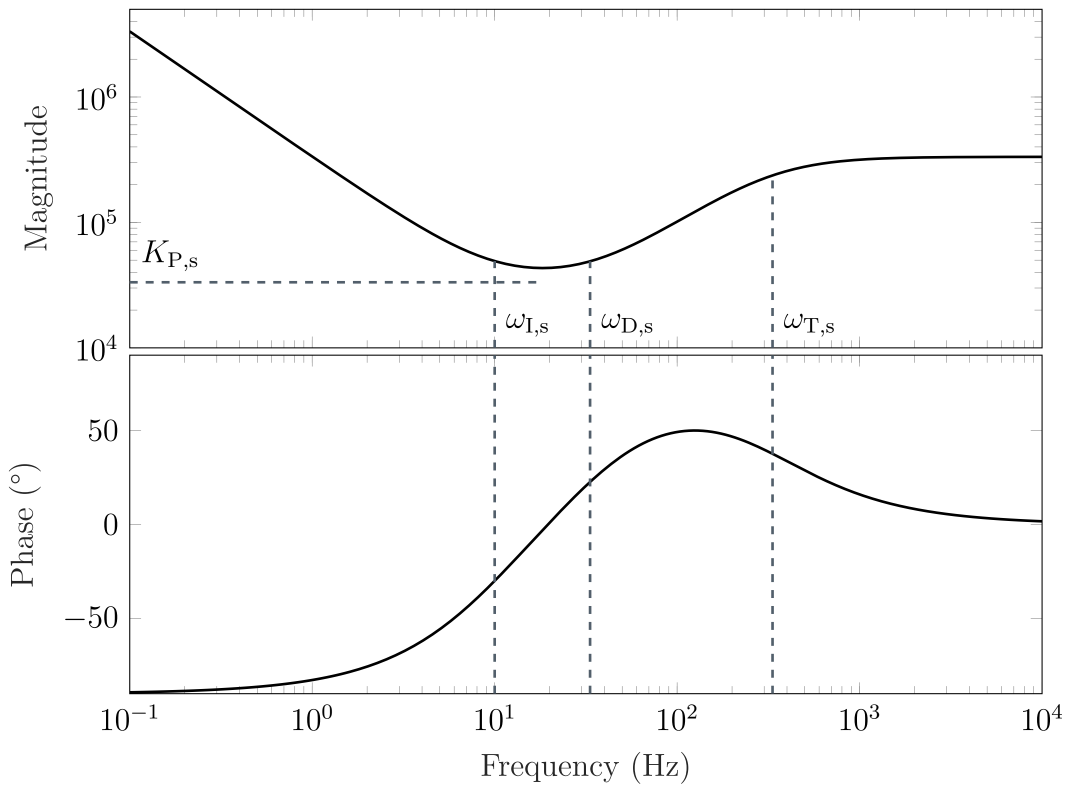

Figure 4.8: Bode plot of the PID controller designed for the plant in Figure 4.7.

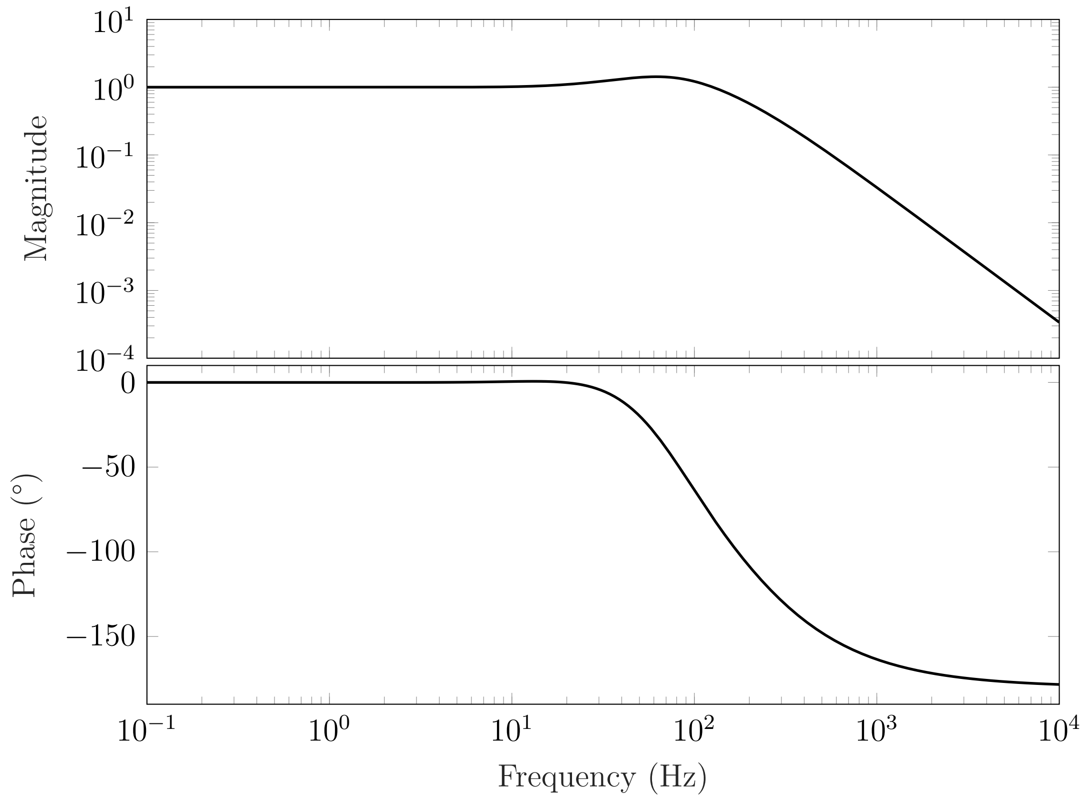

Figure 4.9: Bode plot of the closed-loop transfer function with the PID controller from Figure 4.8 and the plant from Figure 4.7.

function. In Figure 4.7 amplitude and phase of a linear example system based on [48] are given. It is a model of a weakly damped mass-spring system. The task of the controller tuning is to shape the closed-loop transfer function in a desired way. For this system, a constant amplitude of 1 (equivalent to $0\,\mathrm{dB}$) for frequencies significantly below the chosen controller bandwidth $\omega_\mathrm{c} = 2\pi \cdot 100\,\mathrm{Hz}$, i.e. $\omega \ll \omega_\mathrm{c}$, and an amplitude slope of $-2\,\mathrm{dec}^{-1}$ (i.e. $-40\,\mathrm{dB}\,\mathrm{dec}^{-1}$) for frequencies $\omega \gg \omega_\mathrm{c}$ is requested. The controller bandwidth was chosen in [48] such that it is higher than the mass-spring systems resonance frequency at $20\,\mathrm{Hz}$. As shown in Figure 4.9 this can be obtained with the controller presented in Figure 4.8.

To obtain the desired closed-loop behavior, the following heuristics were derived in [48]:

$$\omega_\mathrm{I,s} \approx \frac{\omega_\mathrm{c}}{10}, \tag{4.8}$$

$$\omega_\mathrm{D,s} \approx \frac{\omega_\mathrm{c}}{3}, \tag{4.9}$$

$$\omega_\mathrm{T,s} \approx \frac{10}{3}\omega_\mathrm{c}, \text{ and} \tag{4.10}$$

$$K_\mathrm{P,s} \approx \frac{1}{3A_\mathrm{pl}(\omega_\mathrm{c})} \tag{4.11}$$

with the plant's amplitude at the controller bandwidth $A_\mathrm{pl}(\omega_\mathrm{c})$. These heuristics are targeted at a double integrating (I_2) plant or a PT_2 system which behaves similar to an I_2 system around the controller bandwidth ω_c. This bandwidth is a tuning factor which determines up to which frequency the closed loop provides unity gain.

If a system is not linear and also cannot be linearized at one operating point with sufficient accuracy for the whole relevant operating range, a nonlinear controller design is necessary. A comparatively simple nonlinear approach is to change the controller parameters based on the operating point, which is called gain-scheduling. For this method, one or more scheduling variables have to be found based on expert knowledge or additional analysis methods. These scheduling variables should correlate well with changes in the system's dynamics. Several operating points of the scheduling variables are defined. For each operating point, the system is linearized and a linear controller is tuned. Finally, these controllers are merged by interpolating the controllers' parameters between the single designs based on the scheduling variables. See e.g. [2] for details. Reference [29] contains a literature survey on gain-scheduling analysis and design.

Algorithm 4.1 Procedure for tuning the supervising controller loosely based on [55].

```
determine input to be controlled and output to be supervised
define scheduling variables and operating points
create preliminary system model
for each operating point do
    choose controller bandwidth $\omega_c$
    determine plant's amplitude at the controller bandwidth $A_{pl}(\omega_c)$ in simulation
    calculate parameters of serial structure using (4.8) to (4.11)
    repeat
        simulate phase margin $\varphi_R$
        if $\varphi_R \leq 50°$ then
            manually tune cutoff frequencies $\omega_{P,s}, \omega_{I,s}$, and $\omega_{D,s}$
        end if
    until $\varphi_R > 50°$
    check controller's stability
    if controller is unstable then
        tune $\omega_c$ and recalculate controller parameters
    end if
    calculate parameters of parallel structure using (4.4) to (4.7)
end for
evaluate final controller in simulation
```

The overall procedure for designing the supervising controller is presented in Algorithm 4.1. The inputs and outputs used by the controller are chosen based on knowledge about the structure of the system. The supervised outputs should be those which may not exceed a certain range in order to guarantee the system's safety and integrity. The inputs should have a sufficient impact on the supervised outputs, so that the system is controllable using these inputs and outputs.

The phase margin φ_R is a measure for the stability of a closed linear control loop. It is defined as the difference between the phase lag of the closed loop at the unity gain crossover frequency and $-180°$. The higher the phase margin, the more robust the stability of the closed loop is. By requiring a phase margin of 50° Algorithm 4.1 accounts for disturbances and model mismatch. If no sufficient phase margin is achieved in using the rules of thumb above, manual re-tuning is required. Further advice for suitable manual tuning can be found in [48].

As stated in the algorithm, a preliminary system model is required. This model can e.g. be a dynamic data-based model, which was tuned using measurement data from a limited region of the input space, which is known to be safe by expert knowledge. The model is used to determine $A_{\mathrm{pl}}(\omega_{\mathrm{c}})$, to calculate the phase margin φ_{R}, and to evaluate the final controller. Of course, using a preliminary model can cause problems. The model might not represent the system's behavior well enough and a controller tuned using the model might thus perform poor on the real system. Using a conservative DoE which stays well clear of the system's boundaries to gather measurement data for the preliminary model makes the problem even worse: The controller is used to prevent limit violations but the model used for tuning did not see any data from that operating range. Finally, the placement of operating points for the necessary linearization of the model and the gain scheduling can be critical. If they are chosen far away from the system's boundary, the accuracy of the linearized model might be even worse near the boundary where the controller is required to perform well.

These pitfalls have to be kept in mind when designing the supervising controller. The controller should be tested carefully during the first runs on the real system before using it to secure critical DoEs. The controller tuning offers many interesting research questions for the future. For example, it would be beneficial to be able to place operating points near the boundary automatically. Another option is to use even simpler tuning methods which do not require a system model at all. Examples for this category are the well known Ziegler-Nichols method [67] or the tuning method by Chien, Hrones, and Reswich [9].

In the literature, a stability analysis of the closed-loop is a common step after designing a controller, see e.g. [48]. Usually this analysis is based on a model of the system. As mentioned above, this model is preliminary in case of the supervising controller and additional effects like the linearization may further decrease its performance. Hence, a proof of stability based on the model has very limited evidentiary value and therefore an in-depth discussion is omitted here.

4.2.2 Supervising Model

The performance of the supervising controller can be improved by introducing a prediction model for the supervised output. The model allows the controller to act before a limit violation actually occurs. This is especially beneficial if a system has large time constants or a dead time. In such cases, limit violations are often hard to predict in an early stage by only defining a safety margin slightly below the physical

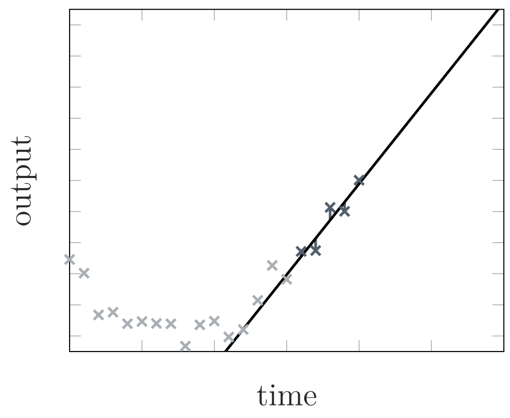

Figure 4.10: Concept of linear extrapolation. The crosses describe measured noisy output samples. The five dark gray ones ($n_{\text{past}} = 5$) were used to estimate the black line, which represents the prediction.

limit. If limit violations can be predicted early enough, the controller can be tuned less aggressive and still drive the system to the setpoint without large overshoot.

A suitable model should comply with several requirements: First, it should be sufficiently precise to predict a limit violation early enough. Nonetheless, the requirements regarding the model's precision are not very high. Depending on the system's dynamics, a prediction horizon of a few time steps can be sufficient. Furthermore the model needs to be precise only near the system's limit.

Second, the modeling process should be fast and simple. The supervising model is part of the measurement process trying to obtain measurement data for model training. Therefore, a causality dilemma results, if a large amount of training data would be needed to train the supervising model. Thus, simple data-based or physical models are suitable. The former could be trained with data collected in an unsupervised measurement run with known to be safe input limits, similar to the model used for controller tuning. These limits could, for example, result from an earlier stationary identification using ODCM or stationary SAL, see Chapter 3. The latter are created using expert knowledge. Nonetheless such knowledge is often unavailable or insufficient in cases, where a data-based modeling procedure was chosen.

Finally, the evaluation of the model needs to be possible in real time. This is another constraint on the complexity of the model.

Based on these requirements, two modeling procedures are evaluated in this thesis: A very simple extrapolation approach and a linear ARX model. The concept of the linear extrapolation approach is shown in Figure 4.10. During the measurement procedure, a linear model is estimated in each time step based on the last n_{past} measured outputs. Therefore, a least squares regression is used. The estimated straight line is then used to predict the output in the future. As long as the prediction horizon is not too large and/or the system's current behavior is close to linear, a sufficient accuracy can be achieved. It has to be emphasized that the system's inputs are not used in the prediction. Thus, the prediction horizon has to be so small that the inputs have only minor effect on the system's output, due to comparatively large time constants, or the input may not change considerably. Nonetheless, if these conditions are met, this prediction method is quite robust and can deliver good results, as will be shown in Section 6.3.3.

The second approach is to use an ARX model. This kind of model was introduced in Section 2.4.2. It is also a linear model, but does include the input signals and can represent dynamics of the system. Thereby it can deliver more accurate predictions and can also be used for systems with large time constants. A drawback is that it has more parameters than the linear extrapolation approach and thus needs more training data. It can not be estimated during the measurement procedure, but needs data collected beforehand. Even though methods like recursive least-squares or adaptive filters enable online-estimation of ARX models, they do not provide a sufficient model quality during the first measurement steps which disables the supervision in the early measurement. Furthermore, they would render the supervising algorithm even more complex, which is why these methods were not considered in this work. Initial data could, for example, be measured in a safe region of the input space which was determined using ODCM or SAL. In this case the system needs to be sufficiently linear so that the model can extrapolate and the prediction close to the boundary is still accurate enough.

4.2.3 Switching Logic

As mentioned earlier, the supervising controller is switched on and off regarding the current system state. In normal operation the system is measured in open loop, but as soon as critical conditions occur, the controller is switched on and the control loop is closed. To achieve a good performance of the supervision algorithm, several aspects have to be considered when designing the switching logic. This ensures to reliably

variable	definition
y_{hardmax}	limit above which emergency shutdown is initiated
y_{max}	physically allowed maximum
y_{set}	setpoint for controller
$y_{\text{lim,on}}$	limit for measured output above which the controller is switched on
$y_{\text{lim,off}}$	limit for measured output below which the controller is switched off
$y_{*,\text{lim,on}}$	limit for predicted output above which the controller is switched on
$y_{*,\text{lim,off}}$	limit for predicted output below which the controller is switched off

Table 4.1: Definition of threshold for the supervised output, used in the switching logic.

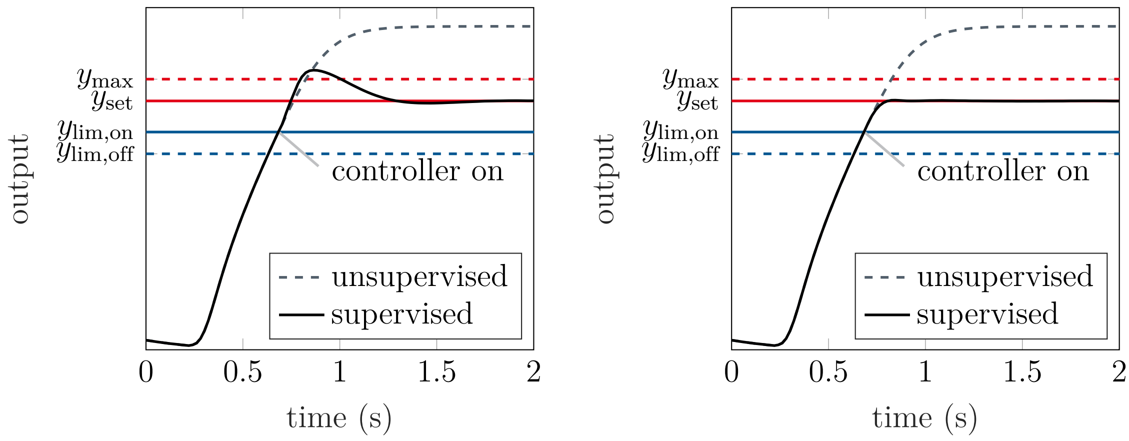

Figure 4.11: Simulation of a PID controller for supervision. On the left the system output is used as manipulated variable, whereas on the right the predicted output is used. The controller is switched on as soon as the output exceeds $y_{\text{lim,on}}$.

keep the system within its limits, as well as a minimum disturbance of the original DoE.

First, a number of thresholds for the supervised output is defined in Table 4.1. The order of these thresholds may vary, but often $y_{\text{lim,off}} \leq y_{*,\text{lim,off}} < y_{\text{set}} \leq y_{\text{lim,on}} \leq y_{*,\text{lim,on}} \leq y_{\text{max}} \leq y_{\text{hardmax}}$ was used in the experiments presented in Section 6.2.7 and 6.3.3. Using these thresholds, different switching criteria can be developed.

In the simplest case, the controller is switched on as soon as the supervised output y exceeds $y_{\text{lim,on}}$ and switched off after it has fallen below $y_{\text{lim,off}}$ again. To prevent frequent toggling, $y_{\text{lim,off}}$ is chosen smaller than $y_{\text{lim,on}}$ which implements a hysteresis. Additionally, a minimum active time can be defined to further reduce toggling in case of a fast oscillating supervised output. If the controller is on, it controls the output to the setpoint y_{set}. To account for a possible overshoot, the setpoint is chosen smaller than the physical maximum of the system y_{max}.

As shown for an artificial example in the left plot in Figure 4.11, this simple strategy can lead to large overshoots when activating the controller. A main issue is that the controller experiences a negative control error when being activated, which makes it even increase the controller output compared to the DoE. This problem can be slightly reduced by applying the minimum of controller output and DoE to the system. The latter is also necessary to allow the system's output to drop below the threshold again, as the controller would otherwise keep the system at its setpoint for an infinite time (neglecting disturbances). In [4] the predicted output instead of the measured output is used for feedback. Thereby, the controller not only tries to limit the current output to y_{set}, but also its values predicted a few steps ahead. The maximum of predicted and measured system output is chosen as controlled variable there. Using this strategy, a controller with minimum overshoot can be designed for the example system; compare the right plot in Figure 4.11.

Another option to prevent overshoot is to use the predicted output to switch the controller on and off. This gives the controller more time to anticipate a limit violation, so that it can be tuned less aggressive. Two additional thresholds $y_{*,\mathrm{lim,on}}$ and $y_{*,\mathrm{lim,off}}$ for the predicted output are therefore introduced. They can be chosen closer to the maximum y_{max} than the limits for the current output, as a predicted violation can still be anticipated by the controller. The limits for the current output can also be tightened, as they are only the second line of defense in this scenario. The controller is triggered by either the predicted or the current output, i.e. if $y > y_{\mathrm{lim,on}} \vee y_* > y_{*,\mathrm{lim,on}}$. It is switched off again, if both values have fallen below their limits, i.e. as soon as $y < y_{\mathrm{lim,off}} \wedge y_* < y_{*,\mathrm{lim,off}}$ is fulfilled. As shown in Figure 6.22 in the application chapter, this approach can also successfully limit the output without overshoot. As in the previous case, a minimum controller operation time can be requested to prevent frequent toggling.

If a controller with integrating part is used, the integrator has to be initialized properly each time the controller is switched on. This can be done using bumpless transfer, as presented e.g. in [1]. Thereby, steps in the controller's output are avoided. Furthermore, an anti-windup mechanism is necessary. It prevents the integrator from infinitely accumulating the control error if the controller has no influence on the plant because its output is larger than the DoE values.

As an additional safety measure, a hard limit for the supervised output y_{hardmax} can be defined. If this limit is exceeded, an emergency shutdown is initiated, the system is transferred in a safe state and the measurement procedure is aborted. Depending on the system properties, the limit and the shutdown strategy vary. In some systems a slight violation of the physical limits y_{max} is permitted for a short period of time. For such systems y_{hardmax} can be chosen larger than y_{max} to ensure that measurements are taken as close to the boundary as possible without unplanned shutdowns. Other systems are more critical and require y_{hardmax} to be smaller or equal to y_{max}.

5 Safe Active Learning

In this chapter, methods for online adaptive DoEs are presented. These methods optimize the information obtained with a measurement point or trajectory and at the same time avoid limit violations. Thereby they address the two most important goals of experimental design. While an improved information content can lead to better models and/or less required measurement samples, an automatic identification of the system's boundary in the input space reduces the effort for the design of experiments.

The first section deals with safe active learning (SAL) for stationary measurements. The second part of this chapter presents a variant of SAL for controller tuning and a safe Bayesian optimization (SBO) approach. The last section covers SAL for dynamic measurements.

5.1 Stationary Safe Active Learning

The stationary SAL approach considered in this thesis was first published in [51]. Therein, a theoretical analysis together with simulation experiments was presented. In this PhD project it was further developed and applied to two different real-world applications. The first application is the data-based modeling of a high pressure fuel supply system of a gasoline system. The results of this application were published in [45, 47] and are presented in Section 6.2.4. The second application uses SAL to optimize the parameters of a PI-controller for the same system. In [46], SAL is compared with a safe Bayesian optimization approach, which is also presented in Section 6.2.6.

According to [51], SAL pursues three main goals:

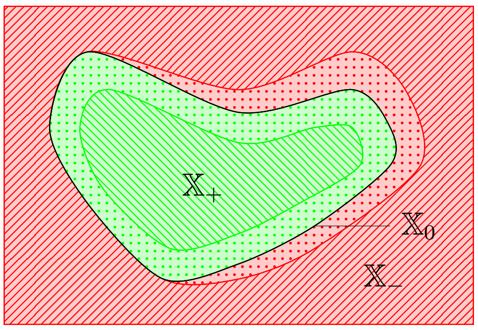

Figure 5.1: Partition of the input space $\mathbb{X}$ into a safe explorable area $\mathbb{X}_+$ and an unsafe region $\mathbb{X}_-$ separated by the unknown decision boundary $\mathbb{X}_0$. In the dotted areas, a continuous discriminative function value can be measured, whereas in the hatched areas only binary class labels are available. The figure is taken from [51].

- learn a model of the system from data as informative as possible,
- use only a limited budget of measurement points, and
- avoid critical input points during the measurement process.

The selection of informative queries using an active learning approach was already presented in Section 2.5. Schreiter et al. [51] use a Gaussian process based differential entropy selection scheme for this task. As presented earlier in Section 2.5, this selection scheme is a special case of uncertainty sampling. It aims at selecting the query where the model is least certain which output value to predict. In case of GP models, this simplifies to a maximization of the predicted variance of the model, compare (2.31).

To ensure a safe exploration of the input space, this selection scheme is combined with a discriminative model which distinguishes feasible from infeasible regions. This second model learns from continuous discriminative function values calculated using measurements at each queried input point. The discriminative function value provides information about the grade of feasibility of a sample. Its range can roughly be mapped to $(-1, 1)$ by a suitable definition, where positive values are feasible and negative values are infeasible. Consequently, the decision boundary is at 0. Metaphorically speaking, the discriminative function represents the health of the system. If, for example, the system may not exceed a given maximum temperature, a low temperature is equivalent to a discriminative function value close to 1, whereas a temperature nearly at the maximum results in a discriminative function value slightly above 0. If the temperature exceeds its limit, the discriminative function value approaches (or even exceeds) -1.

In the original publication, the discriminative model is in fact a problem-specific GP classifier. It is a hybrid model, which can use binary class labels as well as a continuous discriminative function values as training data. The assumption behind this approach is that the input space consists of 4 areas, compare Figure 5.1. First of all, the input space $\mathbb{X}$ is divided in a feasible subspace $\mathbb{X}_+$ (green) and an infeasible subspace $\mathbb{X}_-$ (red). Each of these two subspaces is further divided based on the information which is available when measuring there. In the dotted areas, a continuous discriminative function value can be measured. In the hatched areas, only the labels "feasible" or "infeasible" are obtained.

The division of the input space according to Figure 5.1 can be motivated at the temperature example introduced above: some systems can tolerate a limit violation, e.g. of a temperature, to a minor degree and for limited time. Thus, the stationary temperature can be measured in these cases, but not if the limit is violated considerably, because this would damage the system. An example for a system which partly only delivers labels in the feasible input space is enrichment for combustion engine component protection, measured by the lambda sensor. The amount of enrichment can be measured at a lambda value between approximately $\lambda = 0.7$ (depending on the sensor) and $\lambda = 1$. Values below $\lambda = 0.68$ are infeasible, because the engine could stall. If the sensor measures $\lambda = 1$, the system is feasible, but no grade of feasibility can be determined.

As stated before, two GP models are learned in this input space. The first one is a regression of the noisy system output $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon,\ \varepsilon \sim \mathcal{N}(0, \sigma^2)$. It is called *regression model* in the following and variables corresponding to this model are indexed with f. The second model is a hybrid model which learns from class labels $c(\boldsymbol{x}) \in \{-1, +1\}$ and noisy discriminative function values $h(\boldsymbol{x}) = g(\boldsymbol{x}) + \zeta \in (-1, 1),\ \zeta \sim \mathcal{N}(0, \tau^2)$, depending of the location of the input point $\boldsymbol{x}$ in $\mathbb{X}$. This model is called *discriminative model* and associated with the index g.

At most systems, the discriminative function values h necessary for training, i.e. the health states, cannot be measured directly as a scalar value in the range $(-1, 1)$. Instead, they are calculated from one or more measured *supervised outputs* $\boldsymbol{z}$ using a predefined scalar function $\tilde{h}(\boldsymbol{z})$. The system output y used for training the regression model can be a part of $\boldsymbol{z}$ but does not need to be. The function $\tilde{h}(\boldsymbol{z})$ is called *risk function* in the following. The influence of the risk function on SAL's performance and possible choices are covered in more detail in Section 5.1.3.

Both models use a squared exponential kernel and zero mean centered Gaussian priors. In case of the regression model, the posterior can be calculated analytically. This is not possible in case of the discriminative model, though. Here, a numerical Laplace approximation is necessary, due to the mixed kind of training data, compare Section 2.4.3.

