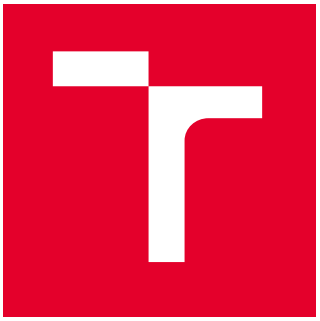

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING

FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

HYBRID METHOD FOR MODELLING AND STATE ESTIMATION OF DYNAMIC SYSTEMS

HYBRIDNÍ METODA PRO MODELOVÁNÍ A POZOROVÁNÍ STAVŮ DYNAMICKÝCH SYSTÉMŮ

DOCTORAL THESIS

DIZERTAČNÍ PRÁCE

AUTHOR — **Ing. Martin Brablc**

AUTOR PRÁCE

SUPERVISOR — **doc. Ing. Robert Grepl, Ph.D.**

ŠKOLITEL

BRNO 2023

Abstrakt

Tato Disertační práce se zabývá vývojem nové hybridní metody pro současné odhadování stavů a parametrů nelineárních dynamických systémů, založené na myšlence lokálních lineárních modelů, která využívá odhad nejistoty parametrů modelu pro automatické nastavení některých parametrů Kalmanova filtru (KF), čímž se výrazně zjednodušuje její nasazení a nastavení v praktických aplikacích. V první části se disertační práce věnuje shrnutí aktuálního stavu poznání v oblasti dynamických systémů, simultánní estimace, KF a modelování nelineárních dynamických systémů. Následně se ve dvou samostatných kapitolách věnuje modifikaci KF pro situace, kde dominantním vlivem způsobujícím procesní šum jsou nepřesné parametry modelu, a dále modifikaci metody *Receptive field weighted regression (RFWR)* tak, aby mohla být použita pro duální estimaci. Nakonec práce popisuje vyvinutou hybridní metodu složenou z modifikovaných algoritmů RFWR a KF nazvanou *Receptive field dual estimation - (RFDE)* a demonstruje její funkčnost na simulačních i reálných datech.

Klíčová slova

Dynamické systémy; Modelování a simulace; Teorie řízení; Systémy se spojitým časem; Systémy s diskrétním časem; Aproximační metody; Lokální lineární modely; Lokální lineární regrese; Receptivní pole; Vážená regrese; MATLAB; Odhadování patametrů; Odhadování stavů; Kalmanův filtr; Procesní šum; Kovariance Gaussovského šumu; Silmultánní estimace, Sloučená estimace; Duální estimace; Rekursivní metoda nejmenších čtverců;

Abstract

This Doctoral thesis deals with the development of a new hybrid method for the dual estimation of states and parameters of nonlinear dynamic systems based on the idea of local linear models, which uses the estimation of the uncertainty of the model parameters to automatically adjust the parameters of the Kalman filter (KF), thus greatly simplifying its deployment and adjustment in practical applications. In the first part, the dissertation summarises the current state of knowledge in the field of dynamic systems, simultaneous estimation, KF and modelling of nonlinear dynamic systems. Then, in two separate chapters, it discusses the modification of KF for situations where inaccurate model parameters are the dominant influence causing process noise, and the modification of the *Receptive field weighted regression (RFWR)* method so that it can be used for dual estimation. Finally, the paper describes the developed hybrid method composed of modified RFWR and KF algorithms called *Receptive field dual estimation - (RFDE)* and demonstrates its performance on simulation and real data.

Keywords

Dynamic systems; Modelling and simulation; Control theory; Continuous time systems; Discrete time systems; Approximation methods; Local linear models; Local linear regression; Receptive fields; Weighted regression; MATLAB; Parameter estimation; State estimation; Kalman filter; Process noise; Gaussian Noise Covariance; Simultaneous estimation; Joint estimation; Dual estimation; Recursive least squares;

Bibliographic citation

BRABLC, M. *Hybrid Method for Modelling and State Estimation of Nonlinear Dynamic Systems*. Brno: Brno University of Technology, Faculty of Mechanical Engineering, 2023. 116 pages, Supervisor: doc. Ing. Robert Grepl, Ph.D.

I hereby affirm that the presented doctoral thesis titled *Hybrid Method for Modelling and State Estimation of Nonlinear Dynamic Systems* is my genuine work and that it was created with the support of the stated references, under the supervision of doc. Ing. Robert Grepl, Ph.D.