The queries are selected one after another by solving a constrained optimization problem. The objective is the already introduced differential entropy criterion (2.28) in the variance form (2.31). As constraint, a minimum probability of feasibility predicted by the discriminative model is requested. Thus, it is demanded that

$$\Pr(g_* \geq 0|\boldsymbol{x}_*, \boldsymbol{c}, \boldsymbol{h}, \boldsymbol{X}) \geq p. \tag{5.1}$$

Thereby g_* is the predicted discriminative function value at the test point $\boldsymbol{x}_*$, $\boldsymbol{c} = [c_1, \dots, c_{m_c}]$ as well as $\boldsymbol{h} = [h_1, \dots, h_{m_h}]$ are the m_c measured class labels and m_h measured discriminative function values used for training at the $m = m_c + m_h$ locations stored in the matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_m]$, and p is the required probability of safety, i.e. the probability of the query being in $\mathbb{X}_+$. As stated in [51], (5.1) can be simplified using the confidence parameter $\nu = \Phi^{-1}(p)$, where Φ^{-1} is the inverse cumulative distributive function of the standard normal distribution. Typical values used in the experiments in Chapter 6 are $p = 99\,\%$ corresponding to $\nu = 2.33$ or $p = 50\,\%$ equivalent to $\nu = 0$. This results in the selection scheme

$$\boldsymbol{x}_{i+1} = \underset{\boldsymbol{x}_* \in \mathbb{X}}{\operatorname{argmax}}\, \sigma^2_{f*}(\boldsymbol{x}_*) \tag{5.2a}$$

$$s.t. \quad \mu_{g*}(\boldsymbol{x}_*) - \nu\sigma_{g*}(\boldsymbol{x}_*) \geq 0. \tag{5.2b}$$

The constraint of the optimization can be interpreted as follows: in order to allow a query $\boldsymbol{x}_*$ whose predicted mean $\mu_{g*}(\boldsymbol{x}_*)$ is only slightly above 0, the predicted standard deviation $\sigma_{g*}(\boldsymbol{x}_*)$ needs to be small, i.e. the prediction has to be reliable. If the predicted mean is higher, a larger uncertainty in form of a higher predicted standard deviation is tolerated. The larger the confidence parameter ν (or the required probability of safety p) is chosen, the more strict and thus careful the constraint is,

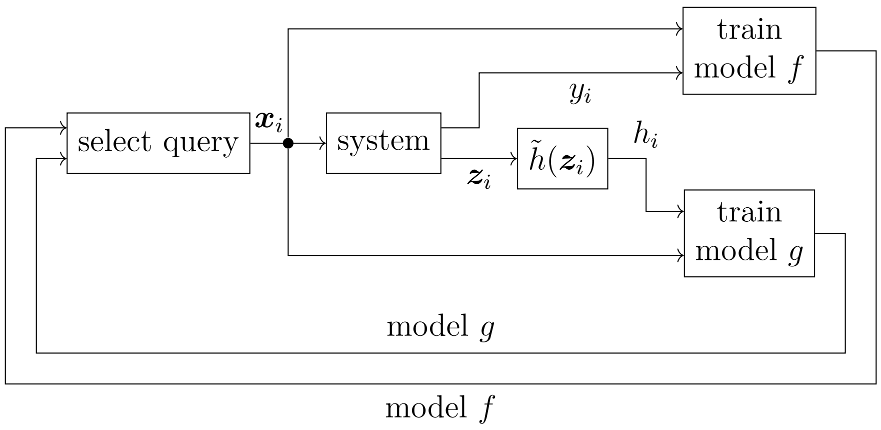

Figure 5.2: Block diagram of the SAL algorithm. Details on the *select query*-block are shown in Figure 5.3.

as the standard deviation is weighted higher.

Pseudocode for the implementation of SAL is presented in Algorithm 5.1. Therein $\mathcal{D}_i$ contains all input and output data at step i, m_0 is the number of initial samples, $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_g$ are the hyperparameters of the regression- and the discriminative model, $m_{\max}$ is the desired number of samples in total, p the requested safety of the exploration, and $\boldsymbol{x}_i$ the input vector at step i resulting in the scalar output for the regression model y_i and the vector of outputs for the discriminative model $\boldsymbol{z}_i$. Variable h is a discriminative function value which can be calculated from the measured outputs using the predefined risk function $\tilde{h}(\boldsymbol{z})$. Furthermore, the algorithm is visualized as block diagram in Figure 5.2. Here, σ_{f*} describes the predicted standard deviation of the regression model f, while μ_{g*} and σ_{g*} denote the predicted mean and standard deviation of the discriminative model g used in the selection of the next query.

In the Matlab implementation used for the evaluation of stationary SAL in Section 6.2.4, the optimization of the next query is done by a convex optimization algorithm with multi-start. In addition to the feasibility of the query, also the path from a global safe point (GSP) to the query is assessed for safety. After each measurement, the system is returned to the GSP. The system's inputs are varied slowly to prevent large overshoot and give the automation enough time to react in case of limit violations. If no feasible query is found by the optimization, a random query near the GSP is sampled, which is assumed to be safe. Optimizing the next query, learning the hyperparameters, and training the models needs approximately 2.2 s on average for the system considered in Section 6.2.4.

Algorithm 5.1 Pseudocode for the SAL algorithm. This code was already published in [47].

```
Require: initial measurement data $\mathcal{D}_{m_0}$ containing $m_0 \geq 1$ samples,
      initial hyperparameters $\boldsymbol{\theta}_f$, and $\boldsymbol{\theta}_g$, desired sample size $m_{\max}$,
      desired safety $p$
  train models $f$ and $g$ using $\mathcal{D}_{m_0}$
  for $i = m_0 + 1, \dots, m_{\max}$ do
      select $\boldsymbol{x}_i$ using (5.2)
      measure $y_i$ and $\boldsymbol{z}_i$ and add them to $\mathcal{D}_{i-1}$ to get $\mathcal{D}_i$
      calculate $h_i = \tilde{h}(\boldsymbol{z}_i)$ and add it to $\mathcal{D}_i$
      optimize hyperparameters $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_g$ using $\mathcal{D}_i$ if applicable
          (compare Section 5.1.1)
      train models $f$ and $g$ using $\mathcal{D}_i$
  end for
```

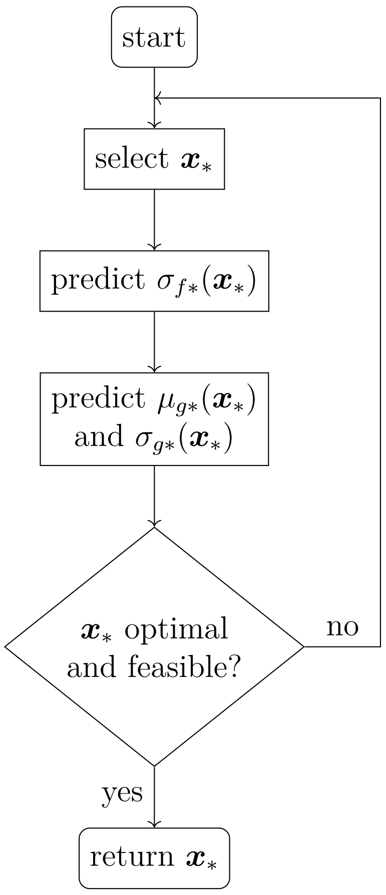
start
select $\boldsymbol{x}_*$
predict $\sigma_{f*}(\boldsymbol{x}_*)$
predict $\mu_{g*}(\boldsymbol{x}_*)$ and $\sigma_{g*}(\boldsymbol{x}_*)$
$\boldsymbol{x}_*$ optimal and feasible?
no
yes
return $\boldsymbol{x}_*$

Figure 5.3: Flow chart of the query selection of the SAL algorithm, compare Figure 5.2. This chart is only a rough simplification. In the actual implementation, an optimizer for constrained nonlinear functions is used to solve the optimization problem (5.2).

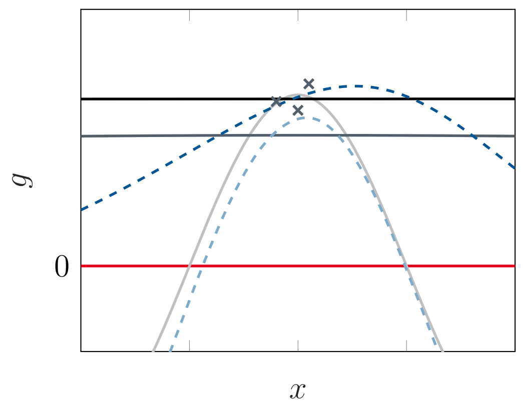

Figure 5.4: Comparison of the discriminative model's prediction with only three training points and different hyperparameters for an artificial example. The crosses denote the training samples, the lines predicted mean μ_{g*} (dark colors) and the safety criterion $\mu_{g*} - \nu\sigma_{g*}$ (light colors). The hyperparameters were either learned from the training points (which results in a very large length-scale; solid lines) or on 30 training points spanning the whole input range, which yields a better estimation (and smaller length-scale; dashed lines). The true discriminative function is plotted in gray. The confidence parameter was chosen as $\nu = 2.33$.

5.1.1 Training of the GP Models' Hyperparameters

As already explained in Section 2.4.3, GP models have a number of hyperparameters which need to be tuned. Schreiter et al. assume in [51] that the hyperparameters are previously given. This is generally not true in case of real-world applications. In non-active learning frameworks, the hyperparameters are often fitted by maximizing the model's likelihood (2.26) based on training data. This poses some problems in case of SAL. The hyperparameters cannot be estimated as long as less samples than hyperparameters have been measured, i.e. during the first few cycles of the algorithm. But even if just enough data is available to make the hyperparameters calculable, the estimated parameters can still be unreliable, e.g. because of noise in the few measured points, compare Figure 5.4. Wrongly estimated hyperparameters, especially of the discriminative model, can imply safety violations. Experiments showed, that the length-scales of the discriminative model are often overestimated with little training data, as shown in the mentioned figure. In this example, the length-scale is estimated much larger than it is estimated using more training samples. Thus, the feasible region of the input space (where the true discriminative function is positive) is massively overestimated when using learned parameters, while it is met well with better estimated hyperparameters. As a result, the SAL algorithm might select infeasible queries. This is less likely when using an improved estimate of the hyperparameters, compare the dashed lines.

Hence, it is not possible to simply learn the hyperparameters during the SAL run reliably. Three possible alternatives are:

- sample initial (space-filling) points and learn the hyperparameters from these points,
- estimate the hyperparameters using expert knowledge about the system under test, or
- perform a constrained optimization of the hyperparameters during the SAL run with heuristically chosen constraints.

In the first case, additional measurement points are sampled in advance in order to optimize the hyperparameters. This approach is suggested in [51] and used in their simulation examples. It has the disadvantage, that additional measurement points are required, which might not be placed optimally. To guarantee the safety of the measurement, the points have to be sampled in a subspace of $\mathbb{X}$ known to be safe by expert knowledge. This may also deteriorate the quality of the points' distribution with respect to model learning. As will be shown in Section 6.2.4 and [47], the proper choice of the number of initial samples is crucial for a successful SAL run, if the hyperparameters are learned this way and the total number of measurement points is limited.

The second approach needs even more expert knowledge. Usually it is very hard for a human to estimate proper hyperparameters. This holds especially true if the human is an expert in the technology of the system under test, but not in the modeling algorithm. In special cases, this approach is still possible. In the following, a heuristic is proposed for the hyperparameters of the discriminative model. It was successfully used for determination of the initial hyperparameters for the identification of the HPFS system, see Section 6.2.4 and [47] and as fixed hyperparameters for the controller tuning task presented in Section 6.2.6.

According to [39], the length-scales of a GP model with squared exponential kernel can be interpreted as the distance one has to move in the input space before the modeled function changes significantly. Thus, large length-scales of the discriminative

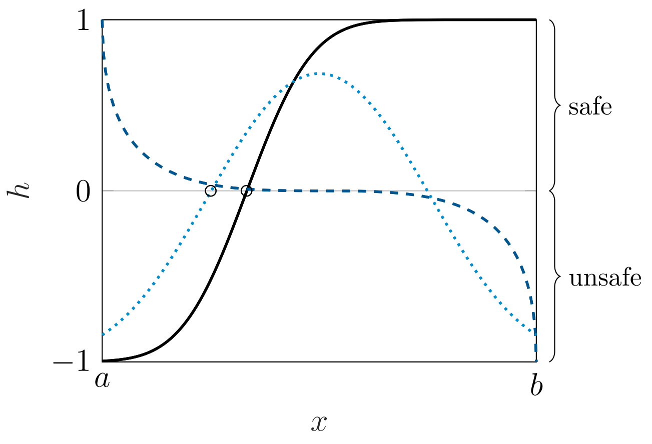

Figure 5.5: Three representative examples for a discriminative function h without noise in 1D. In this case, $\mathbb{X} = [a, b]$ and $\mathbb{X}_+ = \{x \in \mathbb{X} : h(x) > 0\}$. As one can see, two of the three examples have one zero-level upcrossing, marked with circles, the last one has none. The figure was taken from [47].

model will lead to a fast exploration, because little changes in the discriminative function are assumed. On the other hand, small length-scales result in a more careful behavior, as a high variance is predicted even in small distances from previously measured points. A motivation for suitable hyperparameters, no formal derivation though, can be given based on the expected number of level zero upcrossings in a 1D example. According to [39], the mean number of level-zero upcrossings n_0 in the unit interval for a GP with squared exponential kernel is

$$\mathbb{E}(n_0) = \frac{1}{2\pi\lambda}, \tag{5.3}$$

where λ is the GP's length-scale. If $\mathbb{X}_-$ is assumed non-empty and as $\mathbb{X}_+$ was already required to be compact and connected, the discriminative function h is expected to have $\frac{2}{3}$ level-zero upcrossings on average. This is illustrated in Figure 5.5: The feasible input space, denoted by $h > 0$, can be either on the left side of the input space, in the middle or on the right side. As all possibilities are equally likely a priori and two of the three possibilities have one level-zero upcrossing while the third one has none, an expected value of $\frac{2}{3}$ results. If $\mathbb{X}$ is scaled to the unit interval, a length-scale of

$$\lambda = \frac{\Delta x}{2\pi\mathbb{E}(n_0)} = \frac{\Delta x}{2\pi\frac{2}{3}} \approx \frac{\Delta x}{4} \tag{5.4}$$

is expected with the input space's extent $\Delta x = b - a$. The other hyperparameters of the discriminative model σ_g^2 and τ^2 can also be motivated. As the discriminative function should be roughly scaled to $[-1, 1]$, $\sigma_g^2 = 1$ is a good initial guess. τ^2 encodes the noise level of the discriminative function. This depends on the system under test and the measurement equipment. In the case of the two applications at the HPFS system, a low noise level was expected, thus $\tau^2 = 0.01$ was chosen. In case of the regression model, similar considerations can be carried out. They are more specific for the system to be modeled though and are thus discussed in Section 6.2.4 and 6.2.6.

The third possibility to obtain suitable hyperparameters mentioned above is a constrained optimization during the SAL run. This approach was chosen for the application at the HPFS system, compare Section 6.2.4 and [47]. Thereby, optimal hyperparameters can be found, but a non-successful optimization cannot lead to completely wrong parameters and dangerous over-estimation of the feasible input space. As stated above, the length-scale of the discriminative model is the most critical hyperparameter. At the same time it is prone to be chosen too large if little measurement data is available. Thus, it is limited to a multiple of the heuristically chosen parameters mentioned in the previous paragraph, as long as only a small number of samples has been measured. The limit is raised gradually as the optimized hyperparameters become more trustworthy than the initial heuristic. A similar approach is used in the ODCM algorithm, where the length-scales of the classifier are permanently limited.

5.1.2 The discriminative model

In the original publication [51] a problem-specific classifier is used as discriminative model. As explained above, it can be trained with continuous discriminative function values as well as discrete class labels. Even though the concept of this classifier is appealing, it does have two disadvantages in practical application, as was shown in [47]. First, hyperparameter training is not possible in the conventional way, i.e. by maximizing the model's marginal likelihood, if only labels of one class have been measured so far, compare [65]. This case is not uncommon if the algorithm starts with a very secure point, for which only a binary class label is available. Another drawback is that the discriminative model can gain little information from labels and the SAL algorithm might therefore generate multiple queries close to each other, especially if an infeasible point was selected. Compare Figure 5.6 for an example. This

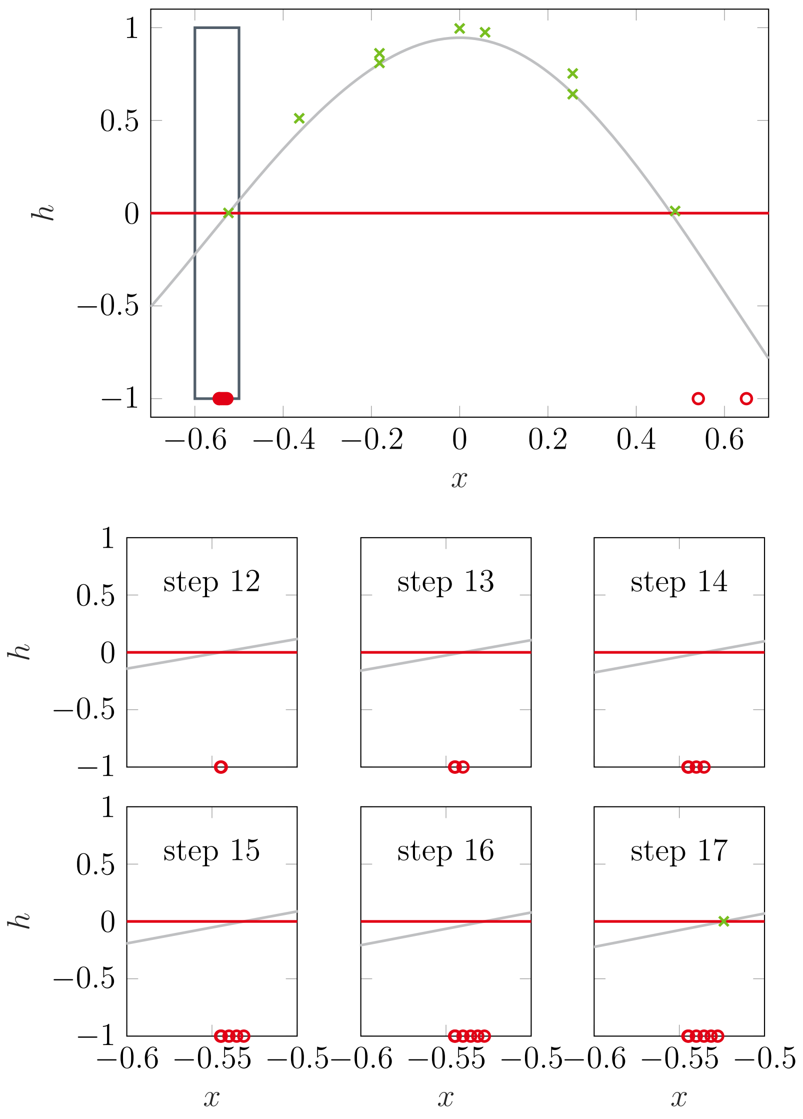

Figure 5.6: Artificial example showing the low impact of infeasible labels on the SAL algorithm with a hybrid classifier. Green crosses mark measured feasible discriminative function values, red circles measured infeasible class labels. The optimization constraint $\mu_{g*}(x) - \nu\sigma_{g*}(x)$ is plotted in gray, its decision boundary 0 is plotted in red. The six lower plots are zoomed into the upper plot and show the six previous steps of the algorithm. In total, five infeasible points (step 12 to 16) were sampled close to each other, before a feasible one (step 17) was measured. The infeasible points only provided labels to the algorithm, no discriminative function values. It is visible that the optimization constraint changes very slowly with the newly measured labels.

can be explained as follows: The infeasible point does not deliver any information to the regression model so that the result of the unconstrained optimization will be the same. Regarding the discriminative model, an infeasible label only tells that the discriminative function is negative at the label's location. A continuous training sample, on the contrary, delivers a specific value. The training algorithm can presume that the discriminative function was only slightly below zero at the location of the labeled training sample. In this case, even a point very close to the position of the negative label could be predicted to be larger than zero and thus feasible. This encourages the closely placed queries observed in Figure 5.6.

As an alternative, the problem-specific classifier was replaced by a regression model in the applications presented in Section 6.2.4, 6.2.6, and [45, 46, 47]. In these applications, continuous discriminative function values can be measured in the whole feasible input space $\mathbb{X}_+$ while only labels are provided in the complete infeasible input space $\mathbb{X}_-$. Thus, special measures to cope with these labels are required when using a non-hybrid model. In case of the HPFS modeling, alternative points close to the infeasible point are queried without using SAL's selection scheme until a feasible point is found. This converts SAL to a screening approach according to the categorization in Section 3.1. For the controller optimization problem, the infeasible labels were replaced by fixed negative discriminative function values. For example, if a query is measured as infeasible, the algorithm always gets the feedback that $h = -0.1$ was measured. Thereby, some kind of least-squares classification model is constructed, compare [39]. Even though this kind of model has some theoretical flaws[I], it showed good performance in the mentioned application. The Laplace approximation which the hybrid classifier features to calculate the posterior is not required when using a regression model. This is another advantage of the simplified model, as the posterior can be calculated analytically and no numerical approximation is necessary. The fixed discriminative function value provided when an infeasible sample occurs can be used as a tuning parameter. It defines the influence of infeasible samples on the discriminative model. A value far below zero will make the exploration more cautious than a value only slightly negative. Nonetheless, this additional tuning parameter is rather a disadvantage, as a similar effect can already be achieved by the parameters p or ν, as introduced above.

[I] For example, GP regression assumes a Gaussian noise which is not true if labels are used for training.

5.1.3 The risk function

As mentioned earlier, the discriminative function values used to train the discriminative model should encode the health state of the system. Usually, the health of the system is not directly measurable. Therefore, it is calculated from other quantities which can be directly measured at the system. These are called *supervised outputs* $\boldsymbol{z}$ in the following. Examples for supervised outputs are

- pressures (e.g. in case of the HPFS system),
- temperatures (e.g. exhaust gas or engine temperature),
- turbo charger speed or
- controller loss function value (in case of the application described in Section 5.2).

The scalar discriminative function h, which is learned depending on the system's inputs, should be roughly in the range $(-1, 1)$, where positive values are feasible and negative values are infeasible. This scaling simplifies the selection of hyperparameters and the tuning of parameters like ν. Thus, a risk function $\tilde{h}(\boldsymbol{z})$ is necessary, which combines the multiple measured supervised outputs $\boldsymbol{z}$ to a scalar discriminative function value in the desired range. In the temperature-example introduced at the beginning of Section 5.1, z would be the supervised temperature, while the risk function could, for example, be defined as

$$\tilde{h}(z) = \frac{z - z_{\max}}{z_{\min} - z_{\max}}. \tag{5.5}$$

This function maps the feasible range $(z_{\min}, z_{\max})$ of the supervised temperature z to discriminative function values h in the range $(0, 1)$. Infeasible temperatures $z \geq z_{\max}$ are mapped to negative discriminative function values $h \leq 0$. In case of multiple supervised outputs, i.e. $\boldsymbol{z} \in \mathbb{R}^{d_z}$, one possible risk function would be

$$\tilde{h}(\boldsymbol{z}) = \min_{i \in \{1,\ldots,d_z\}} \left(\frac{z_i - z_{\max,i}}{z_{\min,i} - z_{\max,i}} \right). \tag{5.6}$$

The definition of $\tilde{h}(\boldsymbol{z})$ influences the dependence of h on the inputs $\boldsymbol{x}$ which subsequently affects the exploring behavior of SAL, compare Figure 5.2. The latter can be motivated with Figure 5.5: The solid line has a well-defined zero crossing.

Assuming the predicted discriminative function perfectly fits the real course but still has a non-zero variance, SAL's safety criterion will allow queries close to the zero crossing. This will lead to a fast exploration but with an increased risk of false positives, for example due to wrong hyperparameters. On the opposite, the dashed line has a small gradient near zero. In case of a perfect model match with the same variance and confidence parameter as before, the safety criterion will allow queries only in a larger distance to the zero-crossing. Additional measurement points will eventually reduce the variance and permit measurements closer to the real boundary. The exploration will be slower than in the first case but feature a reduced risk of large limit violations.

Thus, by suitably defining $\tilde{h}(\boldsymbol{z})$, the exploring behavior can be affected. Unfortunately, this approach is not realistic in practice, as the risk function needs to be defined before the SAL algorithm is started and the discriminative function depending on the system's inputs is estimated. Perhaps an online optimization of the risk function would be possible, but this would most likely significantly increase the complexity of SAL and raise reliability issues. Thus, this idea is not immersed in this thesis and the risk functions are fixed. See Section 5.2.1 and 6.2.4 or [46, 47] for the choices made in the different application examples.

5.2 Safe Active Learning and Safe Bayesian Optimization for Control

So far, this thesis focused on learning a model of a system under test. Thereby the inputs of the model are the physical inputs of the system (and additional delayed physical inputs or outputs in case of dynamic models, compare Section 2.4). In the conventional procedure described in Chapter 1, the model can be used for controller optimization in a second step. For this optimization, controllers with different parameter sets might be challenged to track a given reference trajectory with the model output. Their performance can be summarized in a scalar loss value. The optimization finally yields the parameter set which results in the smallest loss.

In this section, an alternative approach for controller parameter tuning is pursued. Instead of learning a dynamic model of the system, a stationary model for the controller's loss depending on its parameters is created. The inputs of this model are the parameters of the controller and the output is a loss value acquired by evaluating each parameterized controller at the same reference tracking task. The task of design of experiments changes accordingly: Instead of applying dynamic input trajectories to the open loop system and measuring the outputs, different sets of controller parameters are evaluated at the system in closed loop.

Two different approaches for online query selection are compared here: First, the already known stationary safe active learning. Second, a newly proposed safe Bayesian optimization approach. While active learning aims at distributing the queries such that maximum information is obtained in the whole input space, the used Bayesian optimization flavor trades exploration of the complete input space off against exploitation of the regions which are likely to contain the global optimum. First, the concept of the learned loss function and discriminative function is explained and defined in Section 5.2.1. In the following section, the fundamentals of safe Bayesian optimization (SBO) are presented. Afterwards, SAL and SBO for controller tuning are compared and discussed in Section 5.2.3.

5.2.1 Definition of Loss and Risk Function

The loss and the risk function are used to transform the measured trajectories of the system to scalar values. These describe the quality (in case of the loss function values) and the safety (in case of the discriminative function values calculated using the risk function) of the currently evaluated controller parameters. As already mentioned, the loss and discriminative function values are calculated based on a reference trajectory. For the application at the high pressure fuel supply system, it is shown in the upper plot of Figure 6.13. The controller is required to track the reference using the currently evaluated controller parameters. An optimal controller should achieve a minimum tracking error and at the same time a smooth actuation signal. These two criteria are merged in the loss function

$$J_1 = \frac{1}{n}\sum_{k=1}^{n}\left(1 - \exp\left(-\frac{\theta_\mathrm{e}}{2}(y(k) - r(k))^2\right)\right), \tag{5.7}$$

$$J_2 = \frac{1}{n}\sum_{k=1}^{n}\left(1 - \exp\left(-\frac{\theta_\mathrm{u}}{2}(u(k) - \bar{u}(k))^2\right)\right), \tag{5.8}$$

$$J = \max(J_1, J_2), \tag{5.9}$$

where $r(k)$, $u(k)$, and $y(k)$ are the time-discrete reference, actuation signal, and plant output trajectories of length n, whereas θ_e and θ_u are tuning parameters. In addition, $\bar{u}(k)$ represents a moving average of the actuation signal, given by

$$\bar{u}(k) = \frac{1}{l_\mathrm{w}}\left(\left(\sum_{l=k_\mathrm{min}}^{k_\mathrm{max}} u(l)\right) + w(k)u(k)\right) \tag{5.10}$$

with window size l_w, lower bound of summation $k_\mathrm{min} = \max\left(k - \frac{l_\mathrm{w}-1}{2}, 1\right)$, upper bound $k_\mathrm{max} = \min\left(k + \frac{l_\mathrm{w}-1}{2}, n\right)$, and the additional fade in / fade out weight

$$w(k) = \begin{cases} \frac{l_\mathrm{w}-1}{2} - k + 1, & k \leq \frac{l_\mathrm{w}-1}{2} \\ 0, & \frac{l_\mathrm{w}-1}{2} < k \leq n - \frac{l_\mathrm{w}-1}{2} \\ \frac{l_\mathrm{w}-1}{2} + k - n, & k > n - \frac{l_\mathrm{w}-1}{2}. \end{cases} \tag{5.11}$$

The fade in / fade out weight is required to properly handle the first and last samples of the recorded trajectory, where not enough samples are available to calculate the moving average in the usual way. In this cases, $u(k)$ is weighted higher.

J_1 penalizes a large control error, whereas J_2 punishes rapid and oscillating control actions. Using this definition, J is always in the range $[0, 1)$, which is advantageous e.g. for setting the GPs' hyperparameters. This scaling is the main reason for choosing an exponential loss function.

The safety criterion is also twofold in this application: On the one hand, the maximum allowed output value of the plant must not be exceeded. On the other hand, very bad controllers should be excluded to save measurement time. This motivates the risk function

$$\tilde{h}_1 = \frac{\max_{k\in[1,n]}(y(k)) - y_{\max}}{y_{\text{minmax}} - y_{\max}}, \tag{5.12}$$

$$\tilde{h}_2 = 1 - \frac{J}{J_{\max}}, \tag{5.13}$$

$$\tilde{h} = \min(\tilde{h}_1, \tilde{h}_2), \tag{5.14}$$

where $y_{\max}$ is the maximum allowed output value, y_{minmax} is the lowest expected maximum output value (e. g. the maximum of the reference trajectory), and $J_{\max}$ is the maximum allowed loss function value. The risk function as well as the loss function were chosen based on simulation results. Defined like this, the loss function and the discriminative function with controller parameters as input can be modeled well using GPs. Some alternative choices led to severe local nonlinear effects, which are hard to model using GPs with squared exponential kernel.

5.2.2 Safe Bayesian Optimization

Bayesian optimization is a technique to efficiently find the optimum of a cost or loss function in cases where the function is expensive to evaluate. It features a GP model which approximates the loss function. Instead of frequently evaluating the loss function to find its optimum, intermediate optimization steps are performed at the surrogate model and their result are validated at the real loss function. Thereby, the model helps to find the optimum, but it also needs informative measurements to accurately reproduce the real loss. Thus, a trade-off between an informative training point for the model (exploration) and a potential optimum (exploitation) is necessary when choosing the next query. A good overview on the topic is given in [8]. A more general discussion of optimization with surrogate models is provided in [63].