Brno, 12.1.2023 **Martin Brablc**

Díky mé ženě Hance za dlouhodobou podporu a trpělivost, díky VS, MR a RG za spoustu plodných diskuzí a díky všem, kteří se v minulosti pohybovali nebo momentálně pohybují v Mechlabu a vytváří tohle jedinečné, někdy nervy drásající, ale produktivní prostředí.

Martin Brablc

Contents

List of Figures

List of Tables

1 Introduction

Dynamic systems often occur in both research and engineering projects across various scientific fields. When working with such systems, the need to control or otherwise influence the system often arises. Controlling the system can simply mean to achieve a target value or follow a set trajectory of a single state or multiple states (physical quantities) of the dynamic system, maintaining the system in a stable or unstable state, initiation, maintaining or damping oscillations, and many other forms.

In this thesis, we deal primarily with systems that are typical for the field of mechatronics. These are systems with concentrated parameters of low order, often nonlinear, which typically originated in several other domains, e.g., electronics, electromagnetics, solid body mechanics, thermo- or hydromechanics, etc. A very common property is the existence of an algebraic or differential relationship between the system states, meaning that we can often find pairs or larger groups of states being in the time domain derivative/integral relation, e.g., position - velocity, charge current, etc., which gives the system special properties that we further investigate. A typical example of such a mechatronic system is an electric drive.

When implementing a control algorithm, especially when dealing with nonlinear systems, there are three distinct tasks which need to be solved:

- creating a model of the system,
- choosing and tuning a signal processing filter or state observer,
- designing n and tuning the control algorithm itself.

Usually, the model of the system we want to control, often called the plant, is required to be able to simulate the system response in various situations and in combination with various filters and controllers without the risk of destroying the real plant. There are many different approaches to modelling dynamic systems, and the choice is often very application-specific. For the purpose of this thesis, we assume that we can deduce a set of ordinary differential equations (ODEs) based on the first-principles approach, which describes or approximates the system at hand with reasonable precision. By reasonable precision, we mean that we can use it to identify the system states and significant nonlinearities, at least on what states the nonlinearities depend on. This set of ODEs can be directly used as a model of the system (assuming that we are able to estimate or measure the unknown parameters) or it can be further approximated by one of many various methods which loosely fall into categories called local and global methods.

The global methods work with a single, usually very complex model structure (equation, function) that is valid over the whole state space of the system. Common examples of such methods are polynomial models, neural networks, symbolic

programming, Autoregressive Exogenous (ARX), Autoregressive Moving Average exogenous (ARMAX), linear state space models, transfer functions, etc. On the other hand, local methods usually work with multiple simple model structures that are valid only over a portion of the state space. Most notably, these methods are based on local linear models such as Locally weighted learning (LWL), Receptive field-weighted regression (RFWR), Locally weighted projection regression (LWPR), locally linear model tree (LOLIMOT), etc.

In practical applications, it is rare for the system structure to change spontaneously, however, its parameters can change over time. For example, some parts of a mechanism may wear, clog, or degrade, the electric resistance may change with temperature, etc. Also, the model parameters could have been estimated with data measured on a reference system and then the model could have been used to simulate a second system of the same type, manufactured with finite production tolerances, thus, having slightly different parameters. In similar cases, it is necessary to use model types that are adaptive. For this reason, we mainly focus on local modelling techniques, which generally adapt more easily to new data and are considered more stable during the adaptation process.

The second base task that needs to be dealt with when developing a control algorithm can be summarised as estimating the states of the system to be controlled. This task is important as it provides the necessary feedback information for any control algorithm, and the signal quality (accuracy, delay, signal-to-noise ratio, etc.) greatly influences the quality and stability of the control process. Solutions to this task generally fall into two categories: filters and state estimators. Putting aside obvious low-pass and anti-aliasing filters, derivative filters are often used in situations where one of the states is not measured directly, such as central difference or Savitzky-Golay type filters. For example, when we only measure the position of a mechanism and its speed, which is also one of the system states that needs to be determined using a derivative filter.

In cases when common filtration techniques are not sufficient, a state estimator should be used. The theory of state estimator methods is well developed and verified by many practical applications. The most common ones are the Luenberger state observer and the Kalman filter with its many variations. All the methods use a similar approach: predict the system behaviour based on a model and correct the prediction using the measurement of the states that are available. In this way, states that are not measured directly can also be estimated, provided that they are observable. The standard variants of both of these estimators can be properly (and optimally) applied on linear dynamic systems only, however, there are variations to the Kalman filter designed to work with nonlinear systems too. Namely, these are the Extended Kalman filter (EKF), the Unscented Kalman filter (UKF), and the Particle filter.