At first sight the goals of active learning and Bayesian optimization might look very different. On the mathematical level, they can be almost identical, nonetheless. While the variance-criterion used in SAL is

$$\boldsymbol{x}_{i+1} = \operatorname*{argmax}_{\boldsymbol{x}_* \in \mathbb{X}} \sigma^2_{f*}(\boldsymbol{x}_*), \tag{5.15}$$

see (5.2), the frequently used lower confidence bound criterion for Bayesian optimization is given as

$$\boldsymbol{x}_{i+1} = \underset{\boldsymbol{x}_* \in \mathbb{X}}{\operatorname{argmin}} \, \mu_{f*}(\boldsymbol{x}_*) - \kappa \sigma_{f*}(\boldsymbol{x}_*). \tag{5.16}$$

Apart of minor disparities irrelevant for the optimization result (the square in (5.15) and the minimization of negative σ_{f*} in (5.16) instead of maximization), the only real differences between both objectives are the predicted mean μ_{f*} and the tuning parameter κ. The latter allows to weight exploration against exploitation: With $\kappa \to 0$ the next point is chosen at the predicted minimum, i.e. the model is purely exploited. On the opposite, $\kappa \to \infty$ transforms Bayesian optimization to active learning, which solely explores the input space.

The following considerations might lead to a proper choice of κ: First, all regions of the parameter space without any measurement points should be visited at least once. Furthermore, the algorithm should not get stuck in local optima. In regions without any measurement points, the GP model predicts its prior mean $\mu_{f,0}$ and standard deviation $\sigma_{f,0}$. Parameter sets with this prediction shall be preferred to others with an almost perfect prediction (resulting from very dense measurements), i. e. $\sigma_{f*} \to \sigma$, even if the latter are close to the theoretical minimum of the loss function, which means $\mu_{f*} \to J_{\min}$. Thus, (5.16) implies

$$J_{\min} - \kappa\sigma \overset{!}{>} \mu_{f,0} - \kappa\sigma_{f,0} \tag{5.17}$$

$$\Leftrightarrow \kappa > \frac{\mu_{f,0} - J_{\min}}{\sigma_{f,0} - \sigma}. \tag{5.18}$$

When choosing controller parameters, safety constraints must be met. For example, a bad choice of parameters could make the closed loop unstable. Thus, the Bayesian optimization approach is enhanced in the same way as SAL. A second discriminative model is introduced and its predicted probability of feasibility is incorporated as constraint in the selection of the next query. This leads to the safe Bayesian optimization (SBO) selection criterion

$$\boldsymbol{x}_{i+1} = \underset{\boldsymbol{x}_* \in \mathbb{X}}{\operatorname{argmin}} \, \mu_{f*}(\boldsymbol{x}_*) - \kappa \sigma_{f*}(\boldsymbol{x}_*) \tag{5.19a}$$

$$s.t. \quad \mu_{g*}(\boldsymbol{x}_*) - \nu \sigma_{g*}(\boldsymbol{x}_*) \geq 0. \tag{5.19b}$$

This SBO approach was already introduced by the author of this thesis in [46]. There, it was compared with SAL for tuning a PI-controller of a high pressure fuel supply system. This application is also discussed in Section 6.2.6. The application specific choices of parameters and the definition of the risk function required to calculate discriminative function values are also presented there.

A more complex Bayesian optimization algorithm which respects safety constraints was published in [58]. The algorithm called SafeOpt is compared to a simpler approach similar to that presented above. In [5], SafeOpt is used to optimize controller parameters of a quadrotor.

5.2.3 Comparison of SAL and SBO for controller optimization

This subsection reiterates the comparison already published in [46]. Both approaches, SAL with subsequent offline optimization and SBO, have certain advantages compared to classical controller tuning approaches like Ziegler-Nichols method or manual tuning:

- No open-loop measurement is necessary, thus, they are also applicable to unstable processes.
- The methods can be automated so that no user interaction during the measurement is necessary.
- The loss function for the controller can be adapted to custom requirements and preferences.
- The methods are essentially applicable to a wide range of parameter-based controllers, not only to PID controllers.

Comparing SAL with SBO, both methods show different strengths:

- In case of objective functions that are very hard to optimize and possess many local optima, SAL could probably show better results than SBO. The sampling procedure of SAL is not sensitive to local optima and in the subsequent offline optimization, more advanced optimization techniques can be used. Nonetheless, this point needs more detailed research.

- If the loss function is adjusted after the measurements were conducted, it is likely that measurements using SAL can be reused due to their space-filling distribution. In case of SBO, the measurements need to be repeated if the location of the loss function's optimum changes.

- For the application presented in Section 6.2.6, SBO obtains a better optimization result requiring less measurement samples.

Of course, both methods also have disadvantages and weaknesses:

- The implementation of SAL and SBO can be more time consuming than using simpler methodologies. This effort might amortize, if multiple similar controllers have to be optimized.

- Both methods use tuning parameters, which have to be suitably chosen.

- SAL learns a model in the whole parameter space, not only in those regions where the optimum is found. This can result in a waste of measurement time.

- A repetition of the optimization is expensive in case of SBO, as all measurements have to be repeated. In case of SAL, the offline optimization can be repeated multiple times with different optimization algorithms, also global ones, at low cost.

- In future research, the convergence of SBO should be analyzed. Due to the restrictions of the discriminative model, it is not guaranteed that the method will find the optimum in every case. For example, [58] present a counterexample.

5.3 Dynamic Safe Active Learning

Up to this point, this chapter covered active learning for stationary systems. In the following, safe active learning for dynamic systems is discussed. First, the different requirements for stationary and dynamic SAL are described, followed by a summary of the previous work on dynamic SAL. In the next subsection, the challenges of applying dynamic SAL to a real system are presented. Finally, advantages and caveats of the current implementation are discussed and proposals for future work are given.

	stationary SAL	dynamic SAL
system is time dependent	no	yes
regression model	stationary	dynamic
discriminative model	stationary	dynamic
typical number of samples	low ($10^1 \sim 10^3$)	high ($10^4 \sim 10^6$)
GP models	regular	sparse
optimization variable	next point	parameterized trajectory
real-time capability	not required	required
space to be explored	physical inputs	also delayed inputs

Table 5.1: Comparison of the requirements on stationary and dynamic SAL.

5.3.1 Methodology

In case of dynamic systems, several aspects change compared to the stationary case, which render the SAL algorithm more complex. Table 5.1 provides an overview. The most notable difference is that dynamic instead of stationary models have to be used. This does not only affect the regression model, but also the discriminative model. Hence, the feasible and infeasible input spaces $\mathbb{X}_+$ and $\mathbb{X}_-$ become time-dependent as well, i.e. $\mathbb{X}_+(t)$ and $\mathbb{X}_-(t)$. The complete input space $\mathbb{X}$ defined by box-constraints remains time independent, though.

The number of samples obtained from the system under test often differs by multiple orders of magnitude. Stationary systems in the automotive domain are usually modeled using a few tens to a few thousand samples. Dynamic systems are commonly sampled with a fixed sampling time. Many systems in the automotive domain operate with sampling times in a range of 10 ms to 100 ms. Each sampling interval a new sample is measured, which sums up to a large number already after a short measurement time. For example, if the system is sampled at 10 ms sampling time for 10 min, 60 000 samples are measured. This is a much higher number than in the stationary case and a burden for the modeling algorithm. Especially GP models suffer from a high number of training samples m, as the complexity for model training is $\mathcal{O}(m^3)$, see [39]. Thus, more efficient sparse GP models were developed. They use different approximations to reduce the complexity to $\mathcal{O}(mm_{\text{idx}}^2)$, where m_{idx} denotes the size of an index set and is usually much smaller than m, see e.g. [50].

To optimize each sampled point in case of dynamic SAL would be a huge computational load. The optimization of the next query would be necessary in every sampling interval, i.e. each 10 ms to 100 ms as mentioned above. Furthermore, consecutive points are not independent in the dynamic case, due to limited allowed gradients. Thus, it is reasonable to bundle multiple points to trajectories, which are parameterized and whose parameters are optimized by the dynamic SAL algorithm. Still, the real-time capability of the algorithm is a challenge. Each new trajectory needs to be planned until the previous is completely measured. This is opposed to the stationary case, where no hard bound on the planning time exists.

The input space relevant for exploration is also different in the stationary and dynamic case. This space is to be explored by the learning algorithm so that the final model can provide reliable predictions in the largest possible fraction of the feasible part of the input space. Therefore, GP models need a good coverage of all combinations of their inputs. In the stationary case the input space is given by the physical inputs of the system and denoted as $\mathbb{X}$. As mentioned in Section 2.4.3, dynamic GP models with an external dynamics approach use an external NARX structure to represent the dynamics of the system. This results in a feature vector $\boldsymbol{\varphi}$ as defined in (2.27), which is fed into a stationary GP model which performs one-step predictions. Hence, it is not sufficient to have a good coverage of $\mathbb{X}$ for good predictions, but the space spanned by all the features in $\boldsymbol{\varphi}$ has to be covered instead.

The used type of trajectory influences which features can be covered well. If, for example, ramps are optimized, it is hard to excite higher derivatives of the input, as these are either zero or infinite.

In the literature, several approaches for dynamic active learning algorithms were published, especially with focus on combustion engine measurements. Reference [12] proposes an adaptive experiment design for dynamic engine measurements. APRB-signals or ramp-hold-signals are optimized online using either D-optimality or a model-committee criterion. Multilayer perceptron neural networks are utilized for modeling. The different methods are evaluated at a gasoline engine. The evaluation not only focuses on the dynamic accuracy, but also highlights an improved stationary accuracy of the identified models. In [11] the approach is enhanced to Runge-Kutta neural networks. Furthermore, system limits are incorporated in the design, which were omitted earlier.

In [57] an online DoE based on the receding horizon principle of model predictive control is presented. The future trajectory is optimized in each step for a finite

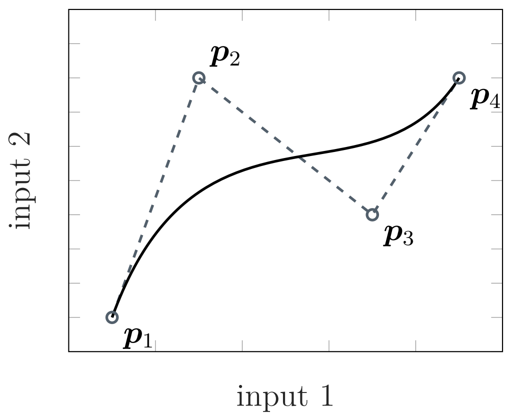

Figure 5.7: An example of a cubic Bézier curve in 2D.

prediction horizon, but only the first planned point is applied to the system. A D-optimality criterion is used to generate most informative queries. System output limits are also incorporated. For modeling, local model networks are chosen. The approach is successfully evaluated in simulation. Unfortunately, the runtime of the algorithm is not addressed. Probably this idea will be hard to apply to a real-world system, as the optimization in each time step might be computationally demanding. In [56] the selection criterion is further developed towards a D-S-optimality criterion. More details on the algorithm and further simulation examples are presented. Real-time capability is mentioned as a challenge, but it is not stated whether it was achieved and, if so, for which sampling time.

In [49], Schreiter presents an approach for dynamic SAL based on his stationary SAL algorithm. Thereby, cubic Bézier curves are optimized. See Figure 5.7 for an example of a Bézier curve in 2D. These curves are defined by four control points $\boldsymbol{p}_1$ to $\boldsymbol{p}_4$. Two of the points denote beginning and end of the trajectory, while the two others attract the curve and thereby also influence the curve's direction at its ends, compare e.g. [42]. According to [49], Bézier curves show multiple advantages for dynamic SAL: They can be computed efficiently, it is possible to define not only their endpoints but also the gradient at the endpoints, and the complete curve is guaranteed to live within the convex hull given by the four control points. Using this property it is easy to check if a planned curve is within $\mathbb{X}$. Nonetheless, if the test fails, the trajectory can still live in $\mathbb{X}$ and more complex checks are required.

For modeling, [49] introduces a sparse deterministic training conditional (DTC) GP approach with a maximum error insertion criterion. Both the regression and the

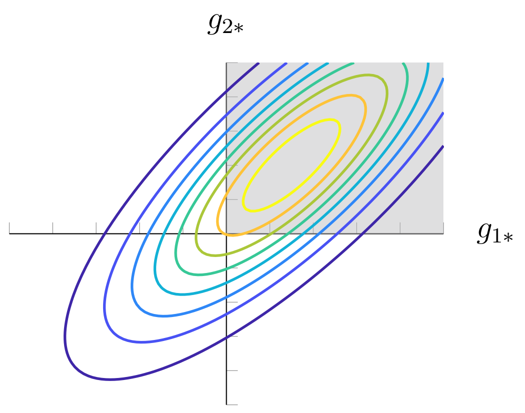

Figure 5.8: Visualization of the probability density function cumulated in (5.20) for a trajectory with two points. The contour lines denote the predicted posterior distribution of the discriminative function value $\boldsymbol{g}_*$ for a trajectory with 2 points. The distribution is Gaussian and has a maximum in the first quadrant. Equation (5.20) integrates this distribution for $g_{1*} > 0$ and $g_{2*} > 0$, i.e. in the gray shaded area. In this quadrant, both discriminative function values are positive and thus feasible.

discriminative model can only be trained using continuous data, i.e. no discrete class labels. Thus, no application specific classifier is used, opposed to the stationary case. The hyperparameters and the NARX structure of the GP models are assumed to be known in advance. Even though this is a severe restriction for practical applications, it simplifies the theoretical analysis and increases the robustness of the algorithm.

In fact, the selection of new measurement points does not use any online criteria. Instead, a sobol set in phase space is created. The phase space is spanned by all input signals and their derivatives. From the sobol points in phase space points are randomly drawn and arranged in a binary tree. Each path in this tree represents a possible trajectory, if the sobolpoints are connected using Bézier curves. As mentioned above, these curves are well-defined using the positions and gradients of their endpoints. For each trajectory the possibility of feasibility is calculated. Finally, the most safe trajectory is chosen and the first Bézier curve of this trajectory is applied to the system. Afterwards the described procedure is repeated.

Besides this trajectory generation scheme, Schreiter's thesis focuses on the safety criterion. Similar to the stationary case, a discriminative model is learned to separate feasible from infeasible trajectories. The criterion therefore is similar to (5.1), i.e.

$$\Pr(\boldsymbol{g}_* \geq \boldsymbol{0}|\boldsymbol{X}_*, \boldsymbol{h}, \boldsymbol{X}) = \int_{\boldsymbol{0}}^{\infty} \mathcal{N}(\boldsymbol{g}_*|\boldsymbol{\mu}_{g*}, \boldsymbol{\Sigma}_{g*})\mathrm{d}\boldsymbol{g}_* \geq p. \tag{5.20}$$

Opposed to the stationary case, $\boldsymbol{g}_* \in \mathbb{R}^n$ is now a vector instead of a scalar, as not only one point but a trajectory consisting of multiple points $\boldsymbol{X}_* \in \mathbb{R}^{n\times d}$ is to be checked for safety. Thus, the normal distribution on the right hand side including the vector of predicted mean values $\boldsymbol{\mu}_{g*} \in \mathbb{R}^n$ and the predicted covariance matrix $\boldsymbol{\Sigma}_{g*} \in \mathbb{R}^{n\times n}$ are multidimensional, compare Figure 5.8 and Section 2.4.3. Thereby, n denotes the number of samples of the current trajectory and d is the dimension of the input space. The integration has to be conducted over each dimension of $\boldsymbol{g}_*$ so that $\Pr(\boldsymbol{g}_* \geq \boldsymbol{0}|\boldsymbol{X}_*, \boldsymbol{h}, \boldsymbol{X})$ is a scalar. Consequently, a trajectory of e.g. 1 s duration at a sampling time of 10 ms consists of $n = 100$ samples, which results in a 100-dimensional integration. Unfortunately, the integral cannot be solved analytically and some common approximations also fail because of the high number of dimensions. In [49] several possibilities to approximate the integration are presented. The algorithm is evaluated in simulation.

The algorithm from [49] was further developed at Bosch Corporate Research. The theoretical aspects of this algorithm are covered in [68]. The new version features two main differences to the original one: First, instead of Bézier curves, ramps are used. They can be computed more efficiently and especially the check if a trajectory is within the box-constraints framing the phase space is less computationally demanding. Second, the algorithm was extended to be a “real” active learning strategy, i.e. the trajectory sections are optimized during the measurement procedure. Therefore, a maximum entropy criterion similar to the stationary case is used. Similar to the discriminative model, this algorithm now has to incorporate a covariance matrix $\boldsymbol{\Sigma}_{f*}$ instead of a single scalar variance. In order to apply an optimization algorithm to $\boldsymbol{\Sigma}_{f*}$ it has to be transformed to a scalar value. In the literature, different approaches have been proposed for this step. For example, the determinant, the trace, or the maximum eigenvalue of $\boldsymbol{\Sigma}_{f*}$ could be maximized, see e.g. [17]. In the following, a simpler approach is chosen: The variance is not optimized along the complete trajectory but only at the trajectory’s endpoint. Even though this optimality criterion is not as mature as those considering the full matrix, it omits the necessity to calculate the complete matrix $\boldsymbol{\Sigma}_{f*}$ and thereby reduces the computational workload.

To avoid a necessary simulation to calculate the variance at the endpoint, a nonlinear finite impulse response (NFIR) structure without output feedback is used for the regression model; see Figure 2.4 for an illustration. With a NFIR structure, the variance at the planned trajectory's endpoint can be calculated in one step. If a NARX model was used, predictions for each intermediate point on the trajectory were necessary, as the model depends on previous outputs. Formally stated, the predicted variance from the intermediate steps needs to be considered in the following steps, as the fed back output is not a single value anymore but a distribution. While the first aspect is simply a computational burden, the latter is mathematically demanding. Thus it is left for future research. This effect is also relevant for the discriminative model, which is implemented as NARX model to obtain a better prediction accuracy. Therefore, a simplification is used: first, only the mean values are iteratively calculated for all intermediate points, ignoring the distribution of the fed back outputs. Second, the covariance matrix for all points is calculated in one step.

In fact, only the current inputs and their derivatives are used as features of the regression model. This is equivalent to using the current and one delayed input per input dimension as features. Using the derivatives has the advantage that it directly is a property of the used ramps. In combination these features uniquely define a ramp trajectory. Derivatives of higher order would not contribute to the model output if used as features, as they are zero along a ramp segment and infinite at the transition to the following ramp. They do not take other values in between.

In a mathematical form, the next endpoint $\boldsymbol{x}_{i+1}$ and the length of the next trajectory n_{i+1} are obtained by solving the optimization

$$[\boldsymbol{x}_{i+1}, n_{i+1}] = \underset{\boldsymbol{x}_* \in \mathbb{X}, n_* \in [n_{\min}, n_{\max}]}{\operatorname{argmax}} \sigma^2_{f*}(\boldsymbol{x}_*, \dot{\boldsymbol{x}}_*(\boldsymbol{x}_i, \boldsymbol{x}_*, n_*)) \tag{5.21a}$$

$$s.t. \quad \Pr(\boldsymbol{g}_* \geq \boldsymbol{0} | \boldsymbol{X}_*(\boldsymbol{x}_i, \boldsymbol{x}_*, n_*), \boldsymbol{h}, \boldsymbol{X}) \geq p. \tag{5.21b}$$

Thereby, $n_{\min}$ and $n_{\max}$ are predefined lower and upper bounds for the length of the trajectory (compare the next section for details). Variable σ^2_{f*} denotes the predicted variance of the regression model, which depends on the currently evaluated inputs $\boldsymbol{x}_*$ and the gradients $\dot{\boldsymbol{x}}_*$. The gradients themselves can be calculated using the last trajectory's target $\boldsymbol{x}_i$, which is the current trajectory's starting point, the endpoint and the number of steps n_*. The constraint is determined using (5.20). The sampling points of the trajectory $\boldsymbol{X}_*$ can also be calculated from starting point, endpoint and number of steps. The integral in (5.20) is solved using a Monte Carlo approach. Thereby, a high number (e.g. 10 000) of random realizations of the vector $\boldsymbol{g}_*$ is picked from the distribution $\mathcal{N}(\boldsymbol{g}_*|\boldsymbol{\mu}_{g*}, \boldsymbol{\Sigma}_{g*})$. Afterwards, the number of feasible realizations (i.e. those where all elements of $\boldsymbol{g}_*$ are positive) is divided by the number of all realizations, which approximates the trajectory's probability of feasibility.

5.3.2 Challenges of a Real-Time Implementation

After some evaluations in simulation, the method was implemented for a real HPFS system as part of the master thesis [52] supervised by the author of this contribution. Because of already existing code and interfaces to the system the algorithm was implemented in Matlab. The main challenge was to give the algorithm a (soft) real-time capability. The development based upon the code used at Bosch Corporate Research for simulation experiments. The runtime of this code was analyzed in detail and the trajectory optimization was identified as the most time consuming part responsible for approximately 81 % of the total runtime. Consequently, a subsequent code optimization focused on this part. Especially the evaluation of a trajectory's feasibility, which is conducted more than 200 times per trajectory optimization on average, turned out to be expensive. Almost 98 % of the time needed for trajectory optimization was spent on the evaluation of the optimization constraint. This workload could be significantly reduced to approximately 10 % of the original time by a combination of multiple measures. Most beneficial were:

- The limitation of the permitted duration of trajectories. During the trajectory optimization, matrix multiplications with $n \times n$-matrices have to be calculated, where n is the length of the trajectory. These scale with $\mathcal{O}(n^3)$. Thus, it is faster to optimize multiple short trajectories with small n instead of less but longer ones with large n. On the other hand, a too short trajectory length does not leave enough time for the following optimization. Therefore, also a lower bound for the trajectory length was introduced.

- The usage of single precision floating point variables instead of double precision in some critical parts. The decreased numerical accuracy turned out to be acceptable. As more data could be stored in CPU cache, some operations significantly gained speed.

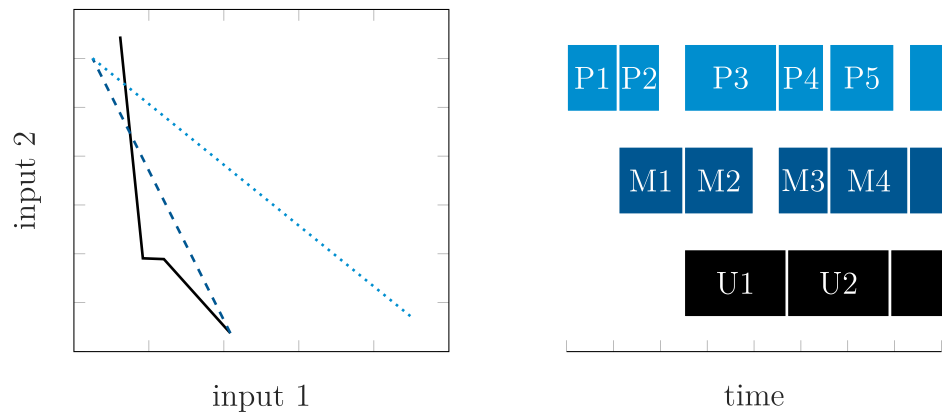

Figure 5.9: Illustration of the parallelized dynamic SAL algorithm. The left plot shows the input space at one certain time instance. The black continuous ramps have already been measured and are used for a model update. At the same time, the dark blue dashed ramp is measured. In parallel, the light blue dotted ramp is planned. The right plot shows how planning (top row), measuring (middle row) and model update (bottom row) for each trajectory (indexed by a number) follow after one another.

- The initial generation of pseudo random numbers required for the Monte Carlo integration. The algorithm even benefits in accuracy from the reuse of the random numbers. The noise effect introduced by using a stochastic method becomes deterministic and does not affect the numerical derivation of the probability of feasibility done by the optimization algorithm.

Other approaches like the parallelization of the trajectory optimization itself or the autogeneration of compiled MEX functions (Matlab Executable) did not result in a performance gain but even decreased speed. The overhead of parallelizing small code segments or calling MEX functions exceeds the benefits of these ideas.

Despite the speedup gained through the described measures, a parallelization of the algorithm is necessary. While in the stationary case the next point can be planned while waiting at the previous point or at a safe initial point, a dynamic measurement requires planning and measuring to run in parallel to avoid stationary waiting. Figure 5.9 illustrates the parallelized algorithm. As shown, the workload is split to three workers which run on different cores:

Planning The first worker is responsible for the optimization of the next ramp segment. Based on the current available regression and discriminative models it finds the trajectory which ends in the point with maximum variance and is presumably feasible. The planning of the next trajectory segments starts each time the measurement of the previous segment was finished. The result of the optimization is provided to the measurement worker.

Measurement The second worker applies the planned trajectory to the system under test, measures the results and continuously supervises the safety of the system. If a limit violation occurs, the measurement worker reacts by driving to a safe point. The measured data is provided to the model update worker. The measurement worker has to wait for the planning of the respective trajectory to finish.

Model update The third worker updates the GP models needed for the trajectory optimization. This worker runs mostly independent of the two others, because model training often needs more time than one trajectory optimization and measurement. As soon as new measurement data is available and the previous update is finished, the models are updated. If the update is completed, the new models are provided to the planning worker and used in the next trajectory optimization.

Besides the described main tasks, several undesirable special cases have to be taken care of by the implementation. For example, the optimization of the next trajectory might not find a feasible solution in a given time budget. In this case, the system is returned to a stationary safe central point and the optimization is repeated with this new starting point. If a limit violation occurs during the measurement, the automation drives the system in a safe point and the planning of the following trajectory is restarted. As worst case scenario, a limit violation at the endpoint of a trajectory segment is already taken into consideration when planning each segment. Not only the currently planned ramp, but also a return trajectory from the next selected point to the stationary safe central point (the direct connection of these points with maximum gradient) is assessed for safety. The evaluation of the presented dynamic SAL algorithm at the high pressure fuel supply system is presented in Section 6.2.8.

5.3.3 Discussion and Proposals for Future Research

The dynamic SAL approach is still a topic of research. As shown in Section 6.2.8, the algorithm presented above can be applied to a real-world system. Thus it is a starting point but offers possibilities for improvement on the theoretical as well as the practical side.

Currently, the NFIR structure used for the regression model featuring only current inputs and their derivatives is too simple for many real-world systems. Thus, after the dynamic SAL run a more complex NARX model with more features is trained using the data optimized on the simpler model. The implications of this procedure have to be studied in more detail. Most likely, data collected for the NFIR model is not optimal for the NARX approach. Directly optimizing for the NARX structure would render the dynamic SAL algorithm more complex, though. Each trajectory would need to be simulated starting from the last point. The predicted variance should be carried through the simulation to gain accurate results.

Probably ramp signals are not the best choice for the excitation of the system under test, even though they are widely used in everyday business. As stated earlier, higher derivatives of the system are not excited properly. Bézier curves do not have this disadvantage, but their computation is more demanding.

Computational efficiency is in fact a major challenge for dynamic SAL. As will be shown in Section 6.2.8, the current implementation mostly achieved a 40 ms sampling time but still suffered from occasional stationary waiting. Adding new features and more complexity to the algorithm will worsen this problem. A relief could be the implementation in a more efficient lower level programming language like C or C++. Nonetheless the obtainable speedup of these languages compared to Matlab is heavily application- and coding-style dependent. Thus, a noteworthy speedup can most likely only be obtained if the new implementation is done by an experienced programmer who focuses on speed.

Another idea for performance optimization is the used optimization algorithm. Currently the trajectories' endpoints and lengths are continuously optimized using a convex optimizer with multi-start. The requirements for the optimization algorithm are somehow special in this application. The optimization should be very fast, as it has to finish during one trajectory segment's length which is currently around 1 s. It would be beneficial if the optimization can guarantee to find a feasible solution after a certain time. As the objective function and the nonlinear constraint are non-convex, a local convex optimization is not sufficient. Nonetheless it is not necessary that the global optimum is found, as long as the found local optimum is “good enough”. The objective function’s gradient can be calculated analytically, as long as a model without output feedback is used. Due to the Monte Carlo approach, no analytical gradient is available for the constraint, though.

Given these requirements and depending on the dimension of the input space, a random brute-force search, see e.g. [28], could deliver performance gains. This quite simple approach is especially suited for low-dimensional problems. It will not find the global optimum, but has a deterministic runtime and no risk of getting stuck in local optima. In fact, random brute-force search is a special case of convex optimization with multi-start, where the number of multi-starts is high and the convex optimization performs zero iterations. A combination of the two is also possible, e.g. by using a high number of multi-starts and a convex optimization with a fixed low number of iterations.

An idea to simplify the optimization by reducing its dimension is to omit the trajectory length as an optimization variable. Currently, this variable is used as major influence on the gradient of the segment. Nonetheless, the gradient could still be varied by the distance of the trajectories endpoint to the current point. The reachability of points in the phase space during one optimization step will be significantly reduced by this approach, as gradient and endpoint become linked. Intermediate points of the trajectory will still be able to cover nearby points with high gradients and further away points could be reached with low gradients in multiple steps. Thus, the link of gradient and endpoint does not need to be a drawback, but the quality of the results still has to be investigated.

6 Applications

In this chapter, the methods described in the previous chapters are applied to different real-world problems and simulated systems. The results are presented and compared with each other. The chapter is a core contribution of this thesis. Most of the methods have not been applied to real-world systems before.