The third task is the design of the control algorithm itself. Aside from the common linear control design methods (PID, LQR , etc.), there are many methods and approaches for the design of a control law for nonlinear dynamic systems. At random, we can mention feedback linearisation, reinforcement learning, model predictive control (MPC), State-Dependent Riccati Equation (SDRE), gain scheduling,

and many others. Independent of the method we choose to implement, the control design methods mostly have to assume the knowledge of all the controlled system states. This requirement is often not mentioned when describing the design of complex nonlinear control algorithms and is left for the reader to deal with on their own, based on the specific application. In practice, this presents a problem which is neither trivial nor exact and usually requires the use of one of the above-mentioned state estimation techniques.

To simplify the design process, the three tasks are, in most practical applications, being solved independently. First, the models of the system are created, then filters or state estimators are implemented and tuned in simulation or with the real plant. Lastly, both are used to implement a suitable control algorithm. The design methods for all three tasks have many parameters which need to be set by the developer, while some are not exact, lacking a guide or criterion function to be correctly set. This is the most common, independent design process, even though the results of all the tasks are highly intertwined and, in the end, serve to solve a single criterion, the control process quality. An example of such problematic parameters is that of the Kalman filter, especially process noise covariance, or others like the number of particles used in a Particle filter, a Model predictive control (MPC) horizon, an RFWR local model distribution optimisation step, and many others, which mostly (vaguely) represent the confidence embedded in some of the information sources or convergence aggressiveness. Most of these parameters, in most practical applications, cannot be exactly set and tuning several parameters at once by hand is complicated and requires a great deal of knowledge and intuition from the developer.

The problems described above often result in a suboptimal control design and push developers to choose the design method out of the great variety of available algorithms they know and have experience tuning, as implementing a new, previously unknown algorithm requires acquiring enough experience to tune it properly. With that in mind, the main goal of this dissertation is to simplify the control design process by removing, or at least minimising, the need to tune some of the most critical design parameters. Based on previous research and experience, this goal can be achieved by combining independent tasks into a single adaptive method or algorithm, which would only be subject to the overall control quality. We will mainly focus on combining the first two tasks, modelling and state estimation, as these have practical applications on their own and may serve as a basis for future research in combining all three tasks. To summarise, the goal is not to achieve better results, but to achieve a simpler practical implementation by the end user of the algorithms.

These approaches fall into the category called *simultaneous estimation*, which allow the estimation of both system states and parameters of a fixed model structure and work exclusively with nonlinear Kalman filters. *Simultaneous estimation* methods can be further divided into subcategories, the so-called *joint estimation* that considers the system parameters as constant states, and *dual estimation*, which uses two parallel filters sharing their best estimates of the states and parameters respectively. Both these approaches, their advantages and disadvantages are described

in more detail in Section 2.4.

Our goal is to develop a hybrid method for the simultaneous estimation of states and parameters of nonlinear dynamic systems, which would adapt, online, the model of the system and tune a state estimator as new data become available. It is an approach similar to dual estimation, where one of the filters is replaced by a more complex approximation method, preferably based on local linear models. Both parallel algorithms share not only their best estimates, but also confidence in that estimate, which allows the automatic parameter tuning.

Naturally, there is also the idea of linking all three tasks, but in this thesis, we shall only work with the first two as the first step in this research direction.

2 Theoretical Survey

This chapter deals primarily with the description and summary of the most important methods and algorithms upon which the rest of the thesis is based. First, we briefly describe the common notation used throughout this thesis in Section 2.1, and then in Section 2.2, we provide a detailed description of dynamic systems from an abstract and mathematical point of view to establish and explain what we mean by different dynamic model types in the following chapters.

Furthermore, in Section 2.3, we describe the classical linear Kalman filter to summarise the notation we use in this thesis and the assumptions with which it works, so that we can expand the algorithm in the next chapters.

Lastly, Sections 2.4 and 2.5 provide a brief introduction to simultaneous estimation and local approximation methods, respectively, to describe the current state of the art in these fields.

2.1 Notation

In addition to common notation and rules, we use the following throughout the thesis:

- scalar quantity - math script: x
- column vector - bold math script: $\boldsymbol{x} = [x_1, x_2, ...]^T$
- vector element - lower case math script with index: x_i
- matrix - upper case bold math script: $\boldsymbol{A}$
- matrix element - lower case math script with double index: a_{ij}
- set (common) - double struck upper case: $\mathbb{R}$
- set (other) - Greek upper case: Ω
- scalar or vector valued function - lower case math script or Greek: f or ϕ
- time domain derivative - dot over the sign: $\frac{dx}{dt} = \dot{x}$, $\frac{d^2x}{dt^2} = \ddot{x}$
- null vector - boldface zero math script: $\mathbf{0}$
- value estimate - hat over the sign: $\hat{x}$
- technical term - cursive: *state space*

Further notations that require a more detailed description are introduced when first used.