The chapter starts with a brief description of the implementation of the methods and the interfaces to the systems under test. The following Section 6.2 presents the application of all methods to the high pressure fuel supply (HPFS) system of a gasoline engine. Two of them, ODCM and dynamic supervised measurement, were also tested at a charge air system. These tests are addressed in Section 6.3.

6.1 Implementation

In order to evaluate the algorithms presented in the previous chapters at a real-world system, they have to be implemented as software. Besides the core algorithm usually a number of auxiliary functionalities are needed. They interface with the system under test, provide additional safety in case the core algorithm does not perform well, handle measured data or provide a user interface. Usually the amount of auxiliary code significantly exceeds the lines of code required for the core algorithm.

In the past, new algorithms for measurement automation were often implemented in a problem-specific manner. This means they are optimized to evaluate one algorithm at one system using one interface. The advantage of this approach is that the implementation is comparatively simple, fast, and does not require advanced programming skills. Especially for first prototypes, when not all requirements for the auxiliary code are known yet, this coding style can be suitable. On the other hand, the reuse and enhancement of this kind of code is often difficult. For example, the adaption to a new system under test might require changes in multiple parts of the program which implies a sound knowledge of the complete implementation.

For this reason, a more general approach was planned and implemented during this PhD project. The software is called *online measurement automation (OMA)*. It is implemented object-oriented in Matlab. The goal of this project was to create a flexible platform which can easily be enhanced by new measurement algorithms, adapted to new systems, and augmented with new interfaces. Furthermore, it should be a good starting point for specialized tools which can be used e.g. by calibration engineers. To achieve this generality, much effort was spent on a suitable division of the algorithm and the auxiliary functions in different classes. Abstract classes are used to define interfaces between single parts of the program which are interchangeable depending on the current situation. The experiments at the charge air system were done using OMA. It can also be used for measurements at the HPFS system. The experiments presented in Section 6.2 still used older, problem-specific implementations, though.

At the time of this writing, OMA consists of 58 classes and around 4000 source lines of code. In the following, only the most important classes are described. They are all abstract and inherit into multiple subclasses which contain the concrete implementations.

omaMethod

This class contains the core algorithms of the used DoE methods. In subclasses the methods ODCM and dynamic supervised measurement are implemented. Further subclasses, e.g. for stationary SAL, are planned. Existing code is used in the ODCM case, where the omaMethod class performs as a wrapper.

omaAutomation

The automation of the measurement process is implemented in this class and its two subclasses for stationary and dynamic measurements. In the stationary case, this consists of the steps shown in Algorithm 6.1. The duration of one loop cycle depends on the system under test and is typically in the range of a few seconds to a few minutes. Several subfunctions contain faster loops running in the sampling time

Algorithm 6.1 Main steps of the stationary automation used in OMA.

```
initialization and setup
repeat
    fetch next point from omaMethod
    change operating point if necessary
    drive to next point
    conduct measurement
    drive back to safe point
    store measurement results
until measurement is terminated by omaMethod
```

Algorithm 6.2 Main steps of the dynamic automation used in OMA.

```
initialization and setup
repeat
    fetch next point from omaMethod
    check for boundary violations
    apply next point to system
    conduct measurement
    store measurement results
until measurement is terminated by omaMethod
```

of the system. These loops also perform a constant supervision of the system and initiate emergency actions if necessary.

The dynamic case contains less steps, compare Algorithm 6.2. In exchange, the loop is repeated much faster, i.e. in each sampling interval down to 10 ms. Thus, the called subfunctions have to be faster and simpler as well and the supervision is directly performed in omaAutomation.

omaComAdapter

The communication adapters implement the interface to the system under test or a lower level automation like a test bench. They provide methods to set input signals and measure outputs. Currently communication adapters using INCA-MIP (Matlab Integration Package), iLinkRT, a manual interface and adapters to several simulation models are available. An adapter for the AK-protocol used by chassis dynamometers is in development. See Appendix A for a brief description of the used software products. Figure 6.1 shows the different interfaces and their applications. INCA-MIP and

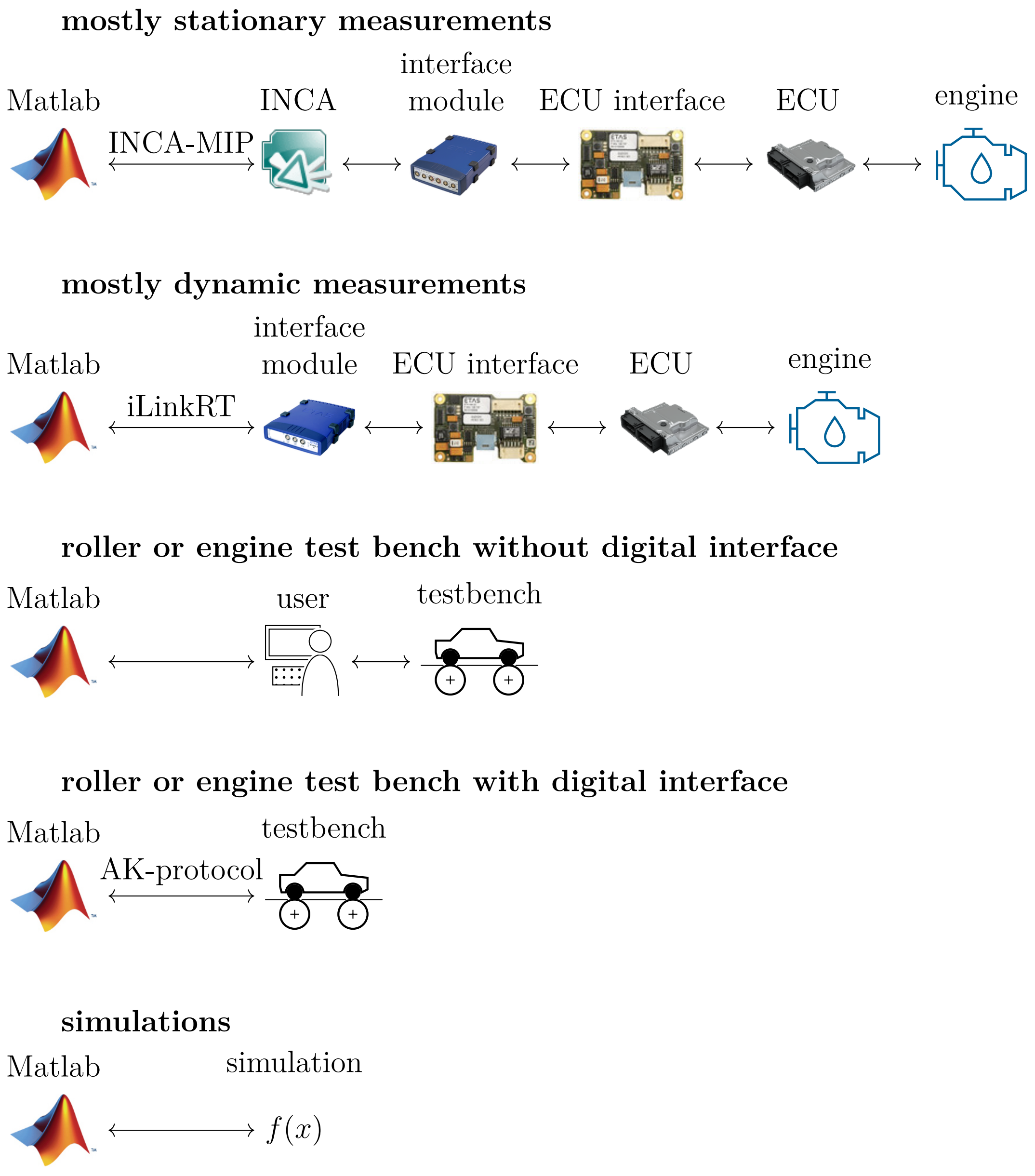

Figure 6.1: Interfaces of OMA provided by different communication adapters. From top to bottom: INCA-MIP, iLinkRT, manual interface, AK-protocol, and simulation adapters.

iLinkRT are targeted at setting and measuring variables in an engine control unit (ECU). While INCA-MIP connects to the software ETAS INCA and subsequently to an interface module, iLinkRT directly communicates with said module. The latter approach is faster. One read or write cycle needs less than 1 ms with iLinkRT compared to about 10 ms using INCA-MIP. This speed comes at a cost, though: the connection lacks features like buffering which introduces the risk of missed measured values, the required interface module is more expensive than a module sufficient for the use with INCA, and the measurement of additional sensors connected to the interface module (not pictured) is not supported. In both cases, the interface module is connected with an ECU interface (ETK), which allows to read and write values in the ECU's memory. Finally, the ECU applies the signals to the engine or measures values using the engine's sensors.

Probably the most simple communication adapter is the manual interface. It makes use of a graphical user interface to display input values to the user and requests measured outputs from him in an input field. The user manually applies or reads these values to or from a test bench automation. This adapter can be used to perform stationary measurements at a test bench with no digital interface to a PC. Of course it is not suitable for dynamic measurements and does not allow a completely automated measurement run. To overcome these disadvantages, an adapter for the AK-protocol is currently in development. This industry standard is used by several test bench automation systems and allows to directly read or set signals like the speed of the test bench.

For testing new measurement methods or systems under test, simulation adapters can be used. They contain Matlab functions implementing simulation models and thereby allow cheap, fast, and safe early evaluations.

omaConfig and its referenced classes

This class and a number of referenced classes feature only few methods but mainly properties. They store various data required by more than one other class, for example

- system dependent settings like the names, dimensions, and interfaces of all signals in omaSystem,

- connection settings for the communication adapters in omaInterface,
- planned inputs and measured outputs in omaData, and
- links to functions which perform emergency actions in case of hard limit violations.

Thus, omaConfig is the central database for OMA. Parts of the class objects can be saved to disk and used in future measurements.

User interface

OMA features a graphical user interface. It allows the user to control the measurement process and conveniently view and alter settings. For some application, it also provides an online visualization of planned and conducted measurements. In the future, additional functionality e.g. for the evaluation of measurement results is planned. The user interface also has an abstract superclass which defines its interface to the rest of OMA. Thus, it is possible to comparatively simple implement other interfaces like a text console or a tablet app.

6.2 High Pressure Fuel Supply System

The first system under test considered in this thesis is the high pressure fuel supply (HPFS) system of a gasoline engine. All methods presented in Chapter 3 to 5 are evaluated at this system. Even though at least some of the methods and a suitable interface are available in OMA, older problem-specific implementations were used for the measurements in this chapter.

Due to the different goals and focuses of the evaluated methods, some rather hard transitions might occur between the subsections of this section. It is structured as follows: First, the fundamentals of the HPFS system are presented. Afterwards, all methods for stationary boundary and system identification are examined: Starting with the classical ODCM (Section 6.2.2) via ODCM's variant with improved boundary estimation (Section 6.2.3) to stationary safe active learning (Section 6.2.4) and the union of ODCM and SAL, ODCM with discriminative model (Section 6.2.5). A special

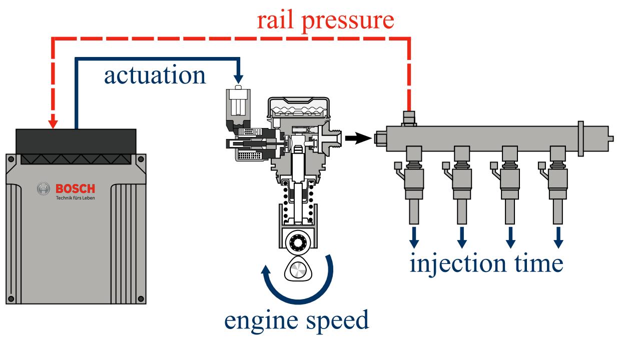

Figure 6.2: Sketch of the high pressure fuel supply system's main components, inputs (solid lines), and output (dashed line). The figure is taken from [62].

application of SAL and safe Bayesian optimization (SBO) for controller tuning is presented in Section 6.2.6. Finally, the two methods for dynamic system identification are tackled: Supervised dynamic measurements (Section 6.2.7) and dynamic SAL (Section 6.2.8).

6.2.1 System Description

The high pressure fuel supply system of a gasoline engine consists of four main components: The high pressure fuel pump, the rail, the injectors, and the engine control unit. They are presented in Figure 6.2. The high pressure fuel pump compresses the fuel delivered by an electric low pressure pump. It is directly propelled by the crankshaft of the engine. A valve in the pump controls how much fuel is delivered with each stroke. The rail distributes the compressed fuel to the injection valves. Depending on the injection timings, a variable amount of fuel is finally released into the cylinders. A pressure sensor in the rail provides the current pressure level. The system is controlled by the ECU. Depending on the measured pressure and operating conditions of the engine, which define the rail pressure set-point, the valve in the high pressure pump is set. A pressure relief valve limits the pressure to the maximum value the components allow. It opens a connection between the high- and the low-pressure side of the pump. The relief valve is an emergency device though and not designed for frequent opening. Thus, all the methods presented in the following try to avoid excessive pressure. More information about the HPFS system can be found in [41].

From a system-theoretical perspective, the HPFS system has three inputs and one output signal. The output is the rail pressure. It is affected by the engine speed, the fuel pump actuation and the injection time. To simplify the experimental setting, the vehicle is only operated in standstill and neutral gear. Hence no chassis dynamometer is required. It also allows to omit the injection time as independent variable for modeling and design of experiments. As no external load is connected to the engine, its fuel consumption approximately only depends on the engine speed and its gradient. Furthermore, freely varying the injection times would imply a large risk of component damage. Too much or too little fuel injection would stall the engine, while too little fuel would furthermore increase the exhaust gas temperature and could damage the exhaust aftertreatment components. Thus, a more powerful supervision would be required especially during early tests of the measurement algorithms. To avoid this additional complexity, the injection times are set by the ECU and not varied manually during the experiments.

6.2.2 ODCM

First, Online DoE with constraint modeling introduced in Section 3.2 is applied to the HPFS system. This algorithm was developed before this PhD project started. The measurement results are thus just given for comparison with other methods like SAL.

Figure 6.3 shows how the algorithm explores the input space. At the beginning, 25 space-filling points were planned. They are measured one after another with increasing distance to a safe starting point, which is at $[1500\,\mathrm{min}^{-1}, 30\,\mathrm{mm}^3]$. As soon as the first infeasible point has been discovered, the classifier is trained and used to predict the feasibility of future queries, see the lower left plot. They are skipped if predicted as infeasible, compare the lower right plot. 3 of the 25 planned points were measured as infeasible, 3 other points were skipped. All of the skipped points would indeed have been infeasible.

To evaluate the quality of the points' distribution and the classifier and compare them to other methods, numerical measures are desired. The first criterion is assessed by learning a regression model from the obtained data and judging its performance on test data. Therefore, the GP modeling algorithm used as regression model in stationary SAL was utilized. One model is trained using the data available after each step of the ODCM algorithm. It predicts the output of $m_* = 100$ feasible space-filling

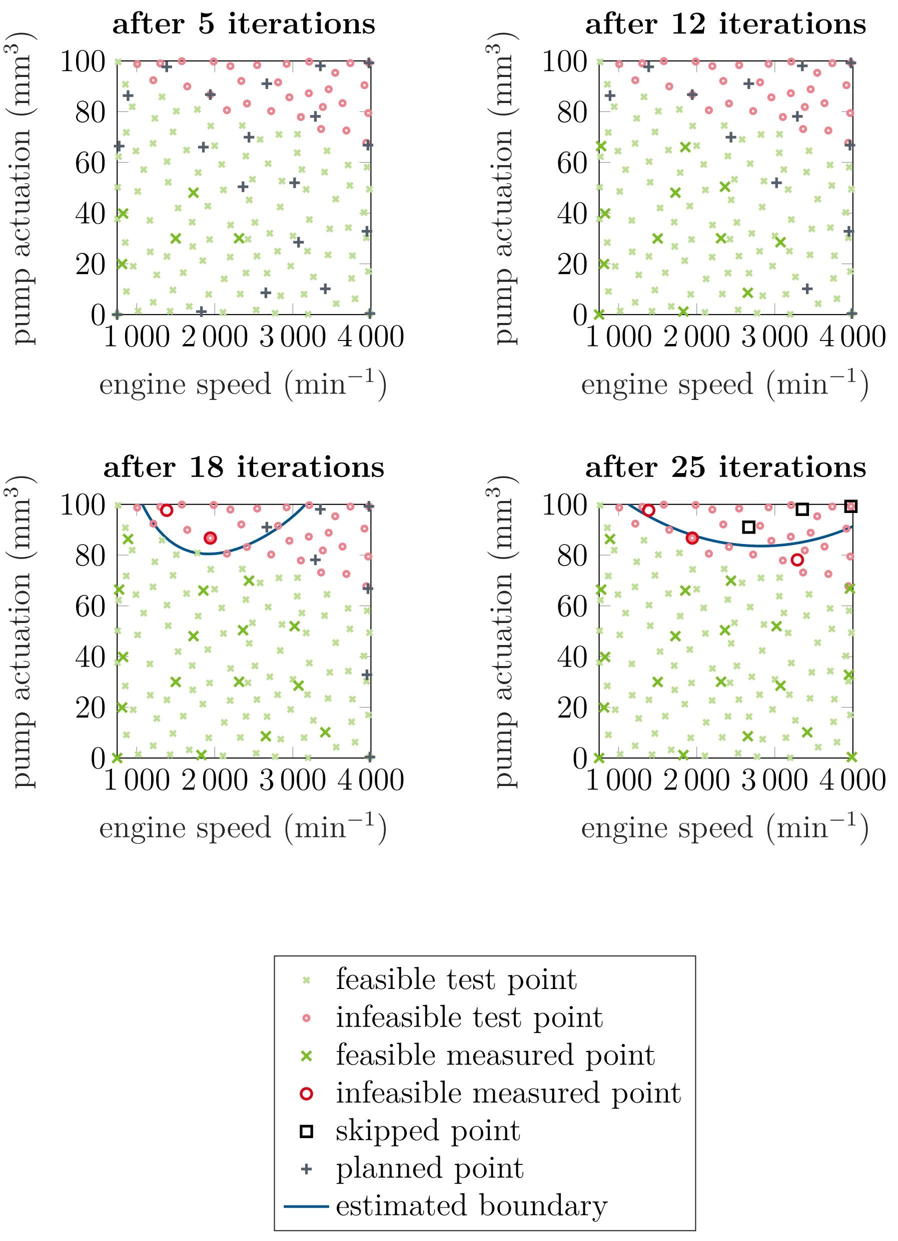

Figure 6.3: Point distribution in the input space with ODCM at the HPFS system. The small crosses and circles in the background are test points. They are plotted to indicate the real boundary and used to calculate sensitivity, specificity, and NRMSE.

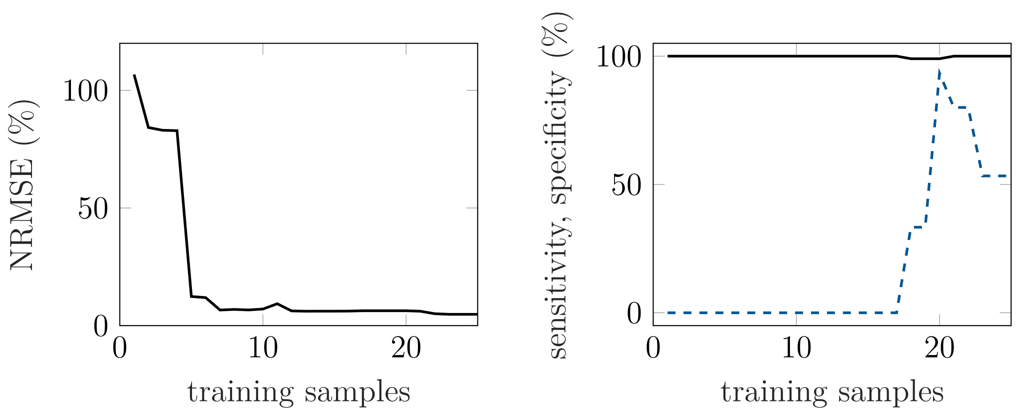

Figure 6.4: NRMSE of the regression model (left plot), sensitivity (right plot, solid line), and specificity (right plot, dashed line) achieved with ODCM plotted over the number of available training points. The NRMSE was calculated on 100 feasible test points, sensitivity and specificity were derived using 130 feasible and infeasible test points.

test-points. By comparing the predictions with the measured outputs the normalized root mean square error (NRMSE)

$$\text{NRMSE} = \frac{1}{\overline{y}} \sqrt{\frac{1}{m_*} \sum_{i=1}^{m_*} (y_{*,i} - y_i)^2} \tag{6.1}$$

can be calculated. Thereby, $\overline{y}$ is the mean of all measured outputs, $y_{*,i}$ the predicted, and y_i the measured output of test sample i. The NRMSE is shown in the left plot of Figure 6.4. After 25 samples a NRMSE of 4.8 % is achieved.

The performance of the classifier is assessed using two figures: sensitivity and specificity. They describe the fraction of correctly classified feasible or infeasible test points, respectively, i.e.

$$\text{sensitivity} = \frac{\text{number of points correctly classified as feasible}}{\text{number of all feasible test points}}, \tag{6.2}$$

$$\text{specificity} = \frac{\text{number of points correctly classified as infeasible}}{\text{number of all infeasible test points}}. \tag{6.3}$$

As the right plot in Figure 6.4 shows, the sensitivity remains almost all the time at 100 % and never falls below 99 %. This reflects the tuning of ODCM's classifier: in doubt it should rather try to measure an infeasible sample than skip a feasible one. A rather extreme effect of this tuning can be seen in the lower right plot of Figure 6.3.

Even one of the infeasible training samples is predicted to be feasible. The downside of this tuning is a decreased specificity. It starts at zero and does not rise until the first two infeasible samples have been measured in step 15 and 18. After achieving a maximum of 93 % in step 20, the specificity decreases again. This is mainly due to the feasible point sampled at [3962 min^{-1}, 67 mm^3], just in front of the real boundary. It pushes the estimated boundary too far into the infeasible area and thereby leads to false positive predictions.

6.2.3 ODCM with Improved Boundary Estimation

The first enhancement of ODCM presented in Section 3.3 provides an improved estimation of the boundary between the feasible and the infeasible parts of the input space. Therefore, additional points are queried near the decision boundary of the classifier. The method is evaluated at the same HPFS system subsequently to the ODCM measurement presented above.

Figure 6.5 shows how the algorithm distributes the additional points in the input space. In the left plot, 3 extra points were measured, corresponding to the 3 points skipped earlier. This means that 25 points in total are available for model training. Thus the total measurement time is identical to a naïve measurement where no points are skipped. In the right plot, 13 points near the boundary were sampled, resulting in 35 measured points available for model training and 38 queries including the skipped points. The queries are no more required to have a predicted feasibility of at least 0.4, as within ODCM, but are generated more aggressively to have a predicted feasibility between only 0.3 and 0.55. Consequently, the number of infeasible points increases. 7 out of the 13 generated points are infeasible, the remaining 6 are feasible. As the figure shows, all but one of them are close to the true system boundary. Thus, they are less critical (only minor boundary violations) than samples far away from the boundary in the infeasible subspace.

NRMSE, sensitivity, and specificity are plotted in Figure 6.6. As expected due to the distribution of the sampled points only near the boundary, the NRMSE does not benefit significantly from the additional points. Note the different scaling of the y-axis compared to Figure 6.4. The sensitivity remains at 100 % for the whole time. The specificity can profit from the additional points. Nonetheless it does not rise continuously and cannot exceed its maximum value during the ODCM run. Note

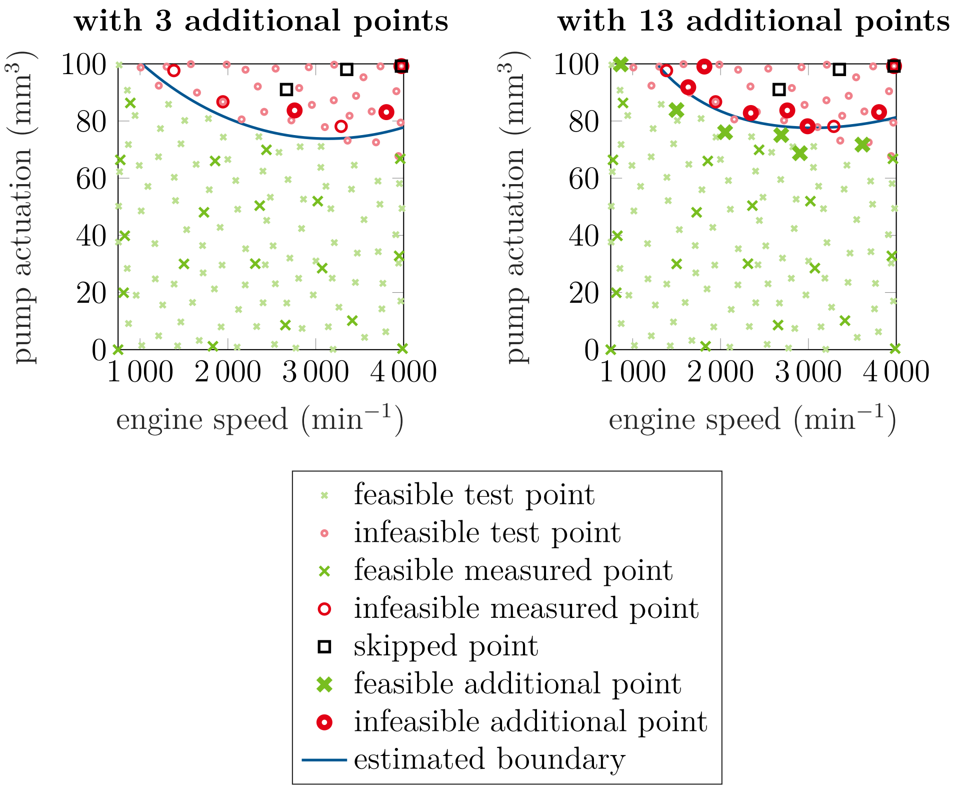

Figure 6.5: Point distribution in the input space with ODCM and subsequent improved boundary estimation at the HPFS system. Additional samples near the boundary are highlighted in bold type.

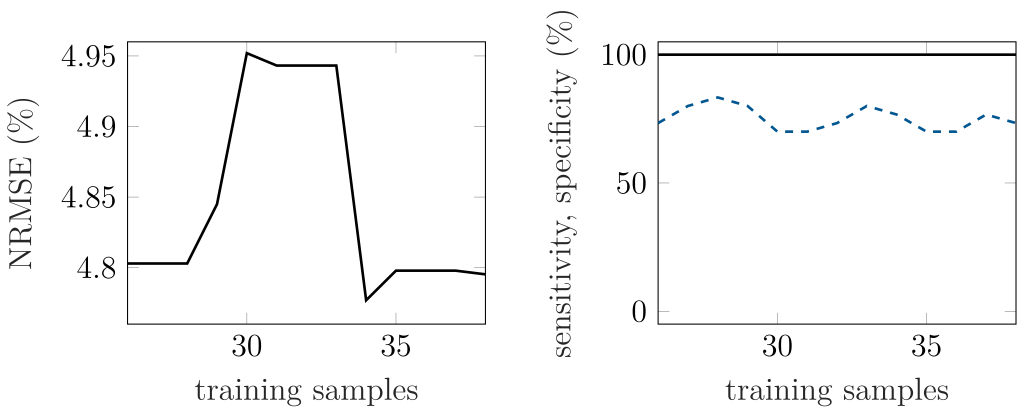

Figure 6.6: NRMSE (left plot), sensitivity (right plot, solid line), and specificity (right plot, dashed line) achieved by ODCM with improved boundary estimation plotted over the number of available training points.

that even a very good ODCM classifier can hardly reach 100 % specificity due to its bias: The decision boundary is not set to 0.5 but to 0.4, i.e. closer to the infeasible label 0 than to the feasible label 1. This ensures a high sensitivity but is a burden on the specificity.

The oscillation in the specificity reflects the rather large influence of new measurement points on the estimated boundary: If an infeasible point is measured, the boundary is pushed towards the feasible input space and the specificity raises. The probability that the next selected point is feasible also increases. If a feasible point is measured this mechanism is reversed: the boundary is pushed towards the infeasible subspace, the specificity is reduced and it becomes more likely that future selected queries are infeasible. With increasing number of points the influence of a single sample is reduced and the amplitude of the specificity-oscillation decreases.

ODCM with improved boundary estimation can even be used without initial ODCM measurements, as Figure 6.7 shows. In this case, extreme queries at the edge of the input space are likely already in early steps, though. This can set the system under test at risk and makes the algorithm only applicable to systems with proper supervision. As visible in Figure 6.8, the NRMSE achieves an equally good level of 4.5 % after all 36 measurement points. This is surprising, as the algorithm only focuses on boundary estimation. Obviously, the points which are spread in the whole input space when the classifier is still uncertain are sufficient to obtain a good model of the HPFS system. In this early phase, the area between a predicted classifier output of 0.3 and 0.55 is quite large. Within this area the point with maximum nearest neighbor distance to all previous points is queried. This results in a point distribution similar to a space-filling maximin design, compare Section 3.1.

The specificity raises quicker than with ODCM, but shows multiple collapses in between. Two of these collapses are also visible in Figure 6.7 after 14 and 21 iterations. Here, the classifier becomes uncertain and predicts the whole input space as feasible. The final specificity of 80 % is comparable to the previous variant. The bad quality of the classifier especially in the beginning lacking a high number of points in the feasible input space can lead to a decreased sensitivity. Compared to the original ODCM this is not that bad though, as new points are created de novo and therefore no false skips can occur.

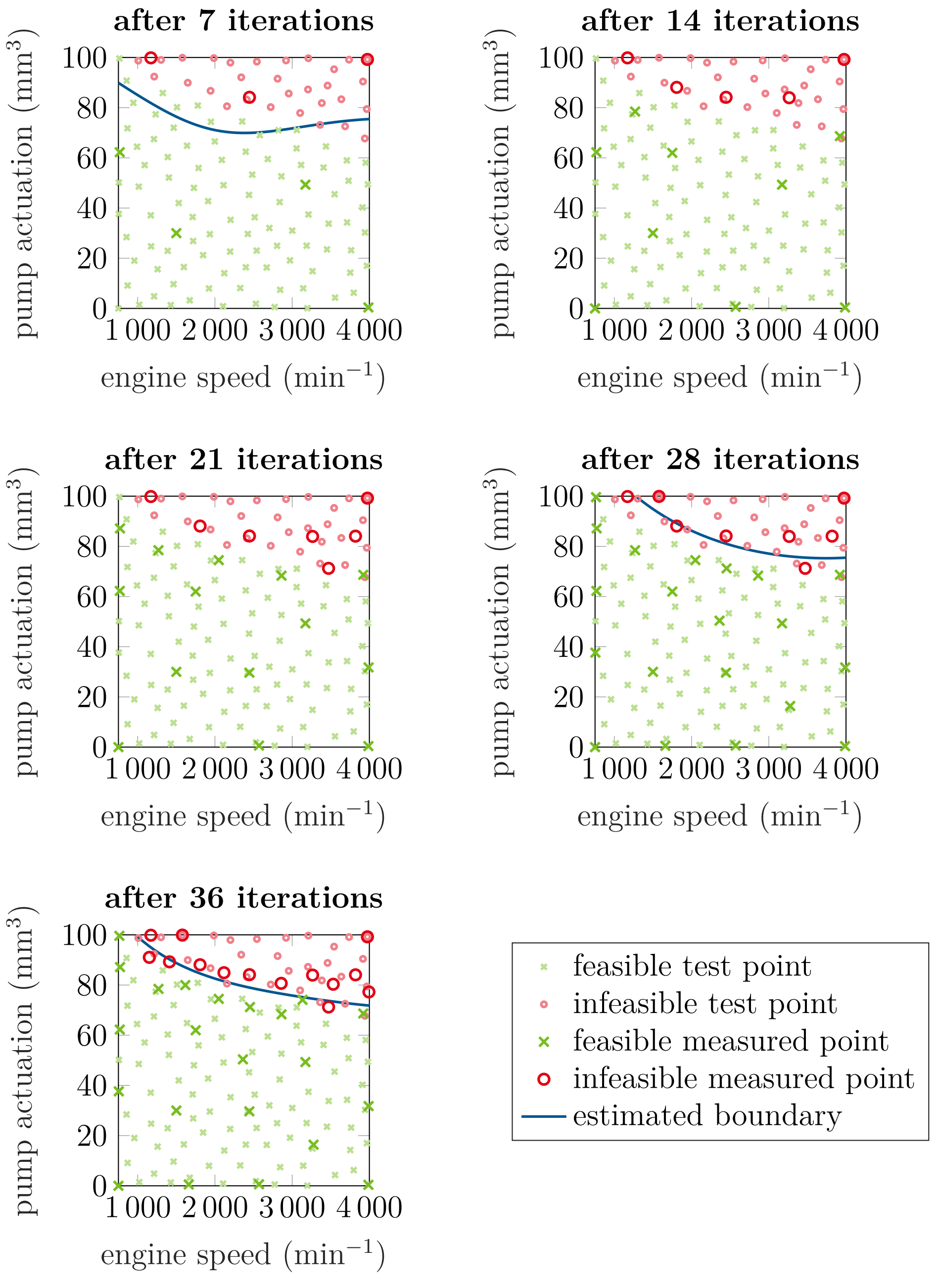

Figure 6.7: Point distribution in the input space with the improved boundary estimation algorithm without previous ODCM measurement at the HPFS system.

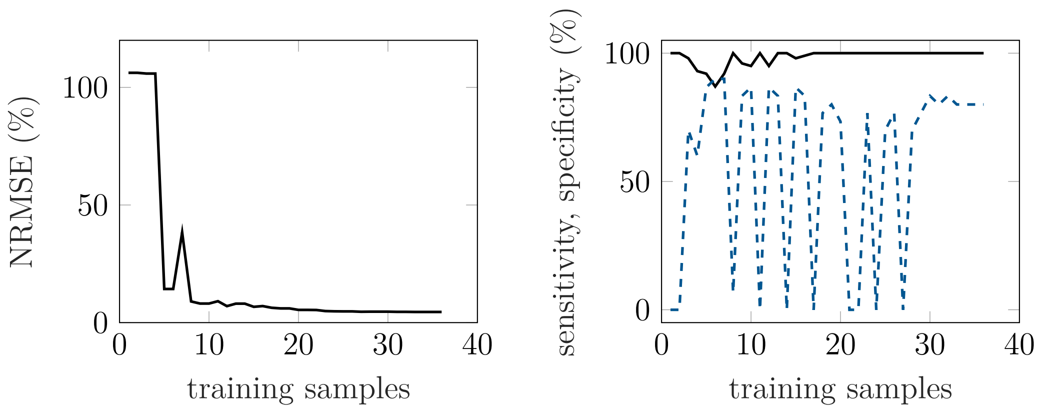

Figure 6.8: NRMSE (left plot), sensitivity (right plot, solid line), and specificity (right plot, dashed line) achieved by the improved boundary estimation without initial ODCM plotted over the number of available training points.

Table 6.1: Comparison of SAL with pre- and online-learned hyperparameters in simulation after 25 samples. The presented results were averaged over 10 simulation runs.

	pre-learned			online-learned
Initial points	5	12	20	0
NRMSE	4.51 %	0.29 %	1.03 %	0.30 %
Sensitivity	78.45 %	99.28 %	97.85 %	99.41 %
Specificity	96.47 %	100 %	100 %	100 %
Samples in $\mathbb{X}_-$	14.7	1.2	0.1	0.8

6.2.4 Stationary Safe Active Learning

The next method evaluated at the HPFS system is stationary safe active learning (SAL). The method is described in Section 5.1. As already mentioned there, the selection or training of correct hyperparameters is crucial to ensure a safe exploration of the input space. Thus, initial experiments were conducted using a simulation model of the HPFS system, to compare different strategies for hyperparameter selection (compare also Section 5.1.1). These simulation results were already published in [47]. Afterwards, the method was applied to the real-world system. The risk-function in all simulation and real-world experiments was defined according to (5.5). Here, z is the measured rail pressure, $z_{\min} = 0\,\text{MPa}$ a rough estimation of the minimal possible pressure, and $z_{\max} = 25\,\text{MPa}$ the maximal allowed pressure.

A stationary GP regression model with additive zero-mean Gaussian noise learned from ODCM data was used as reference in the simulations. The results of SAL were benchmarked on a set of 10 000 space-filling test points. Table 6.1 shows some results of the simulations. Four cases with either pre- or online-learned hyperparameters were compared. In case of pre-learned hyperparameters, 5 to 20 initial points were sampled in an area covering approximately 20 % of the feasible input space $\mathbb{X}_+$. This space was known to be safe by expert knowledge. The initial points were used to optimize the hyperparameters by maximizing the model's marginal likelihood. In case of online learned hyperparameters, the parameters were optimized during the measurement without using previously measured points. For the initial parameters, the heuristic introduced in Section 5.1.1 was used. To avoid an unsafe fast exploration, the length-scales of the discriminative model were limited. Thereby, areas far away from already measured samples are not classified as safe after too few samples. The limits are gradually released with increasing number of samples. Beginning with fixed length-scales $\boldsymbol{\lambda}_g = \frac{\Delta \boldsymbol{x}}{4}$ for the first 5 samples, they are optimized afterwards. Until 10 samples were recorded, the length-scales are not allowed to grow above $\boldsymbol{\lambda}_g = \frac{\Delta \boldsymbol{x}}{4}$. Until 20 points were measured, this limit is raised to $\boldsymbol{\lambda}_g = \frac{\Delta \boldsymbol{x}}{2}$. Starting from 20 points, an unconstrained hyperparameter optimization is pursued. Thereby, $\Delta \boldsymbol{x}$ is the extent of the input space $\mathbb{X}$.

The hyperparameters of the regression model are less safety-critical. If they are estimated wrongly, the resulting density of the samples might be unfortunate. This can be corrected later by inserting additional queries, though. The parameters of the regression model were fixed to $\boldsymbol{\lambda}_f = \frac{\Delta \boldsymbol{x}}{4}$, $\sigma_f^2 = \left(\frac{\Delta y}{2}\right)^2$, and $\sigma^2 = \left(\frac{\Delta y}{200}\right)^2$ during the first 5 steps of the algorithm. Variable Δy thereby denotes the expected range of the output signal y. These initial hyperparameters follow the same heuristic as the discriminative model's hyperparameters, compare Section 5.1.1. Additionally, they consider the different output range of the output y compared to the measured discriminative function values z. Starting with the 6th step of the algorithm, the regression model's hyperparameters are optimized without constraints. The confidence parameter ν was set to 2.33 which is equivalent to a probability of feasibility of 99 % throughout all experiments.

In total, 25 points were queried, which is sufficient to obtain a good model of the HPFS system if the points are distributed properly. As Table 6.1 reveals, 5 initial points are not enough for a proper hyperparameter estimation. The remaining points are distributed unfavorable resulting in a very high NRMSE and a large number of infeasible queries. On the other hand, 20 initial points do lead to the smallest number of infeasible queries. The remaining 5 points for SAL until the termination criteria is fulfilled are not enough to achieve a good model with low NRMSE and a proper estimation of the boundary, though. The latter results in a low sensitivity and specificity. With 12 points, an optimal compromise is achieved: The number of infeasible queries is low, the best NRMSE is achieved, sensitivity as well as specificity are very good. Thus, predetermined hyperparameters learned from initial points can result in a good model and a safe exploration if the correct number of initial points is chosen. A wrong choice will result in a bad model or an unsafe exploration, though.

This critical choice can be avoided by using an online estimation of the hyperparamters. It obtains the best boundary estimation in terms of sensitivity and specificity. The model quality is also comparable to the pre-learned case with 12 points, whereas the number of infeasible samples is even lower. Thus, online learned hyperparameters are also used in the subsequent experiments at a test vehicle.

The distribution of measurement points in this case is shown in Figure 6.9. It is clearly visible how the algorithm gradually explores the input space. Opposed to ODCM (compare Figure 6.3), the estimated boundary starts with a very small extending feasible input space and grows over time. Due to this approach, not a single infeasible point was queried by SAL. On average over two experiments, only 0.5 infeasible samples occur. The different exploring behavior of ODCM and SAL is also visible when comparing SAL's sensitivity and specificity in Figure 6.10 with ODCM's characteristics in Figure 6.4: In case of SAL, the sensitivity starts just above zero and raises continuously up to 95 %. The specificity remains close to 100 % all the time and never drops below 96 %. This behavior is antithetic to ODCM. The regression model's quality cannot compete with ODCM, though. While ODCM achieves a NRMSE of 4.8 %, SAL terminates at 6.1 %.

To summarize the results: SAL allows to explore the input space with much less limit violations compared to ODCM. SAL is therefore more safety-oriented. The discriminative model describes the boundary better than ODCM's classifier. On the downside, the model quality achieved by SAL is worse than ODCM's.

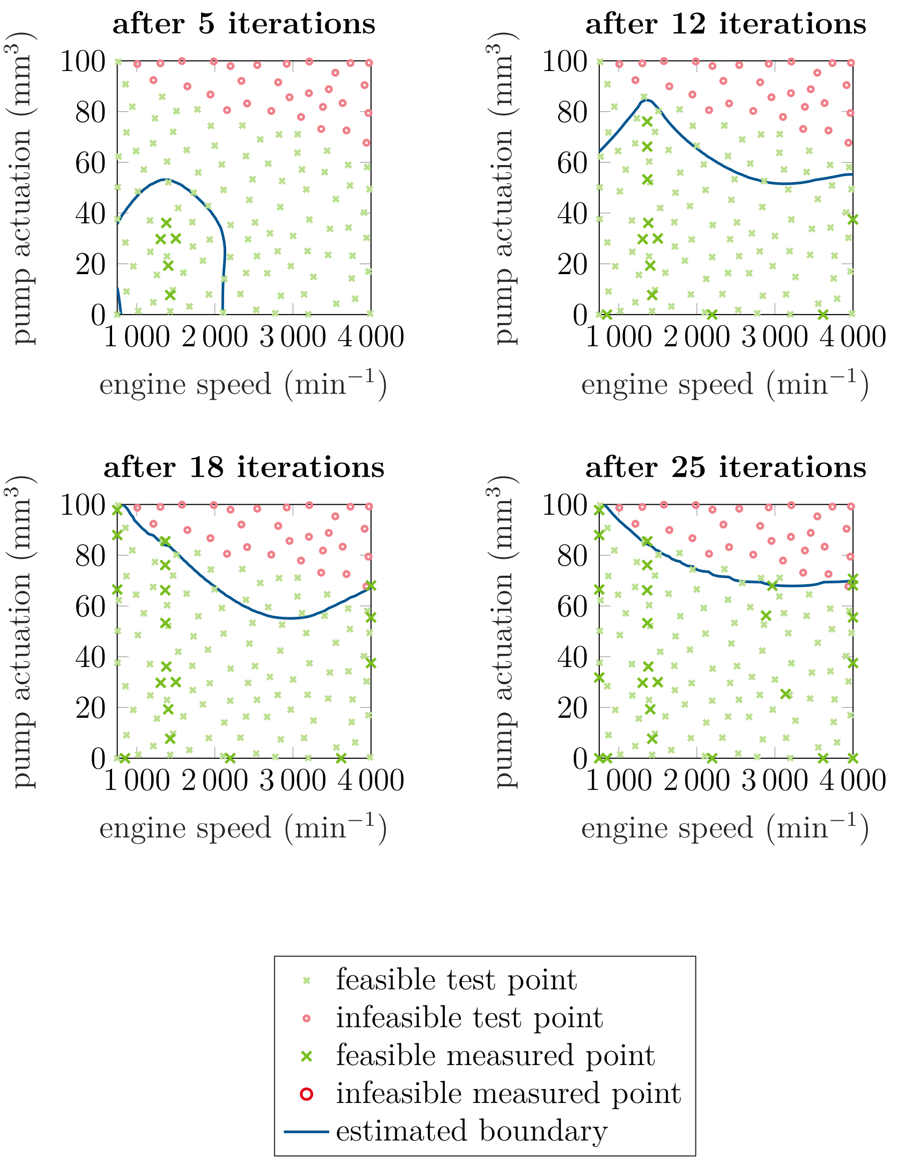

Figure 6.9: Point distribution in the input space with stationary SAL at the HPFS system.

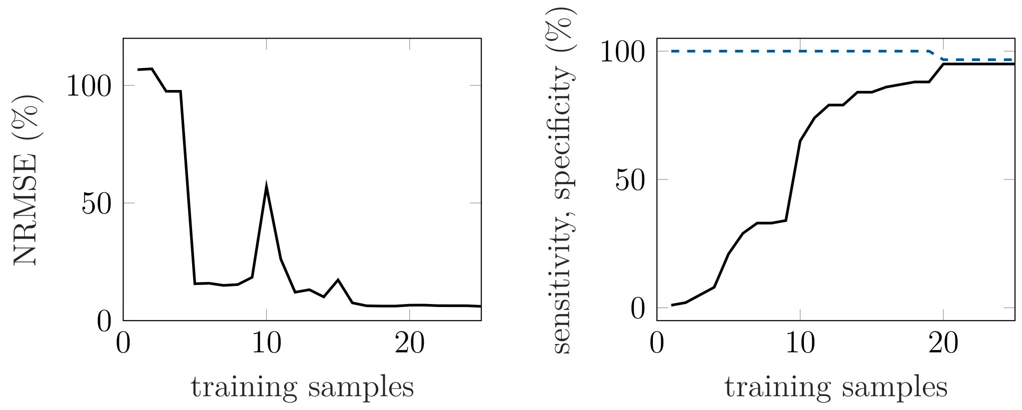

Figure 6.10: NRMSE (left plot), sensitivity (right plot, solid line), and specificity (right plot, dashed line) achieved by stationary SAL plotted over the number of available training points.

6.2.5 ODCM with Discriminative Model

Based on the comparison of ODCM and SAL, a new combination of both approaches was proposed in Section 3.4. It combines the plan-based sampling of ODCM with the discriminative model of SAL. Here, the algorithm is evaluated on the data recorded using ODCM in Section 6.2.2. The same online hyperparameter optimization is used for the discriminative model as in the previous SAL-section. The only difference in the parameter tuning is the confidence parameter ν. It was decreased from 2.33 to zero, which corresponds to a decrease in the requested probability of feasibility from 99 % to 50 %. This drastic drop is necessary to avoid falsely skipped points. If ν is chosen too large, all planned queries might be outside the estimated boundary during the early steps because the classifier is not certain enough, stalling the algorithm. This does not happen in plain SAL, as the optimization of the next query is continuous and simply a point closer to the previous feasible samples would be chosen.

The distribution of the input points and the estimated boundary is shown in Figure 6.11. As one can see, the boundary does not expand as it did with SAL. This is due to the decreased parameter ν. Opposed to ODCM, it can be estimated without measuring any infeasible points, because the algorithm extrapolates the position of the boundary from the measured discriminative function values. One point, highlighted by a gray ellipse in Figure 6.11, was falsely skipped. This point is very close to the real boundary, making it a tough candidate. Fortunately, the achieved NRMSE is not severely affected by this failure, as Figure 6.12 shows: after 25 queries, a NRMSE

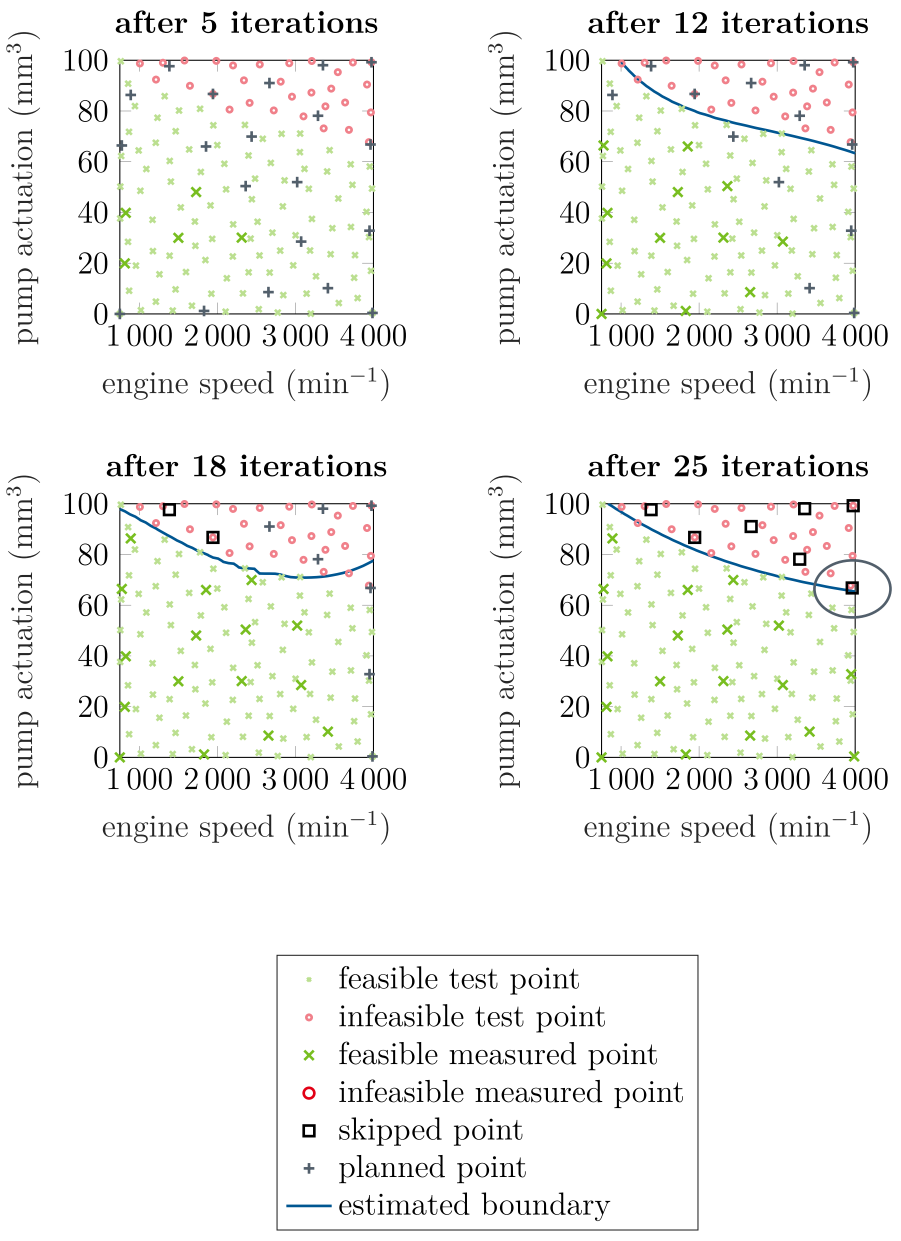

Figure 6.11: Point distribution in the input space using ODCM with discriminative risk model at the HPFS system. The ellipse in the lower right plots highlights a falsely skipped point, compare Figure 6.3.

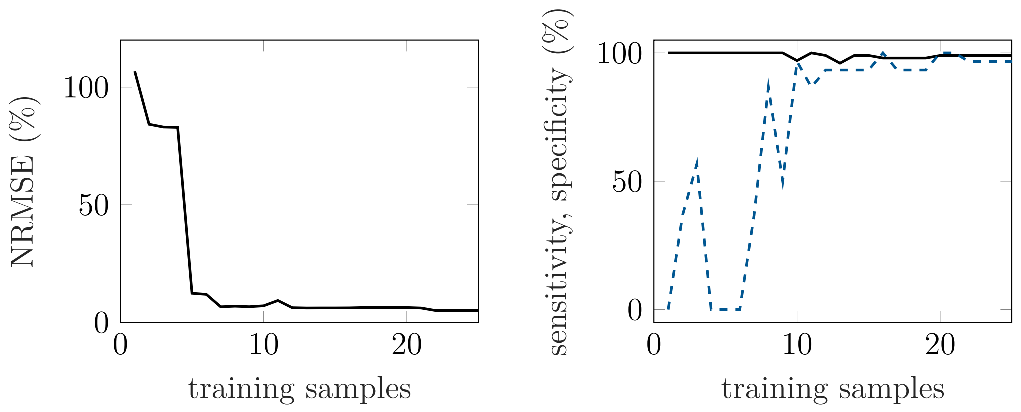

Figure 6.12: NRMSE (left plot), sensitivity (right plot, solid line), and specificity (right plot, dashed line) achieved by ODCM with discriminative risk model plotted over the number of available training points.

of 5.1 % is calculated, which is just slightly worse than ODCM's result. The changed behavior due to the decreased confidence parameter is also visible in sensitivity and specificity. Their progress is more similar to ODCM than to SAL. In the end, the best result of all methods compared in this thesis is achieved: A sensitivity of 99.0 % and a simultaneous specificity of 96.7 %.

To conclude: the combination of ODCM and the discriminative model showed promising results at the HPFS system. It outperforms the already good boundary estimation of SAL while almost achieving the model quality of ODCM. Practical benefits like customizable DoEs are further advantages of this algorithm, compare Section 3.4.

6.2.6 Safe Active Learning and Safe Bayesian Optimization for Control

In this section, the SAL and SBO algorithms for the special application of controller tuning are evaluated at the HPFS system. They were introduced in Section 5.2. The following results including the figures were already published in [46].

In normal operation, the rail pressure is controlled to a varying setpoint by the ECU. The setpoint changes based on the current operating conditions of the engine. A PI-controller with gain-scheduling in combination with a feed-forward model is used to fulfill the control task. This controller has a lot of parameters, as it is gain

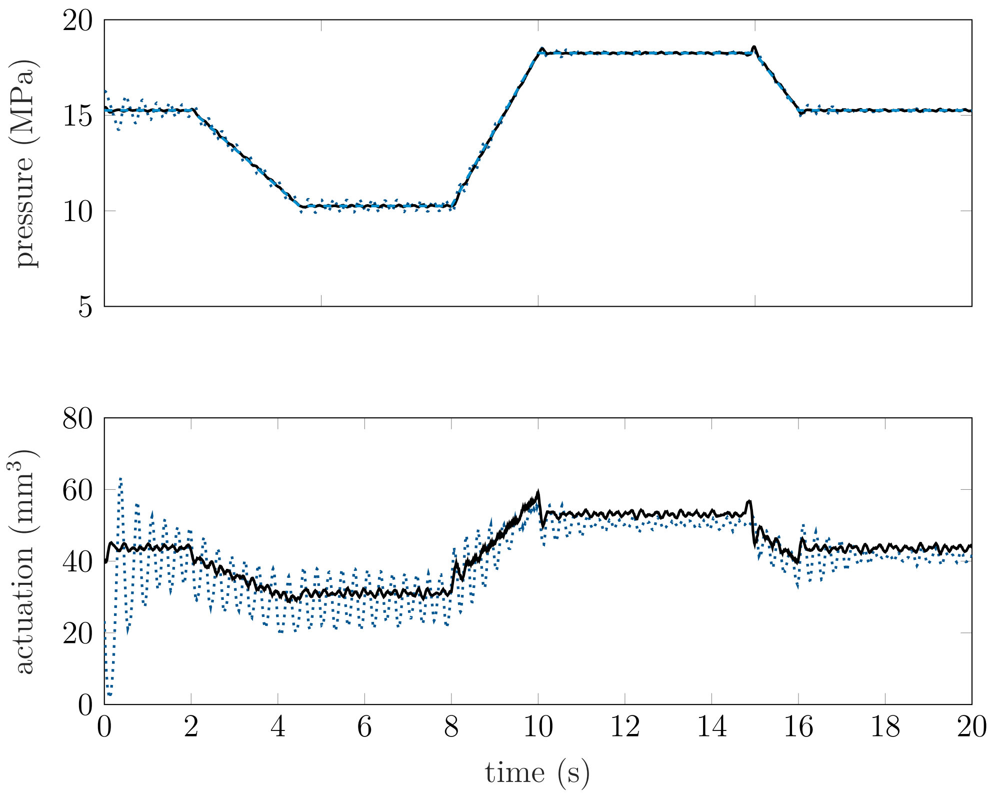

Figure 6.13: Plant output (top) and manipulated variable (bottom) of the controlled HPFS system. The reference is plotted as dashed line in the upper plot. The solid line denotes the best controller found using SBO, whereas the dotted line, for comparison, shows a bad controller being hardly feasible.

scheduled over several signals like engine speed, fuel temperature, and engine load and has additional features like an online-adapted stationary offset. As this evaluation of SAL and SBO for controller tuning is a proof of concept, a simpler controller structure was chosen: a PI controller with only two parameters. Furthermore, only a constant engine speed and no load operation are considered to avoid the necessity of an expensive chassis dynamometer. Tuning more complex controllers is left for future research.

The reference trajectory was designed based on measurements at the calibrated vehicle. It is similar to reference trajectories seen in normal operation. See Figure 6.13 for a plot of the selected trajectory.

The hyperparameters of the GP models were kept constant during the whole experiments. They were chosen similar to the heuristics presented in Section 5.1.1 and 6.2.4. Diverging from the original heuristic, $\sigma_g^2 = 1.5$ was set, as it showed better results during the experiments. Furthermore, the prior mean value of the regression model was set to $\mu_{f,0} = 0.5$ and the mean of the discriminative model to $\mu_{g,0} = -1$. The former improves the modeling quality, as the new value corresponds to the mean of the loss function's range. The latter yields a more conservative exploration. In (5.18) a minimum value for SBO's parameter κ was motivated. For this application the motivation results in $\kappa > 0.56$. In simulation experiments this turned out as a lower bound for good exploration. For the experiments at the test vehicle, $\kappa = 3$ was chosen. The other parameters $\Delta y, J_{\max}, \nu, \theta_{\mathrm{e}}$, and θ_{u} were optimized heuristically in simulations to obtain a small number of limit violations while finding a good optimum.

The distribution of the queried controller parameters is shown in Figure 6.14 for SAL and in Figure 6.15 for SBO. The different strategies of the methods are clearly visible. While SAL yields a rather uniform distribution and explores most of the feasible input space, SBO also starts with an exploration, but later on focuses on exploiting the minimum of the loss function. This leads to an accumulation of samples near the expected optimum. The safety criterion works well: both methods sampled only 2 to 3 infeasible points out of 50 queries. All of them are close to the boundary and thus not very critical.