2.2 Dynamic systems

In this section, we provide a definition of compact dynamic systems, together with the most common terms used in this research area. Keeping in mind that reformulating common theoretical knowledge when existing sources can be used may be redundant work, Section 2.2.1 is necessary to provide meaning to the various model types and terms used in Section 2.2.2 and in the following chapters.

2.2.1 Definition

Under the term *dynamic system*, we can imagine a somewhat simplified representation (model) of a real system that we have in mind, specifically a system that evolves over time. By "simplified", we mean that we only observe the outward behaviour of the system and the specific physical quantities we want to observe or predict, whose model is usually in the form of a set of differential equations. The level of detail is highly dependent on the actual application.

We can find dynamic systems in almost every research field, economics, biology, chemistry, engineering, physics, and more. An electrical circuit, a motor, swinging a pendulum, the spread of an epidemic, and chemical reactions can all be viewed as dynamic systems. In the literature, for example, in [1, 2], we can find various mathematical definitions of a dynamic system, however, all of them define it using similar terms: a *state*, a *phase/state space* and an *evolution function*.

A *state* or a *state vector* of a dynamic system is any set of values of such quantities at any given time instant that they completely describe the future behaviour of the system, without the necessity of knowing the past. To give a few examples, the state of a mathematical pendulum is completely described by two quantities - the angle and angular velocity, or a DC motor usually has three states - the current, angle, and angular velocity. An alternative definition comes from energy conservation laws - a *state* of a dynamic system is a set of values of such quantities that can describe the energy accumulated in the system at any given time. The quantities assembled in the state vector form the *state space*.

To give a more erudite definition, the *state space* of a dynamic system $\Omega \subseteq \mathbb{R}^n$, where n is the number of quantities assembled in a state (i.e., the *order* of the dynamic system), is a set of all possible states $\boldsymbol{x}_t \in \Omega$ in which the system can be at any given time instant t. In addition, the term *phase space* has a similar meaning. In some texts both are used interchangeably, some others like [1] use it in the context of classical mechanics, where momentum and position are used to describe the dynamics of each degree of freedom of the mechanism, instead of velocity and position. For the purpose of this thesis, these terms can be considered equivalent.

Theoretically, the state space of a real system has various mathematical properties depending on the quantities we choose, but, for the systems we deal with, it is enough to say that we deal with systems with concentrated parameters with continuous, complete, and finite state spaces. These assumptions prevent n from being infinite (although, in the applications that we deal with, n is usually lower than 10).

Also, this means that we can use sets of ordinary differential equations (ODEs) to describe models of dynamic systems instead of partial differential equations (PDEs).

We can theoretically form infinitely many state spaces for any dynamic system (e.g. by a linear transformation), however, not all of them are practical. Also, for convenience, sometimes we choose to artificially increase the system order and add another quantity to the state vector.

Depending on the type of the set of all possible time instants T, a system can be in, we can differentiate dynamic systems into *continuous* in time ($\mathrm{T} \subseteq \mathbb{R}$) and *discrete* in time ($\mathrm{T} = \{kT_s : k \subseteq \mathbb{Z}\}$, where T_s is a *sampling period*).

Then, for both discrete and continuous systems, the *evolution function* (2.1)

$$\phi\left(t\,,t_0\,,\boldsymbol{x}_0\right) : \mathrm{T} \times \mathrm{T} \times \Omega \to \Omega \tag{2.1}$$

unequivocally describes the state $\boldsymbol{x}_t \in \Omega$ of the system at time instant $t \in \mathrm{T}$ defined by the initial state $\boldsymbol{x}_0 \in \Omega$ at time instant $t_0 \in \mathrm{T}$. The evolution function can be applied recursively to track a trajectory in the state space of a system. For example, the state of a system $\boldsymbol{x}_2$ at time instant t_2 can be written based on a transformation from state $\boldsymbol{x}_1$ at time instant t_1 or based on $\boldsymbol{x}_0$ at t_0 as in (2.2).

$$\boldsymbol{x}_2 = \phi\left(t_2\,,t_0\,,\boldsymbol{x}_0\right) = \phi\left(t_2\,,t_1\,,\boldsymbol{x}_1\right) \tag{2.2}$$