These observations are confirmed by the sensitivity and specificity plots in Figure 6.16. The sensitivity starts near zero, similar to the SAL application presented in Section 6.2.4, and rises almost monotonically. In the end, SAL achieves a larger sensitivity than SBO, which means that the feasible input space was explored better and less feasible test points are falsely excluded. The specificity of both methods remains close to 100 % all the time, which means that infeasible points are avoided with high probability. Note the scaling of the y-axis in the specificity plot. Sensitivity and specificity were calculated based on 100 space-filling test points which are also thinly marked in Figure 6.14 and 6.15.

Optimization results using both approaches are shown in Figure 6.17. To obtain this plot, offline optimizations based on the regression model learned with the named number of samples were conducted. Thereby the only criterion was to minimize the cost function J. The results of the optimization were afterwards evaluated at the real system. As the figure illustrates, SBO achieves a lower and thereby better loss most of the time. The optimized parameters are shown in Figure 6.18 overlaid with a contour plot of the loss function in input space. Controller output and system output

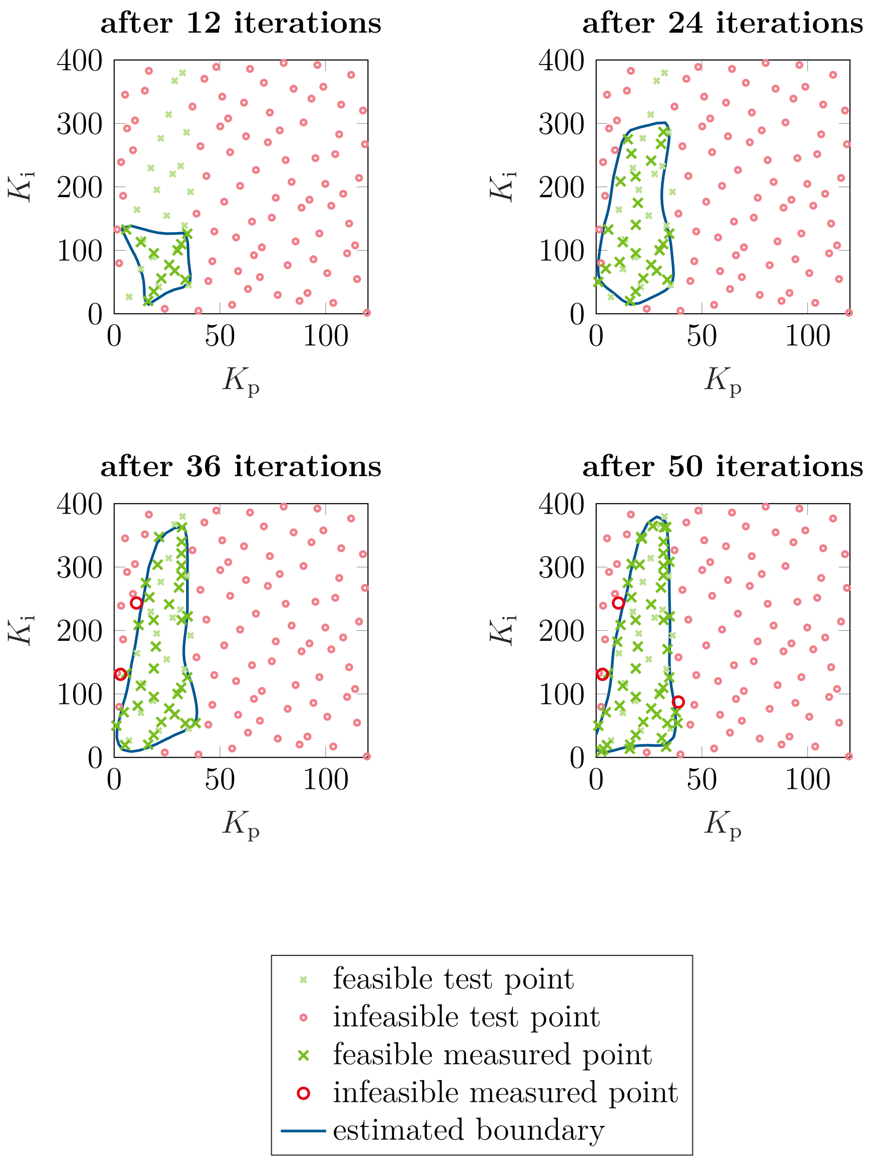

Figure 6.14: Point distribution in the input space using SAL for controller tuning.

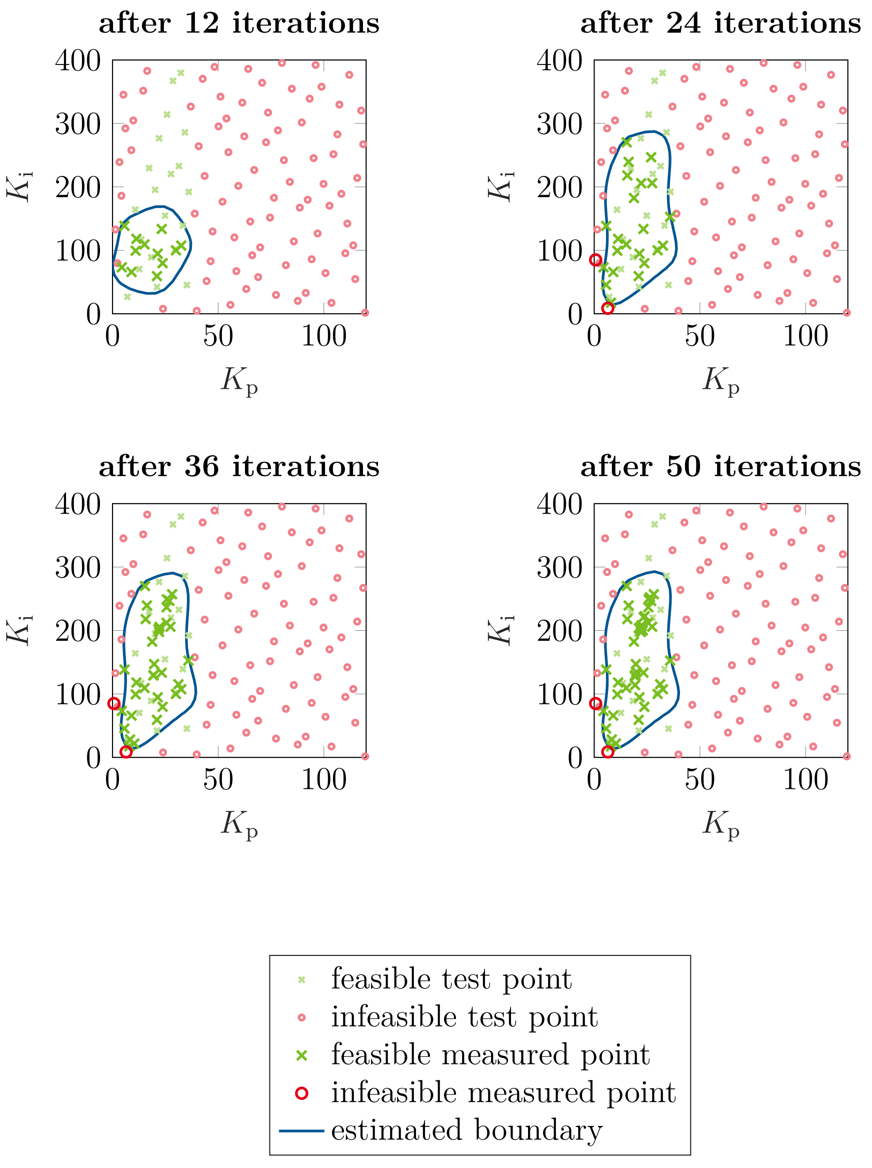

Figure 6.15: Point distribution in the input space using SBO for controller tuning.

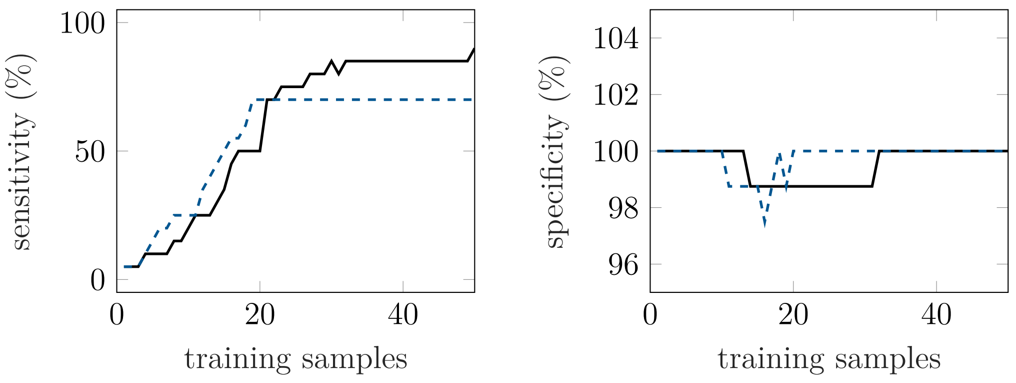

Figure 6.16: Sensitivity (left) and specificity (right) for SAL (solid lines) and SBO (dashed lines) depending on the number of samples used for optimization.

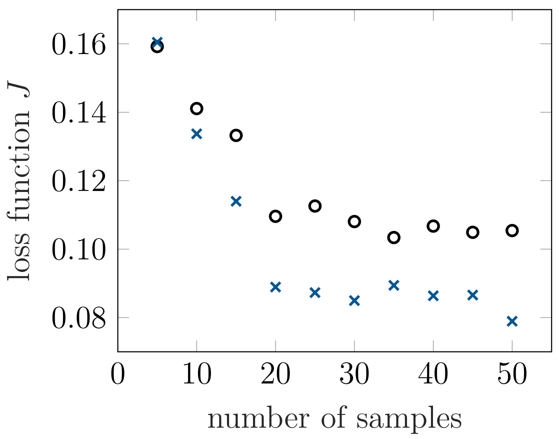

Figure 6.17: Achieved loss function values on real experiments in closed loop with optimized controller parameters depending on the number of measured samples used for optimization. Circles mark the results of SAL, crosses the results of SBO.

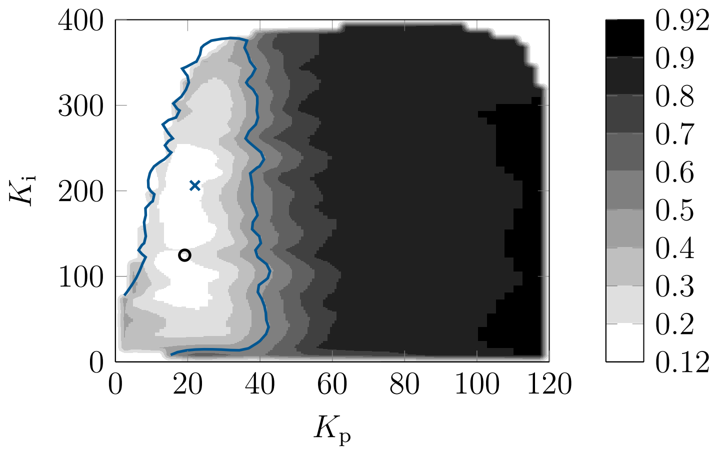

Figure 6.18: Contour of the measured loss function in the input space. The blue line denotes the discriminative function's zero crossing. The optima found by SAL and SBO are marked as circle and cross, respectively.

with the best performing controller are shown in Figure 6.13. For comparison, a bad controller's behavior is also plotted.

6.2.7 Dynamic Supervised Measurement

Until now, stationary measurement methods were evaluated at the HPFS system. In this and the following section, dynamic measurement methodologies are considered, starting with the dynamic supervised measurement algorithm introduced in Section 4.2. For these dynamic measurements, two different DoEs were created. The first consists of ramp signals, the second uses chirp signals. Both were scaled in the stationary boundary from ODCM with improved boundary estimation (see Section 6.2.3) using the algorithm presented in Section 4.1.

The supervising controller limits the rail pressure as manipulated variable by influencing the fuel pump actuation as actuating variable. The latter was selected as it can be changed much faster than the engine speed and has a higher influence on the pressure. A reduced fuel pump actuation yields a reduced pressure. The controller parameters were tuned heuristically in [16]. As supervising model, an ARX model is utilized, compare Section 2.4.2. The model was learned from previously available measurement data. It is evaluated, identical to the complete measurement algorithm, in a 10 ms raster. The ARX model's inputs are measured system input and output

Table 6.2: Thresholds of the rail pressure used in the switching logic. Compare Table 4.1 for a definition of the variables.

variable	value
y_{hardmax}	25.5 MPa
$y_{\max}$	25 MPa
y_{set}	24.5 MPa
$y_{\text{lim,on}}$	24.6 MPa
$y_{\text{lim,off}}$	24.4 MPa
$y_{*,\text{lim,on}}$	24.6 MPa
$y_{*,\text{lim,off}}$	24.4 MPa

values delayed for up to 7 time steps and planned system input values. It predicts the pressure 10 time steps, i.e. 100 ms, ahead. Formally stated this yields

$$y_*(k+10) = \sum_{l=-9}^{4} b_{1,l} x_1(k-l) + \sum_{l=-7}^{7} b_{2,l} x_2(k-l) - \sum_{l=0}^{5} a_l y(k-l) \qquad (6.4)$$

where $x_1(k)$ is the engine speed, $x_2(k)$ the fuel pump actuation, $y(k)$ the measured, and $y_*(k)$ the predicted pressure at time step k. For $k \leq 0$ the inputs are measured values, for $k > 0$ planned values. The model's parameters are denoted as $a_l, b_{1,l}$, and $b_{2,l}, l \in [-9, 7]$. The different summation limits result from the optimization of the model structure. This structure and the model's parameters were optimized in [16] using the Matlab System Identification Toolbox. In a first step, the model was learned for one-step prediction. To predict the output 10 steps in the future, the model has to be evaluated recursively 10 times. In a second step, this recursive calculation was transformed to an explicit calculation by inserting the model's formula in itself multiple times using a computer algebra tool. The obtained explicit representation is computationally more efficient.

The switching limits were chosen heuristically according to Table 6.2. The controller always uses the measured pressure as manipulated variable, never the predicted. This is sufficient, as $y_{\text{lim,on}} > y_{\text{set}}$ and the controller output is limited to the current DoE value. Thus, the controller can never aggravate a limit violation.

To avoid integrator windup when the controller is either in actuator saturation or its output is overruled by a lower DoE value, the integration is interrupted as soon as one of the named conditions occurs. If the DoE is smaller than the controller output, the integrator is allowed to decrease due to a negative control error, though. This feature accounts for the variable upper limit to the controller output, given by the variable DoE values. Bumpless transfer is used when switching on the controller. Thereby, the integrator is initialized such that the controller output is identical to the DoE value in the first time step.

The results of the supervised measurements are shown in Figure 6.19 and 6.20. Note that the plotted engine speed is the measured engine speed, not the planned one. They are not completely identically, as the speed is adjusted by a controller in the ECU. The ramp DoE did hardly lead to any limit violation after scaling to the stationary limits that were previously determined. Thus, the controller was only active during 0.5 % of the measurement time. The chirp signal seems to be more aggressive, requiring the controller to be active during 6.1 % of the measurement time. The controller successfully limits the maximum pressure. The limit is only insignificantly exceeded during the ramp measurement with a maximum of 25.008 MPa. The chirp measurement never exceeds the limit and records a maximum pressure of 24.987 MPa.

In order to evaluate the controller in more challenging conditions, the same ramp DoE was measured again with a maximum allowed pressure of 18 MPa. Controller tuning and prediction model were not altered compared to the previous case. The results are visualized in Figure 6.21. During the measurement the controller was active 17.0 % of the time. Unfortunately, the maximum allowed pressure was still slightly exceeded with a maximum of 18.551 MPa. Note though that without supervision the maximum pressure would have been much higher, as the previous measurement with a measured pressure up to 25 MPa showed. Figure 6.22 presents a corrective action of the controller in detail.

6.2.8 Dynamic Safe Active Learning

The final method evaluated at the HPFS system is dynamic safe active learning. It was introduced in Section 5.3. As described there, the regression model is set up as NFIR structure. The GP model uses engine speed and fuel pump actuation together with their derivatives as inputs. The discriminative model features a NARX approach. In total, 9 features are used for this model: the two physical inputs and the fed back

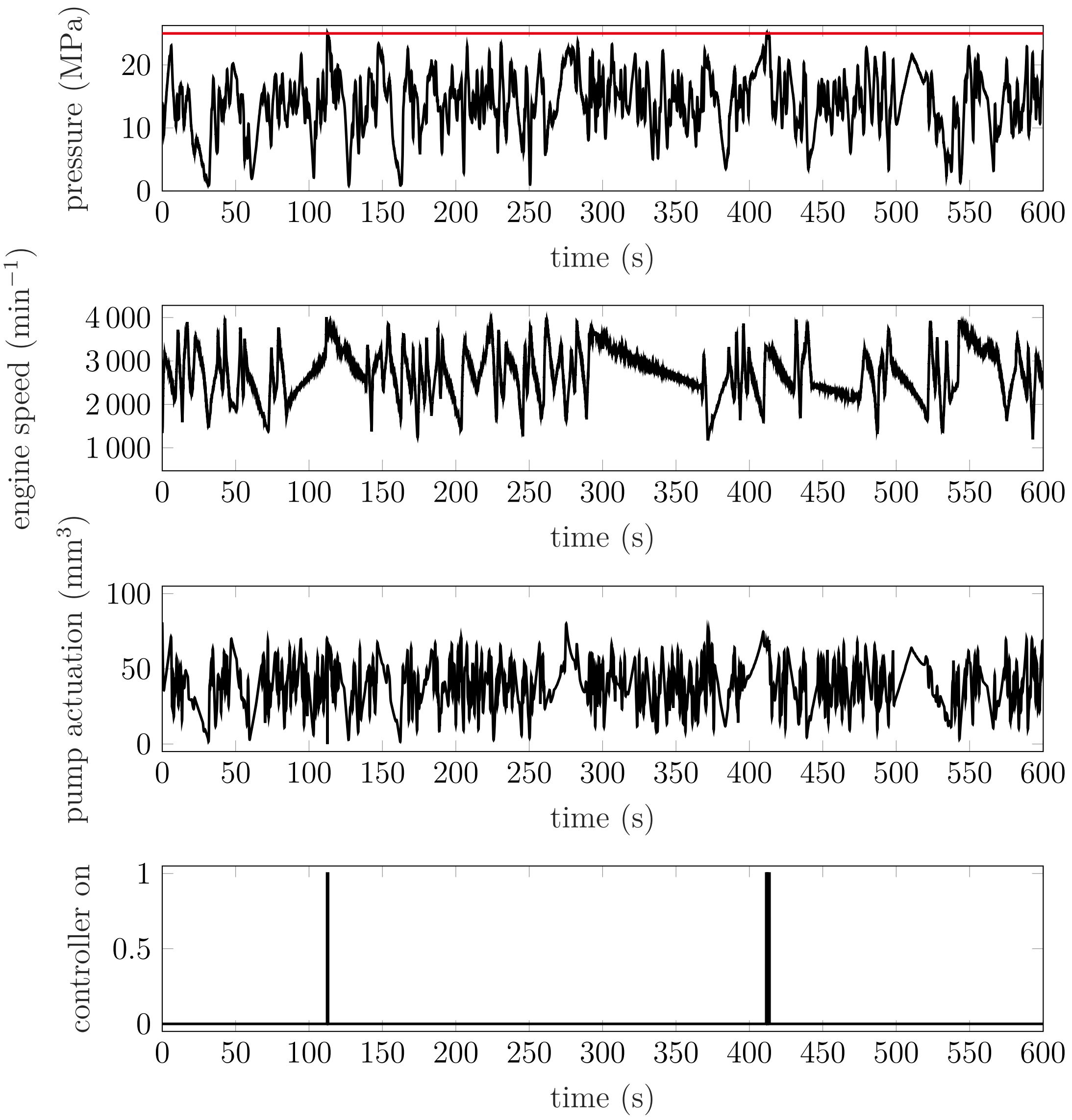

Figure 6.19: Output (top), inputs (middle) and controller state (bottom) for a ramp measurement. The maximum allowed pressure was set to 25 MPa.

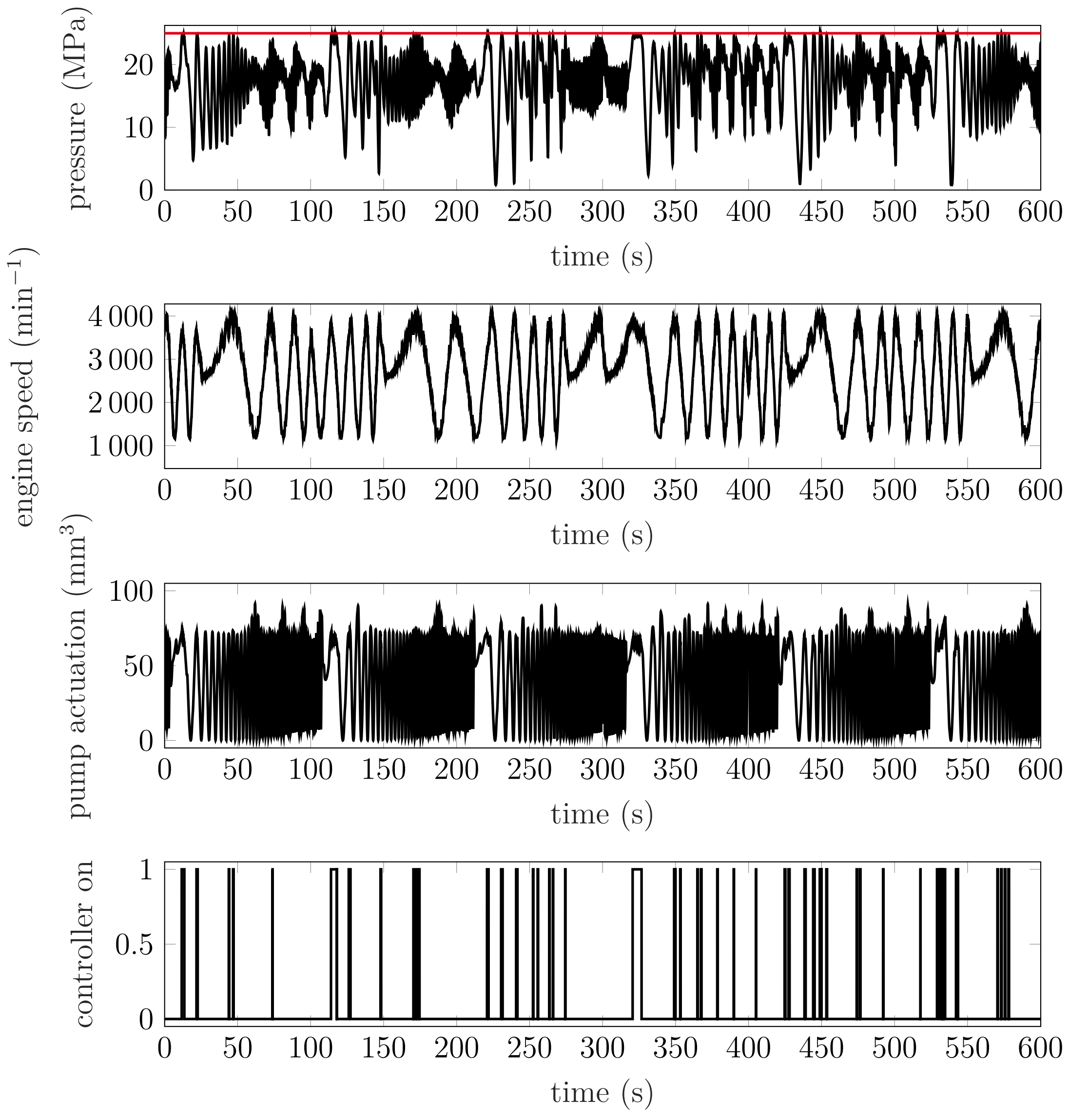

Figure 6.20: Output (top), inputs (middle) and controller state (bottom) for a chirp measurement. The maximum allowed pressure was set to 25 MPa.

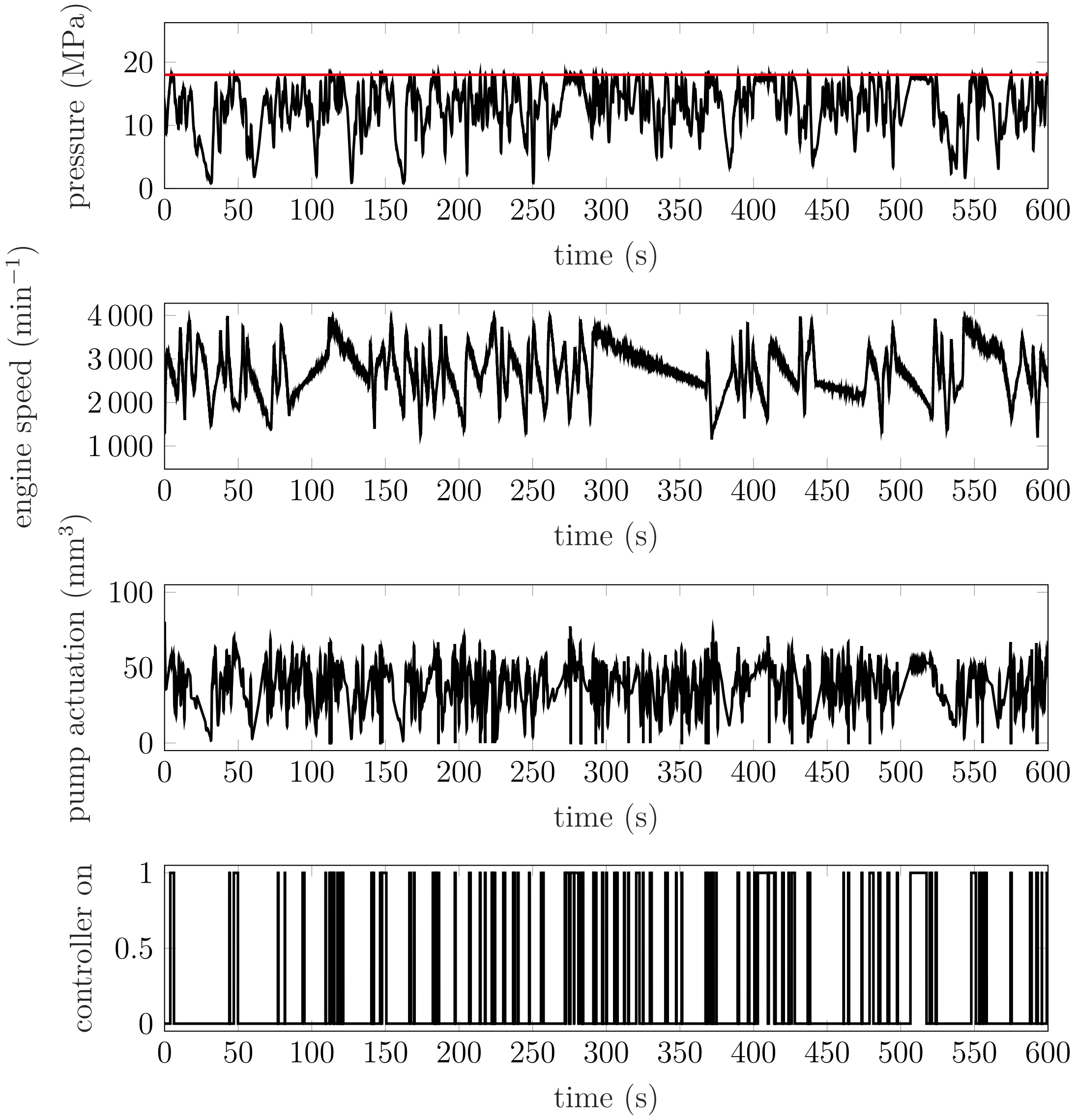

Figure 6.21: Output (top), inputs (middle) and controller state (bottom) for a ramp measurement. The maximum allowed pressure was set to 18 MPa.

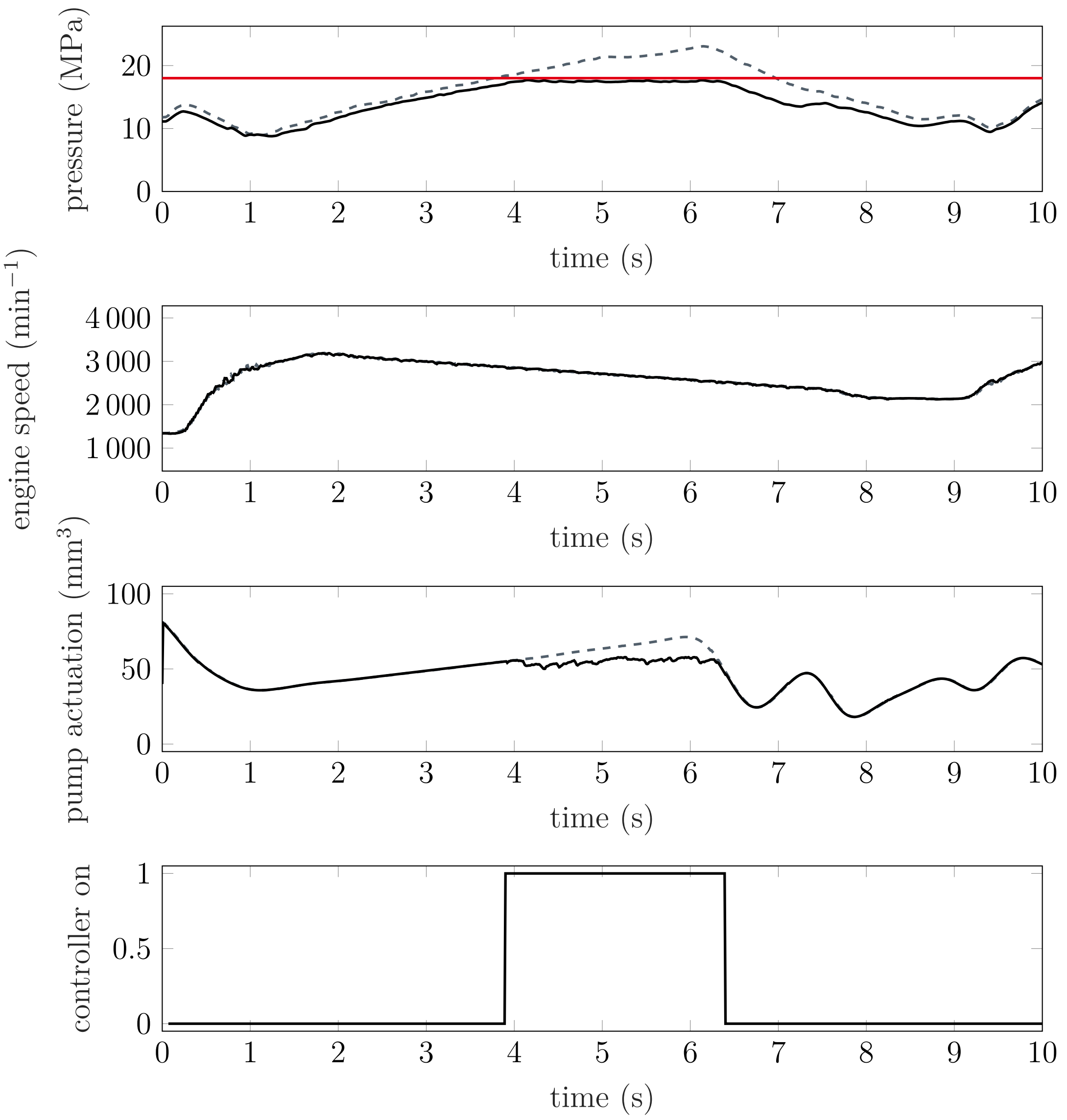

Figure 6.22: Output (top), inputs (middle) and controller state (bottom) showing a corrective action of the supervising controller. For comparison, the measurement with a limit of 25 MPa is overlaid as dashed line. The almost constant offset of the output is most likely related to slightly different temperatures during the measurements.

output delayed 1 to 3 steps each. This structure was found suitable to model the HPFS system in previous experiments. Formally stated this yields

$$y(k) = f_{\mathrm{NL}}\left(\boldsymbol{x}(k), \dot{\boldsymbol{x}}(k)\right) \text{ and} \tag{6.5}$$

$$h(k) = g_{\mathrm{NL}}\left(h(k-1), h(k-2), h(k-3), \boldsymbol{x}(k), \boldsymbol{x}(k-1), \boldsymbol{x}(k-2)\right), \tag{6.6}$$

where f_{NL} denotes the stationary nonlinear part of the regression model and g_{NL} the stationary nonlinear part of the discriminative model. The hyperparameters of the models were optimized on previously available measurement data and kept constant during the experiments. At the beginning of the measurement, 40 initial trajectories were planned in a region of the input space which was known to be safe by expert knowledge. This data is needed to train initial models for the subsequent trajectory optimization. The required safety level for the optimization constraint was set to $p = 0.5$. The sampling time is reduced to 25 Hz compared to 100 Hz which were used during the dynamic supervised measurement in Section 6.2.7. The reduced frequency is a compromise between runtime of the algorithm and modeling quality. The risk function was defined as

$$\tilde{h}(z) = 1 - \exp(\theta_{\mathrm{h}}(z - z_{\max})) \tag{6.7}$$

where z is the supervised output, i.e. the rail pressure. The maximally allowed pressure was reduced to $z_{\max} = 18\,\mathrm{MPa}$ due to safety reasons and the scaling factor was set to $\theta_{\mathrm{h}} = \frac{1}{10}$.

Figure 6.23 shows the input and output signals of a 5 min dynamic SAL run. At the beginning, the initial trajectories with limited amplitudes are visible. After approximately 20 seconds the online-generated trajectories start. Between the ramp segments, sometimes stationary waiting occurs. In these events the optimizer did not finish the planning of the next trajectory fast enough. In total, 64 % of the measurement time were spent waiting for the planning algorithm. If the planning takes too long or if a limit violation occurs, the system is returned to a safe point and a new planning approach is started. Around 6 % of the measurement time were spent at the safe point. The limit of the pressure was kept very well. Only one minor limit violation with a maximum pressure of 18.1 MPa was recorded.

To compare the performance of the regression and discriminative model of dynamic SAL to dynamic supervised ramp and chirp measurements, the models were evaluated

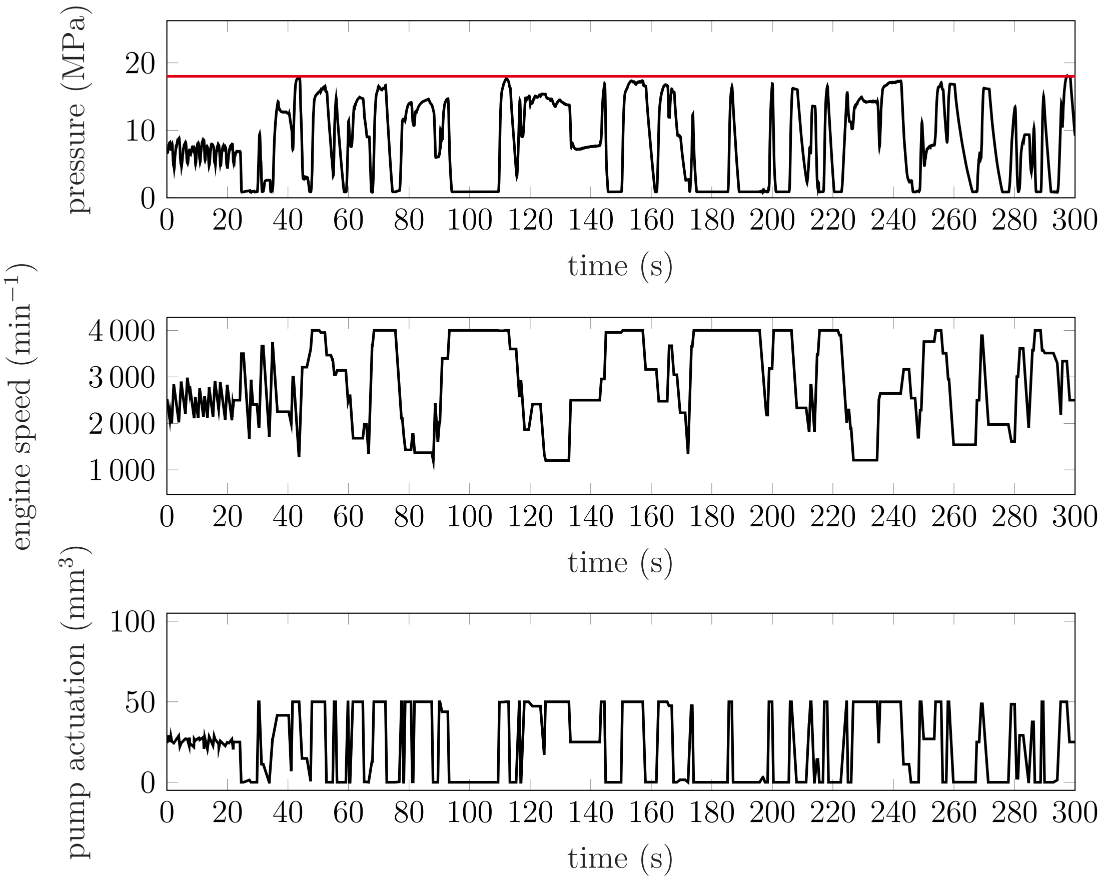

Figure 6.23: Output (top) and inputs (middle and bottom) for a dynamic SAL measurement. The maximum allowed pressure was set to 18 MPa.

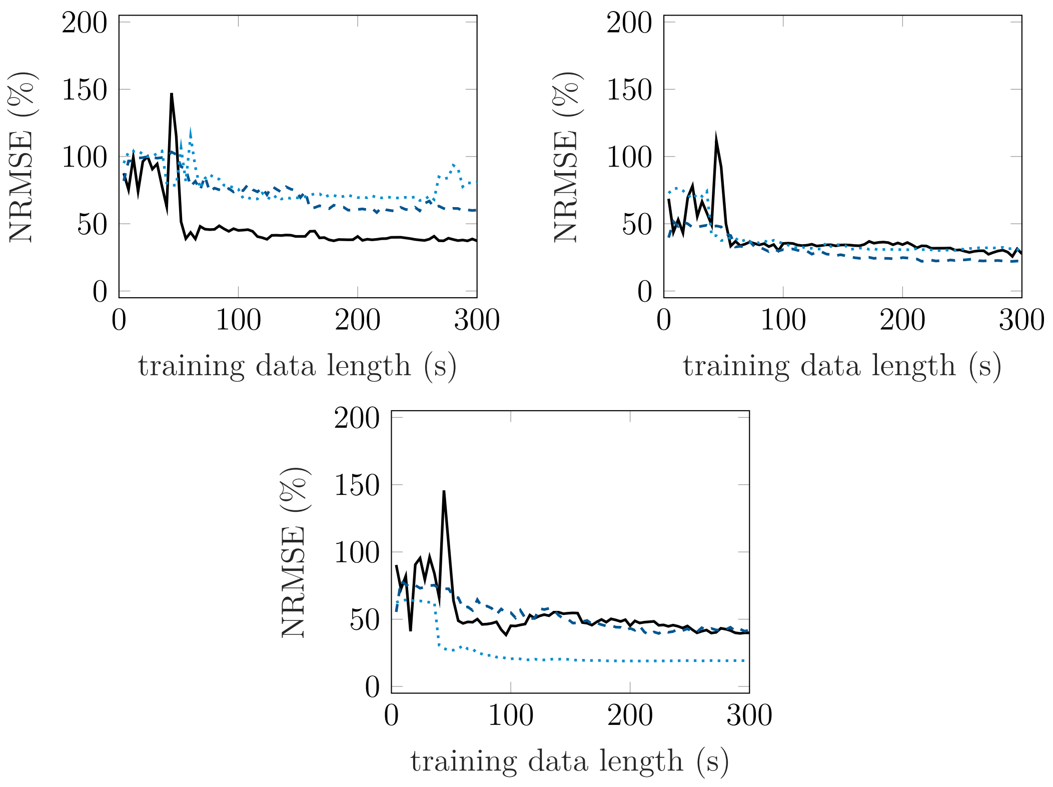

Figure 6.24: NRMSE of a model trained using dynamic SAL (solid lines), ramp signals (dashed lines) or chirp signals (dotted lines). The NRMSE was calculated on three different test datasets: a dynamic SAL run in the top left plot, a ramp DoE in the top right plot, and a chirp DoE in the bottom plot.

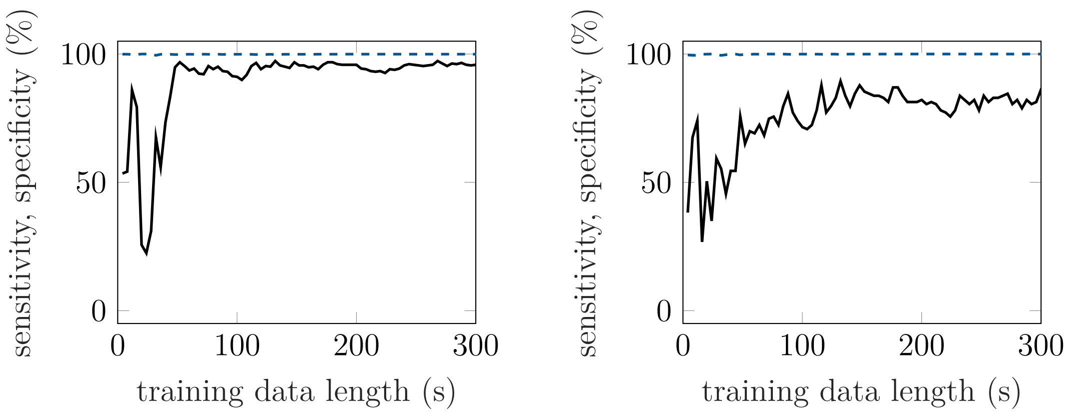

Figure 6.25: Sensitivity (solid lines) and specificity (dashed lines) of dynamic SAL's discriminative model evaluated on two test datasets: The first is a ramp DoE, the second a chirp DoE. Both DoEs result in pressures up to 25 MPa. These measurements were already featured in Section 6.2.7.

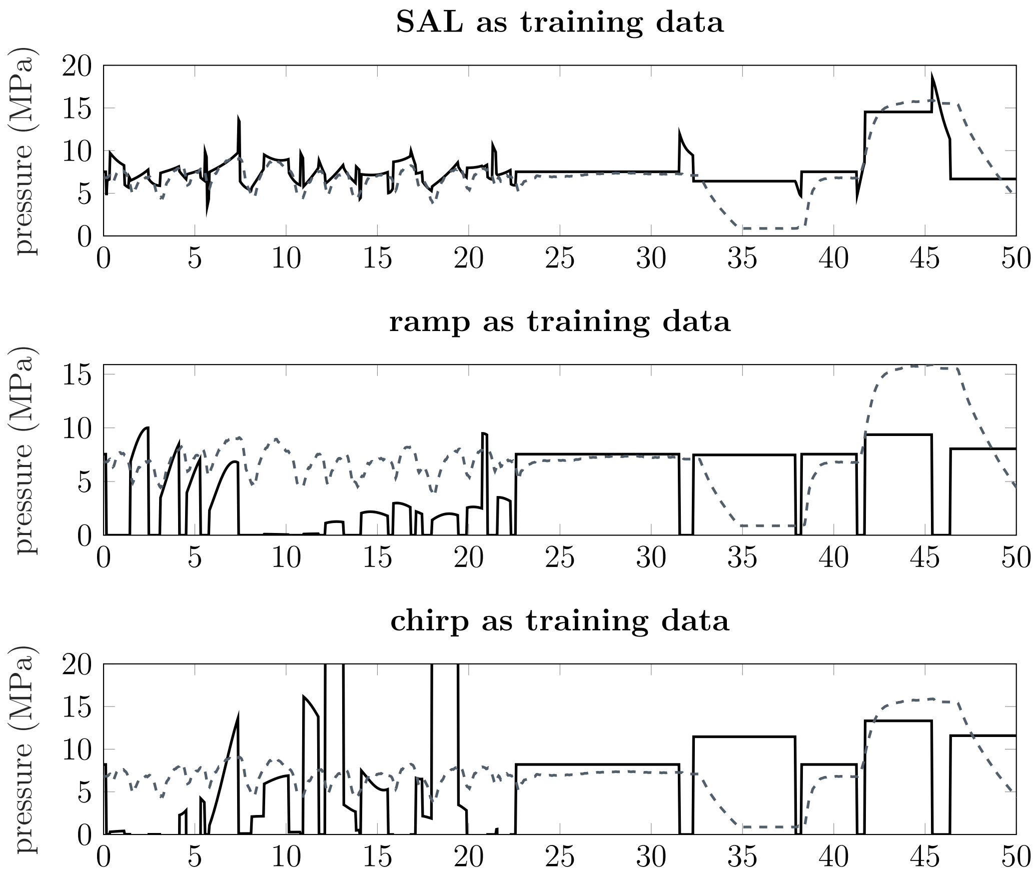

Figure 6.26: Predicted output (solid lines) and measured output (dashed lines) of the first 50 s of the SAL test data. The predictions were made using the models obtained using 300 s of SAL, ramp, or chirp training data.

on test data. All together, five test datasets were recorded: three to benchmark the regression model and two for the discriminative model. The first three are a dynamic SAL run, a chirp, and a ramp DoE. Each of them was recorded with a pressure limit of 18 MPa. The ramp and chirp signals were designed in the stationary safe input space derived using ODCM with improved boundary estimation for this pressure. These datasets are independent of the training data due to varying initializations and different random number realizations. To evaluate the discriminative model, measured data which exceeds the 18 MPa limit is required. Therefore, the ramp and chirp measurements presented in Section 6.2.7 were used. As benchmark, NFIR models similar to that of SAL were learned using a 5-minute chirp and a 5-minute ramp DoE.

As Figure 6.24 shows, each model performs best on a test dataset similar to its training data. SAL's performance is always competitive to the other dataset which was not learned on the type of test data. The absolute value of the NRMSE is comparatively high in all measurements. This becomes obvious when looking at the predicted vs. measured outputs in Figure 6.26. The predicted outputs correspond to the right end of the upper left plot in Figure 6.24. The bad model quality is due to the limited number of features used in the models, i.e. just first order NFIR structures.

Sensitivity and specificity of the dynamic SAL obtained on test data are shown in Figure 6.25. Opposed to the stationary case, they were not calculated by comparing the predicted and measured feasibility of every single test point. Instead, the test data is divided in 1 second long segments and the discriminative model is used to predict the feasibility of each segment independently. Therefore, the same Monte Carlo sampling approach as in dynamic SAL is utilized. This kind of evaluation of the discriminative model closely resembles the way it is used within dynamic SAL. As the figure shows, the specificity never drops below 99 % for both test datasets. This corresponds to the small number of limit violations during the measurements. The sensitivity starts at a lower value and rises up to 96 % and 85 % depending on the test dataset. When applied to ramp data, the performance is better than on chirp data. This observation is similar to the evaluation of the regression model. It is no surprise, as dynamic SAL also optimizes ramp input signals.

The application example of dynamic safe active learning allows the following conclusion: The discriminative model works well and prevents large limit violations. The modeling quality of the regression model is comparable to that of similar models learned on ramp and chirp DoEs and depends largely on the kind of test data. Its absolute quality is not so good though, which calls for the use of more features or a NARX structure in the regression model of the dynamic SAL algorithm. It is possible to run dynamic SAL at a real-world engine. Still, the planing algorithm is not fast enough. Even though some optimizations finish on time, a large fraction of stationary waiting occurs. See Section 5.3.3 for proposals of further improvements.

6.3 Charge Air System

After first evaluations at the HPFS system, selected methods were also applied to the charge air system of a gasoline engine. Compared to the HPFS system, this

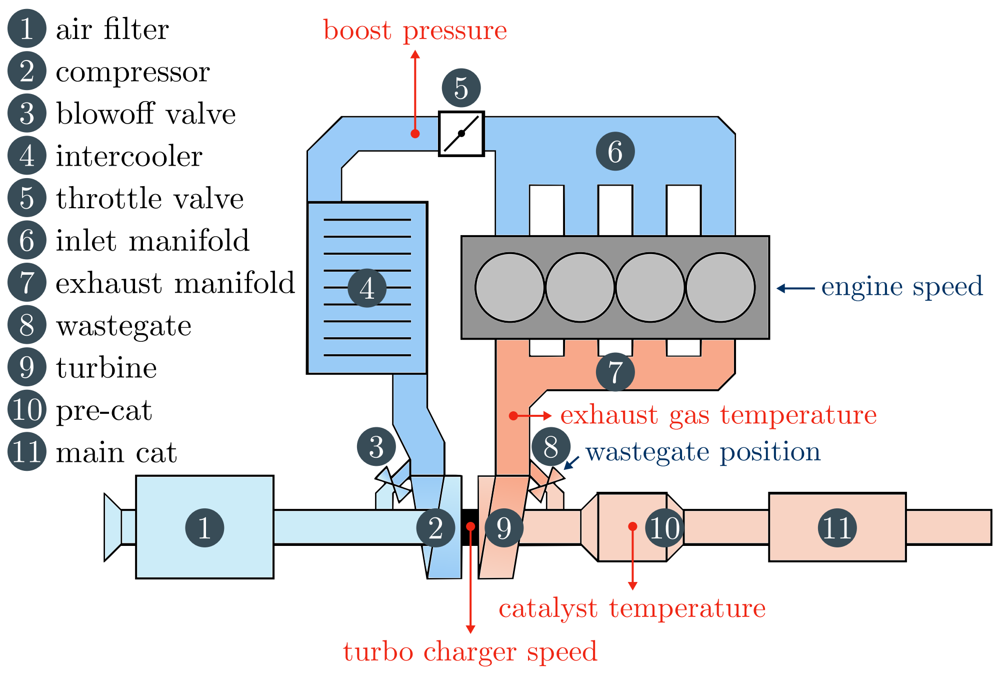

Figure 6.27: Sketch of the charge air system's main components, its inputs (blue) and outputs (red).

application has a higher nonlinearity and larger time constants. Thus, advanced modeling methodologies for this system are more relevant in practical applications. Additionally it is more challenging for the presented measurement methods.

This section starts with a description of the charge air system used in the experiments. Afterwards, two methods are evaluated at this system: ODCM in Section 6.3.2 and dynamic supervised measurements in Section 6.3.3.

6.3.1 System Description

The task of the air system is to provide air for the combustion process. To increase the engine's power and reduce the fuel consumption, turbocharging can be utilized. Thereby, a turbocharger is used to compress the intake air and increase the cylinder fill. The charger is propelled by the exhaust gas stream. It consists of two main parts: the turbine powered by the exhaust gas and the compressor compressing the fresh air. To improve the response of the engine and increase the charge pressure at low engine speeds, the turbocharger is usually designed to reach peak compression already at low engine speeds and thereby low exhaust gas flow rate. To prevent overspeeding of the charger, the charger speed has to be influenced at higher engine speeds. Therefore two common variants exist: turbochargers with wastegate and variable turbine geometry (VTG) turbochargers. In the first case, excessive exhaust gas is bypassed around the turbine through a wastegate. The latter features adjustable guide vanes which influence the impulse and pressure at the turbine blades. This kind of turbocharger is widely used in diesel engines, while gasoline engines, including the one used for the experiments in this thesis, mostly rely on chargers with wastegate. The wastegate can be actuated either pneumatically or electrically. In the engine under test here, an electric wastegate is fitted, which allows a more precise control and an operation independent of the intake pressure.

Figure 6.27 provides a schematic overview of the charge air system considered in the following experiments. The air is sucked in on the lower left of the picture. Behind the air filter, it is compressed by the turbocharger's compressor. The blowoff valve is active and activated to limit pressure peaks if the throttle valve is closed. During the compression, not only the gas' pressure but also its temperature increases. To further raise the air density in the cylinders, the air is cooled down after the compression in the intercooler. The amount of air flowing into the engine is set by the throttle valve just before it enters the inlet manifold and is split to the single cylinders. The hot exhaust gas streams from all cylinders are merged in the exhaust manifold. Afterwards it is either released in the turbine to propel the compressor, or it is passed by the turbine via the wastegate. Finally the gas enters the exhaust aftertreatment consisting of pre-cat and main catalytic converter.

From a control perspective, the charge air system has two input and five output signals. The inputs are

- the engine's speed which determines the intake air's and exhaust gas' volume flow,
- the wastegate position.

The throttle valve could be named as third input. Usually the engine is operated unthrottled as soon as the turbocharger is activated, though. Thus, the throttle valve is not considered as separate input here but completely open (wide open throttle position) during all measurements.

The outputs relevant for this application are

- the boost pressure which is the main output and controlled variable in normal operation,
- the turbocharger speed which may not exceed an upper limit to prevent disintegration of material due to centrifugal forces,
- the exhaust gas temperature which also has to be limited in order to guard components, and
- the catalyst temperature.

All four output signals have upper limits and need to be supervised.

Experiments at the charge air system are more complex than at the HPFS system. Three of the output signals are not measured in a factory model car. Thus additional sensors need to be fitted for turbocharger speed, exhaust gas temperature and catalyst temperature. Furthermore, the experiments cannot be conducted in standstill. As soon as the throttle valve is completely open, the engine produces significant torque which would overspeed it if no load was applied. Experiments on public roads are too dangerous, as a variation of the wastegate results in a changed torque of the engine and thus in an acceleration or deceleration of the car. Thus, all measurements were done at a chassis dynamometer. As test bench time is quite expensive, experiments had to be reduced to a minimum. This prevented a complete evaluation of the DoE methods, as described in the following sections.

6.3.2 ODCM

In a first step, the ODCM algorithm was implemented for the charge air system. Therefore, the online measurement automation (OMA) was developed, see Section 6.1. The algorithm was evaluated at a test bench. Unfortunately, a bug in the automation occured: after the first infeasible point was measured, all following points were also labeled as infeasible. Due to limited resources, the measurements could not be repeated. Test points could not be retrieved either. Thus, no sensitivity, specificity or NRMSE can be calculated. The point distribution is shown in Figure 6.28. Omitting the false measurements, the result looks good. 22 of the 40 planned queries were measured as feasible. 5 infeasible samples were collected and the remaining 13 were skipped.

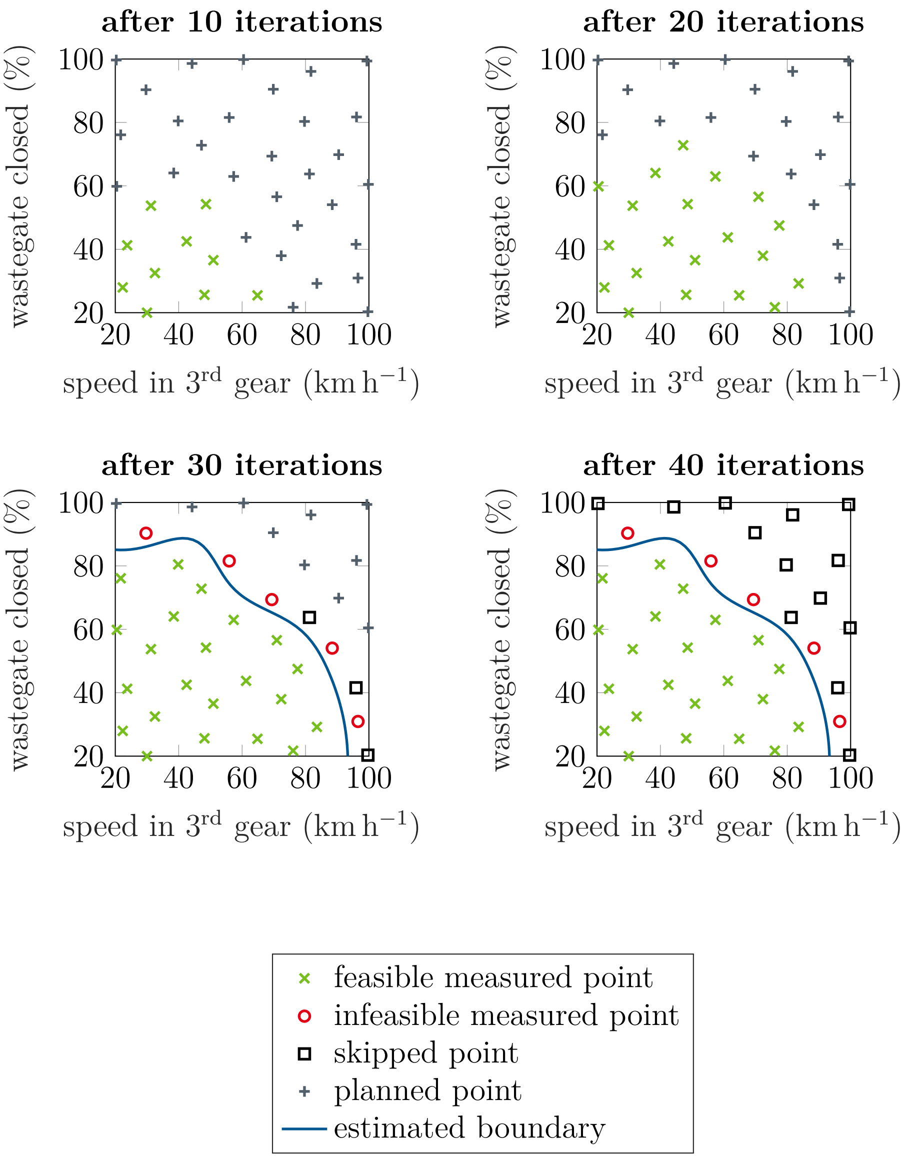

Figure 6.28: Point distribution in the input space using ODCM at the charge air system. Note that, due to a bug in the automation, not all points measured as infeasible are truly infeasible.

6.3.3 Dynamic Supervised Measurement

Two reasons restrained the approach for dynamic DoE and dynamic supervised measurement presented earlier: First, the incomplete ODCM measurements which did not provide a valid stationary boundary. Second, the missing possibility to automatically set the speed of the chassis dynamometer. The test bed used for the experiments presented here has only a manual interface to control its speed. An electronic interface is in development, but could not be evaluated sufficiently early for this thesis, due to limited test bench capacity.

Instead, a simplified procedure was selected to evaluate the dynamic supervised measurement algorithm at the charge air system at least in principle: For one fixed speed, the maximally allowed stationary wastegate position was identified manually. A dynamic 1D-DoE was created and measured without using a supervising controller for this fixed speed. It was thereby assumed that the stationary safe region is also dynamically safe. Based on the collected data a dynamic GP model was learned and used to tune and evaluate the supervising controller. The results of this evaluation in simulation are presented below. Planned test bench experiments with the supervising controller unfortunately failed due to bugs in the automation and limited test bench time. Thus, an evaluation at the real system, especially with variable test bench speed, is left for future research.

Three of the four supervised output signals of the charge air system become safer if the wastegate is opened, i.e. less exhaust gas is routed through the turbine. The catalyst's temperature strongly dependents on the air–fuel equivalence ratio (λ), though. In order to safely limit this temperature, a mere variation of the wastegate would not be sufficient. Varying λ by influencing the injection times would significantly increase the complexity of the automation and require the supervision of additional signals, though. Thus, an alternative approach was chosen here. Before the charge air system is calibrated, a limitation of the catalyst's temperature by reducing λ in high load operating points is usually already implemented. As an additional safety measure, the limit for the exhaust gas temperature were slightly reduced for the subsequent experiments. This allowed to omit the supervision of the catalyst's temperature and thereby simplified the design and tuning of the supervising controller.

The limited operating range due to the fixed engine speed further reduced the number of supervised outputs: The exhaust gas temperature as well as the catalyst's

Table 6.3: Thresholds of boost pressure and turbine speed used in the switching logic. Compare Table 4.1 for a definition of the variables.

variable	boost pressure value	turbine speed value
y_{hardmax}	2 bar	200 000 min^{-1}
y_{max}	1.15 bar	80 000 min^{-1}
y_{set}	1.15 bar	80 000 min^{-1}
$y_{\text{lim,on}}$	1.15 bar	80 000 min^{-1}
$y_{\text{lim,off}}$	1.0925 bar	76 000 min^{-1}
$y_{*,\text{lim,on}}$	1.15 bar	80 000 min^{-1}
$y_{*,\text{lim,off}}$	1.0925 bar	76 000 min^{-1}

temperature changed only inconsiderably during the DoE. Thus, no reasonable model of these two outputs could be learned and a supervision was neither possible nor necessary. This reduced the number of supervised outputs to two, i.e. the boost pressure and the turbine speed.

For each supervised output one supervising controller, consisting of the controller itself and the prediction model, were designed. Both of the controllers calculate a wastegate position. In the end, the minimum of the two controller outputs and the DoE value is applied to the system. As prediction model, an ARX model and a simple linear extrapolation were compared in [18]. Even though the ARX model obtained a more precise prediction especially of the transient behavior, the linear extrapolation turned out to be more robust. Thus, the latter approach was selected. The controllers were designed as PID controllers using the frequency domain tuning method described in Section 4.2.1 and some subsequent manual fine-tuning. They use the measured supervised outputs as controlled variables. Bumpless transfer is not yet implemented. Instead, the integrators are initialized with a constant value each time a controller is switched on. An anti-windup disables the integration as soon as a controller is overruled by the DoE, a saturation of the manipulated variable or the other controller. The sampling time of the system was set to 50 ms. Due to safety reasons, the wastegate was not completely closed when recording the training data for the model used for the evaluation. Consequently, the wastegate is closed no more than 70 % in the DoE used for evaluation.[I] Due to the strong nonlinearity of the system, only comparatively low boost pressure and turbo charger speed values are

[I]Keep in mind that a 100 % closed wastegate results in the maximum boost pressure, turbine speed, and engine power, as all exhaust gas propels the turbine. On the opposite, an open (i.e. 0 % closed) wastegate is safe, as almost all exhaust gas bypasses the turbine.

reached. The real system limits would be at 2 bar boost pressure and 200 000 min^{-1} turbine speed. The thresholds of the switching logic are presented in Table 6.3. They were artificially reduced compared to the physical limits to provoke controller actions on the limited input range. To obtain an increased toggle suppression, the controller is not switched off until it has been on for 5 steps, i.e. 0.25 s.

The simulated input and output signals are presented in Figure 6.29. For this evaluation, a tripartite DoE was used: The first part consists of APRBS signals, the second of a chirp signal, and the last part of ramps. The controllers prevent large overshoot above the defined limits. The simulated supervised outputs top at 1.1569 bar and 80 478 min^{-1}. Due to the different switching limits and the minimum on-time, the controller is active 71 % of the time in case of the boost pressure supervision and 29 % of the time in case of the turbine speed supervision. Nonetheless, only 18 % of the samples are altered by the controllers. This indicated the controllers are switched on during a large portion of the time, but request larger and thus less safe wastegate positions than the DoE.

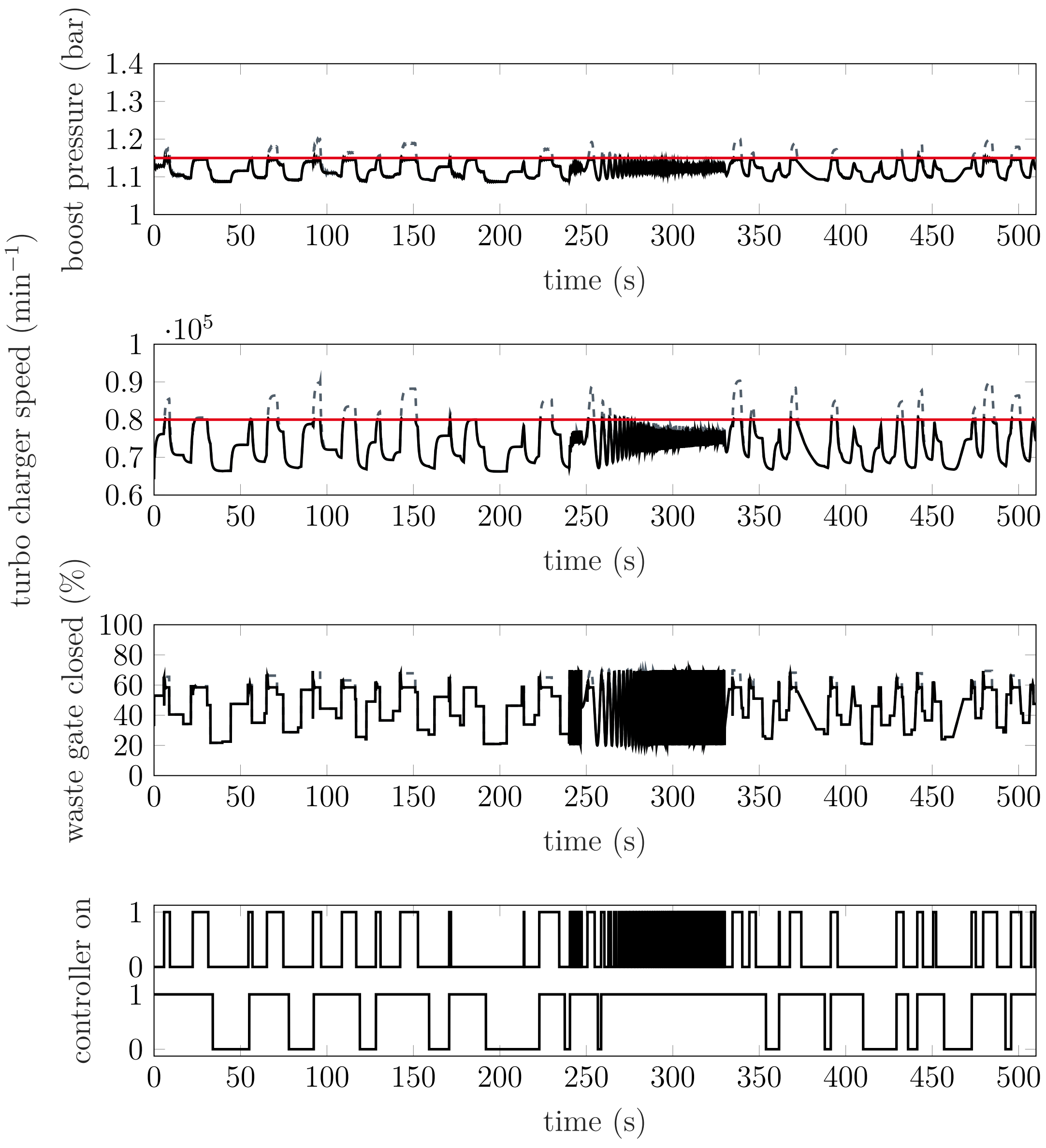

Figure 6.29: Outputs (top), input (middle) and controller state (bottom) for the charge air system. The upper controller state corresponds to the boost pressure controller, the lower to the turbo charger speed controller. The solid lines denote the input and outputs of a simulation with supervising controller, the dashed lines correspond to a simulation without supervising controller.

7 Conclusions and Outlook

This chapter summarizes the results of the thesis and names options for future research.

7.1 Conclusions

In this thesis, two strategies for the safe and efficient generation of training data for dynamic data-based modeling were considered, compare Figure 1.1. The first strategy is tripartite: after an identification of the stationary boundary, a dynamic DoE is scaled into said boundary and a dynamic supervised measurement is performed. The second strategy uses dynamic safe active learning to directly learn a model of the system under test. Both approaches were evaluated at a real high pressure fuel supply system in a test vehicle. Online design of experiment with constraint modeling (ODCM) and the supervised dynamic measurement algorithm were also applied to the charge air system of a gasoline engine.

Four methods for stationary adaptive DoEs were considered. Besides the original ODCM algorithm, ODCM with improved boundary estimation, ODCM with discriminative model, and stationary safe active learning (SAL) were analyzed. While ODCM showed solid results regarding modeling quality and accuracy of the estimated boundary, the latter could be slightly improved using ODCM with improved boundary estimation. A significantly more precise boundary was estimated using ODCM with discriminative model. Regarding stationary modeling accuracy, ODCM performed best. Even stationary SAL was not able to outperform ODCM, while the obtained boundary using this algorithm was more precise.

The dynamic supervised measurement algorithm reliably prevented major limit violations in all application examples. Depending on the system under test, different prediction models and controller parameters performed best.

Safe active learning's greatest benefit turned out to be the discriminative model. In all three variants (stationary, for controller tuning, and dynamic) it prevented large limit violations. Compared to the ODCM classifier, the number of limit violations was reduced. Unfortunately, the active learning part did not lead to an improved modeling quality given the same number of queries. Comparing SAL and SBO for controller parameter tuning, both variants showed individual strengths and weaknesses, as discussed in Section 5.2.3. SBO found the better controller in the end, though.

Dynamic SAL performed on par with ramp and chirp signals regarding modeling quality. It depends on the type of test data which signal type shows the best results. The method was successfully evaluated at a real-world system for the first time.

When comparing the two strategies towards a dynamic model, dynamic SAL revealed its potential. With some future improvements, it could identify the system's behavior more easily and almost completely automated. By now, the three-step strategy is more mature, though, as more development effort was spent on it in the past. When applied to more complex systems, a combination of both is conceivable. For example, dynamic SAL could be safeguarded by a supervising controller.

7.2 Outlook

As always, working on one research question opens up a number of new questions. In this section, some of them are mentioned.

ODCM with discriminative model revealed its potential as a combination of ODCM and stationary SAL suitable to be put into practice. Some further development is necessary though. For example it has to be evaluated if the algorithm is prone to deadlocks, if all candidate points are wrongly classified as infeasible. Alternative strategies for this case need to be invented. Afterwards, the method could be evaluated at more advanced systems with higher dimensional input space, for example at an engine test bench or a chassis dynamometer. Perhaps the idea of this method could be combined with ODCM with improved boundary estimation. The latter algorithm should also work together well with the discriminative model.

For all SAL variants, other query-criteria should be evaluated. For example, the mutual information criterion showed promising results in a comparable application, see [66]. Perhaps this way SAL could outperform offline DoEs not only regarding boundary estimation, but also in modeling quality.

Safe active learning for control and safe Bayesian optimization could be applied to more complex and thus practically more relevant controller architectures. It should be evaluated how well the algorithm scales with an increasing number of controller parameters.

Dynamic SAL proved to be a valuable future option for safe online dynamic design of experiments. The algorithm is still in its infancy though. Major tasks for future research are the consideration of the whole covariance matrix during the trajectory optimization also in real-world applications, the adaption to NARX models which are more suitable for many real-world systems, the use of more flexible input signals like Bézier curves, and online hyperparameter training. Unfortunately, all of these enhancements tend to increase the computational load, which is another very important aspect to work on in the future. Optimization of the algorithm as well as the implementation and perhaps the transition to a more efficient programming language like C++ need to further reduce the runtime of the trajectory optimization.

The supervising controller at the charge air system should be enhanced to varying engine speeds in the next step. As the other methods, it could be evaluated at more complex systems with higher input dimension to proof its practical applicability. Further methods for tuning the controller could be assessed, especially such that do not rely on a system model. The gain-scheduling controller could benefit from a method which is able to place operating points near the boundary automatically.

A Description of Used Software Products

Throughout this thesis, different commercial software products were mentioned. This appendix gives a short description and references to these products.

INCA

The *Integrated Calibration and Application Tool* (INCA) is a tool for measurement and calibration tasks at engine control units published by ETAS. Using, for example, an interface module and an ECU interface, it is possible to measure and change labels in an engine control unit (compare Figure 6.1). Measured data and editable labels can be visualized using measurement displays, oscilloscopes or editors for values, curves, and maps which can be freely arranged in a so-called experiment environment. Using the *Measurement Data Analyzer* (MDA) bundled with INCA, measured data can be plotted and analyzed offline. Additional software is available to extend the functionality of INCA, which is partly covered in the following. In this thesis, the INCA versions 7.0 and 7.2 were used. More information on INCA and its add-ons can be found in [15].

INCA-FLOW

INCA-FLOW is a tool used to automate calibration procedures in INCA also published by ETAS. It features a graphical specification of processes similar to a flowchart. Thereby it allows to automate procedures with little programming knowledge. More complex functionality can be integrated using Matlab functions or dynamic-link libraries (dlls).

INCA-MIP

The *Matlab Integration Package* is an addon to Matlab provided by ETAS. It allows to remote-control INCA from Matlab and transfer data between the two. Thereby, measurement and calibration processes can be automated in Matlab.

INCA-MCE

Measurement and Calibration Embedded is another add-on for INCA by ETAS. It provides real-time connections to the ECU using a specialized interface module and the protocols EtherCAT and iLinkRT. Originally targeted at test bench automation systems, it also allows ECU-access from other software like Matlab. As described in Section 6.1, reading and writing to and from an ECU using Matlab is faster by an order of magnitude using iLinkRT compared to INCA-MIP.

Matlab

Matrix Laboratory is a software originally developed for numerical calculations. It is published by The MathWorks Inc. By now it can also be used as an on-the-fly compiled scripting language. A lot of toolboxes by MathWorks and third party developers, partly also open source, is available. They provide diverse functionality from connection to other systems (compare INCA-MIP) via tools for statistical analysis to powerful optimization algorithms. In recent versions object-oriented programming is possible. Compared to lower-level programming languages like C and C++, Matlab code is easier to write but, depending on the application, often slower. In this thesis, mostly the versions R2015b and R2016b were used. More information on Matlab can be found in [60].

Bibliography

[1] Karl Johan Åström and Tore Hägglund. *Advanced PID Control*. ISA - The Instrumentation, Systems, and Automation Soc., 2006. ISBN: 978-1-55617-942-6.

[2] K.J. Åström and T. Hägglund. *PID Controllers. Theory, Design, and Tuning*. 2nd edition. Research Triangle Park: Instrument Society of America, 1995. ISBN: 978-1-55617-516-9.

[3] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. "The Quickhull algorithm for convex hulls". In: *ACM Transactions on Mathematical Software* 22.4 (1996), pp. 469–483.

[4] Lucas Bentele. "Umsetzung und Überwachung der transienten Streckenvermessung des ungeregelten Hochdruckrailsystems eines Benzin-Direkteinspritzer-Motors zur Simulation im geschlossenen Regelkreis". BA thesis. Duale Hochschule Baden-Württemberg, 2014.

[5] F. Berkenkamp, A. Krause, and A.P. Schoellig. "Safe controller optimization for quadrotors with Gaussian processes". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2016, pp. 491–496. DOI: `10.1109/ICRA.2016.7487170`.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Verlag GmbH, 2011. ISBN: 978-0-387-31073-2.

[7] Terrence Blevins and Mark Nixon. *Control Loop Foundation: Batch and Continuous Processes*. International Society of Automation, 2011. ISBN: 978-1-936007-54-7.

[8] Eric Brochu, Vlad M. Cora, and Nando de Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: *CoRR* abs/1012.2599 (2010).

[9] Kun Li Chien, J. A. Hrones, and J. B. Reswick. “On the Automatic Control of Generalized Passive Systems”. In: *Transactions of the American Society of Mechanical Engineers* 74 (1952), pp. 175–185.

[10] Edwin van Dam et al. *Space-filling designs*. Feb. 2, 2015. URL: `https://spacefillingdesigns.nl` (visited on 11/24/2017).

[11] Michael Deflorian. “Versuchsplanung und Methoden zur Identifikation zeitkontinuierlicher Zustandsraummodelle am Beispiel des Verbrennungsmotors”. PhD thesis. Technischen Universität München, 2011.

[12] Michael Deflorian, Florian Klöpper, and Joachim Rückert. “Online dynamic black box modelling and adaptive experiment design in combustion engine calibration”. In: *IFAC Proceedings Volumes* 43.7 (2010), pp. 703–708.

[13] Nico Didcock, Nikolaus Keuth, and Andreas Rainer. “Non-Convex Hulls for Engine Applications”. In: *Automotive Data Analytics, Methods, DoE. Proceedings of the International Calibration Conference*. Ed. by Karsten Röpke and Clemens Gühmann. 2017. ISBN: 978-3-8169-3316-8.

[14] Nico Didcock, Andreas Rainer, and Stefan Jakubek. “Online Design of Experiments in the Relevant Output Range”. In: *Optimization and Optimal Control in Automotive Systems*. Ed. by Harald Waschl et al. Cham: Springer International Publishing, 2014, pp. 273–289. ISBN: 978-3-319-05371-4. DOI: `10.1007/978-3-319-05371-4_16`.

[15] ETAS GmbH. *INCA Software Products*. 2018. URL: `www.etas.com/inca` (visited on 01/22/2018).

[16] Aleksandar Fandakov. “Transiente datenbasierte Streckenmodellierung zur virtuellen Applikation eines Hochdruckrailsystems”. MA thesis. Universität Stuttgart, 2015.

[17] Valerii V. Fedorov and Peter Hackl. *Model-Oriented Design of Experiments*. Springer New York, 1997. DOI: `10.1007/978-1-4612-0703-0`.

[18] Mariusz Francik. “Supervising Controller for Dynamic Measurements of the Charge Air System”. MA thesis. Hochschule Esslingen – University of Applied Sciences, 2017.

[19] Thomas Godward, Heiko Schilling, and Steffen Schaum. “Use of Dynamic Testing and Modeling in the Engine Calibration Process”. In: *Design of Experiments (DoE) in Engine Development*. Ed. by Karsten Röpke. Expert-Verlag GmbH, 2013. ISBN: 978-3-8169-3217-8.

[20] Christoph Hametner et al. “Optimal experiment design based on local model networks and multilayer perceptron networks.” In: *Engineering Applications of Artificial Intelligence* 26 (2013), pp. 251–261. ISSN: 0952-1976. URL: `http://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0952197612001224&site=eds-live`.

[21] Benjamin Hartmann and Mark Schillinger. “Verfahren und Vorrichtung zum Vermessen eines zu testenden Systems”. German pat. req. 10 2016 206 627.7. Robert Bosch GmbH. 2016.

[22] Benjamin Hartmann and N. Tietze. “Verfahren und Vorrichtung zum Vermessen eines physikalischen Systems”. German pat. req. 10 2014 226 485.5. Robert Bosch GmbH. 2014.

[23] Benjamin Hartmann et al. “Online-methods for engine test bed measurements considering engine limits”. In: *16th Stuttgart International Symposium*. Wiesbaden: Springer Fachmedien, 2016. DOI: `10.1007/978-3-658-13255-2_92`.

[24] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, 2009. ISBN: 978-0-387-84858-7.

[25] Jing He et al. “Steady State Calibration for Catalyst Heat-up Optimization on Gasoline Direct Injection Engine”. In: *Design of Experiments (DoE) in Powertrain Development*. Ed. by Karsten Röpke and Clemens Gühmann. 2015. ISBN: 978-3-8169-3316-8.

[26] Tim Oliver Heinz et al. “Excitation Signal Design for Nonlinear Dynamic Systems. Proceedings of the International Calibration Conference”. In: *Automotive Data Analytics, Methods, DoE*. Ed. by Karsten Röpke and Clemens Gühmann. Expert Verlag, 2017. ISBN: 978-3-8169-3381-6.

[27] Geritt Kampmann, Nataša Kieft, and Oliver Nelles. “Support Vector Machines for Design Space Exploration”. In: *Proceedings of the World Congress on Engineering and Computer Science*. Vol. II. 2012. ISBN: 978-988-19252-4-4.

[28] Richard Khoury and Douglas Wilhelm Harder. *Numerical Methods and Modelling for Engineering*. Springer International Publishing, 2016. DOI: `10.1007/978-3-319-21176-3`.

[29] Douglas Leith and W.E. Leithead. "Survey of gain-scheduling analysis and design". In: *International Journal of Control* 73 (2000).

[30] Daniel Liberzon. *Switching in Systems and Control*. Birkhäuser, 2012. ISBN: 978-1-4612-6574-0.

[31] Lennart Ljung. *System identification. Theory for the user*. 2. ed., 12. printing. Prentice Hall information and system sciences series. Literaturverz. S. 565 - 593. Upper Saddle River, NJ: Prentice Hall PTR, 2012. XXII, 609. ISBN: 978-0-13-656695-3.

[32] Jan Lunze. *Regelungstechnik 1*. 7. Auflage. Berlin, Heidelberg: Springer-Verlag GmbH, 2008. ISBN: 978-3-540-68907-2.

[33] Yutaka Murata et al. "Application of Model Based Calibration to Mass Production Diesel Engine Development for Indian Market". In: *Design of Experiments (DoE) in Powertrain Development*. Ed. by Karsten Röpke and Clemens Gühmann. 2015. ISBN: 978-3-8169-3316-8.

[34] Oliver Nelles. *Nonlinear System Identification*. Springer-Verlag GmbH, 2001. ISBN: 978-3-642-08674-8. DOI: `10.1007/978-3-662-04323-3`.

[35] Oliver Nelles. *Regelungstechnik*. Lecture Notes. 2015. URL: `http://www.mb.uni-siegen.de/mrt/lehre/rt/rt_skript.pdf` (visited on 06/21/2017).

[36] Duy Nguyen-Tuong and Nils Tietze. *Transient Design of Experiment for Model Learning*. Research rep. Robert Bosch GmbH, 2015. Confidential.

[37] The Duy Nguyen-Tuong et al. "Verfahren und Vorrichtung zum Generieren von zulässigen Eingangsdatentrajektorien für ein Test- bzw. Prüfsystem". German pat. DE 10 2013 206 258 A1. Robert Bosch GmbH. 2014.

[38] Luc Pronzato and Werner G. Müller. "Design of computer experiments: space filling and beyond". In: *Statistics and Computing* 22.3 (2012), pp. 681–701. DOI: `10.1007/s11222-011-9242-3`.

[39] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. The MIT Press, 2006. ISBN: 978-0-262-18253-9.

[40] Peter Renninger and Metodi Aleksandrov. “Rapid Hull Determination: A New Method to Determine the Design Space for Model Based Approaches”. In: Karsten Röpke. *Design of Experiments (DoE) in Engine Development II*. Essen: Brill, Ulrich, 2005. ISBN: 978-3-8169-2498-2.

[41] Robert Bosch GmbH, ed. *Ottomotor-Management. Systeme und Komponenten*. 3rd ed. Friedr. Vieweg & Sohn Verlag, 2005. ISBN: 978-3-8348-0037-4.

[42] David Salomon. *Curves and Surfaces for Computer Graphics*. Springer New York, 2007. ISBN: 978-0-387-28452-1.

[43] Nino Pascal Sandmeier and Karsten Röpke. “Efficient online test plan calculation with integrated boundary detection for modern model based engine calibration”. In: *Design of Experiments (DoE) in Powertrain Development*. Ed. by Karsten Röpke and Clemens Gühmann. 2015. ISBN: 978-3-8169-3316-8.

[44] Stefan Scheidel and Marie-Sophie Vogels. “Model-based iterative DoE in highly constrained spaces”. In: *Automotive Data Analytics, Methods, DoE. Proceedings of the International Calibration Conference*. Ed. by Karsten Röpke and Clemens Gühmann. 2017. ISBN: 978-3-8169-3316-8.

[45] Mark Schillinger et al. “Modern Online DoE Methods for Calibration – Constraint Modeling, Continuous Boundary Estimation, and Active Learning”. In: *Automotive Data Analytics, Methods, DoE. Proceedings of the International Calibration Conference*. Ed. by Karsten Röpke and Clemens Gühmann. Expert Verlag, 2017. ISBN: 978-3-8169-3381-6.

[46] Mark Schillinger et al. “Safe Active Learning and Safe Bayesian Optimization for Tuning a PI-Controller”. In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 5967–5972. ISSN: 2405-8963. DOI: `10.1016/j.ifacol.2017.08.1258`.

[47] Mark Schillinger et al. “Safe Active Learning of a High Pressure Fuel Supply System”. In: *9th EUROSIM Congress on Modelling and Simulation*. Oulu, Finland, 2016. DOI: `10.1109/EUROSIM.2016.137`.

[48] Robert Munnig Schmidt. *The Design of High Performance Mechatronics. High-Tech Functionality by Multidisciplinary System Integration*. Ed. by Georg Schitter, Adrian Rankers, and Jan van Eijk. 2nd revised edition. Amsterdam: Delft University Press, 2014. ISBN: 978-1-61499-368-1.

[49] Jens Schreiter. “Data-efficient and Safe Learning with Gaussian Processes”. PhD thesis. Institute for Parallel and Distributed Systems at the University of Stuttgart. In preparation.

[50] Jens Schreiter, Duy Nguyen-Tuong, and Marc Toussaint. “Efficient sparsification for Gaussian process regression”. In: *Neurocomputing* 192 (June 2016), pp. 29–37. DOI: `10.1016/j.neucom.2016.02.032`.

[51] Jens Schreiter et al. “Safe exploration for active learning with Gaussian processes”. In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 133–149.

[52] Lukas Schweizer. “Implementierung und Evaluation eines sicheren aktiven Lernverfahrens”. MA thesis. Karlsruher Institut für Technologie, 2017.

[53] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009.

[54] Claude E. Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27 (1948), pp. 379–423, 623–656.

[55] Bernhard Sieber. “Überwachungsregelung für die dynamische Vermessung nichtlinearer Systeme”. MA thesis. Hochschule Esslingen – University of Applied Sciences, 2016.

[56] Markus Stadlbauer. “Optimization of excitation signals for nonlinear systems”. PhD thesis. Technische Universität Wien, 2013.

[57] M. Stadlbauer et al. “Online measuring method using an evolving model based test design for optimal process stimulation and modelling”. In: *2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings*. 2012. DOI: `10.1109/I2MTC.2012.6229185`.

[58] Yanan Sui et al. “Safe Exploration for Optimization with Gaussian Processes”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France, 2015.

[59] David Martinus Johannes Tax. “One-class classification. Concept-learning in the absence of counter-examples”. PhD thesis. Technische Universiteit Delft, 2001.

[60] The MathWorks Inc. *Matlab*. 2018. URL: `www.mathworks.com/matlab` (visited on 01/22/2018).

[61] Nils Tietze. "Model-based Calibration of Engine Control Units Using Gaussian Process Regression". PhD thesis. Darmstadt: Technische Universität, Feb. 2015.

[62] Nils Tietze et al. "Model-based calibration of engine controller using automated transient design of experiment". In: *14th Stuttgart International Symposium*. Wiesbaden: Springer Fachmedien, 2014. DOI: 10.1007/978-3-658-05130-3_111.

[63] Ky Khac Vu et al. "Surrogate-based methods for black-box optimization". In: *International Transactions in Operational Research* (2016). ISSN: 1475-3995. DOI: 10.1111/itor.12292.

[64] C. K. I. Williams and D. Barber. "Bayesian classification with Gaussian processes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.12 (1998), pp. 1342–1351. ISSN: 0162-8828. DOI: 10.1109/34.735807.

[65] Yingchao Xiao, Huangang Wang, and Wenli Xu. "Hyperparameter Selection for Gaussian Process One-Class Classification". In: *Neural Networks and Learning Systems, IEEE Transactions on* 26.9 (Sept. 2015), pp. 2182–2187. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2014.2363457.

[66] Yijiang Xie. "Optimal Steady-State Base-Calibration of Model Based ECU-Functions". In: *Simulation and Testing for Vehicle Technology*. Springer International Publishing, 2016, pp. 245–265. DOI: 10.1007/978-3-319-32345-9_18.

[67] J. G. Ziegler and N. B. Nichols. "Optimum settings for automatic controllers". In: *Transactions of the American Society of Mechanical Engineers* 64 (1942), pp. 759–768.

[68] Christoph Zimmer, Mona Meister, and Duy Nguyen-Tuong. "Safe Active Learning for Time-Series Modeling with Gaussian Processes". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 2735–2744.