Since state $\boldsymbol{x}_1$ can also be induced from state $\boldsymbol{x}_0$ as in (2.3),

$$\boldsymbol{x}_1 = \phi\left(t_1\,,t_0\,,\boldsymbol{x}_0\right) \tag{2.3}$$

we can also apply ϕ recursively as in (2.4).

$$\boldsymbol{x}_2 = \phi\left(t_2\,,t_1\,,\phi\left(t_1,t_0\,,\boldsymbol{x}_0\right)\right) \tag{2.4}$$

Usually, when describing a dynamic system, we start from a set of differential equations of various orders derived using the first principle methods, such as the Lagrange equations. Using an appropriate choice of the state quantities, the set can be reassembled as a set of first-order differential equations and written as (2.5),

$$\dot{\boldsymbol{x}}(t) = f\left(\boldsymbol{x}(t)\,,t\right) \tag{2.5}$$

where $\dot{\boldsymbol{x}}(t) \in \mathbb{R}^n$ is a vector of the time domain derivatives of each of the state quantities contained in $\boldsymbol{x}(t)$ and f in (2.6),

$$f\left(\boldsymbol{x}(t),t\right) : \Omega \times \mathrm{T} \to \mathbb{R}^n \tag{2.6}$$

where f is a vector-valued function that describes the dynamics of the system. Then,

the evolution function (2.1) is the solution to the *initial value problem* (IVP) of the set of equations (2.6) [1].

It is important to note that Equation (2.5) represents an explicit form of the set of ODEs. In some special cases, the transformation to the explicit form is not possible, and we need to work with an implicit form of the equation instead. In this thesis, we work with systems that can always be transformed this way.

Equation (2.5) describes a dynamic system without an external input, a so-called *autonomous* system. It is important to note that the equation defines a vector field across the entire state space Ω, which means that any trajectory throughout that space would be tangent to the vector field in any given state.

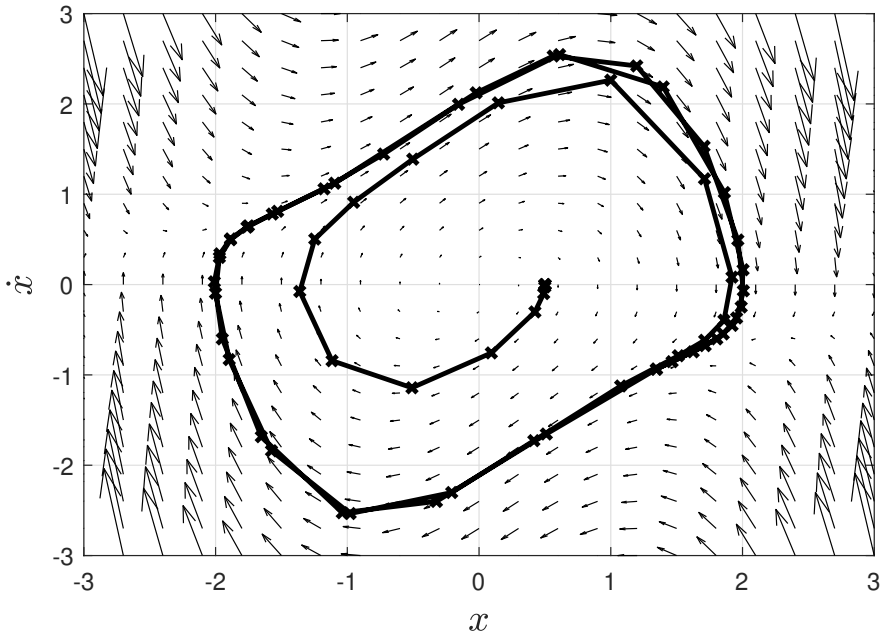

Figure 2.1: An example of a vector field defined by the Equation (2.5) over the state space for the specific example of a Van der Pol oscillator [3].

Naturally, finding the exact analytical solution to an IVP with (2.5) is generally possible only in trivial cases, especially with linear systems. As such, the most common solution to the problem is through numerical integration methods using one of the standard ODE solvers (Euler, Runge-Kutta, etc.) [4]. In situations like these, we effectively replace systems which are naturally continuous in time by a discrete version. Furthermore, with systems that use $\mathrm{T} = \{kT_s : k \in \mathbb{Z}^+\}$, we will denote a state $\boldsymbol{x}$ at a time instant $t_k \in T$ by a subscript index k, i.e., $\boldsymbol{x}_{t_k} = \boldsymbol{x}_k$.

It is useful to develop the evolution function for this special case of a system with a discrete set T to be able to make a transformation between two consecutive time instants as in (2.7).

$$\boldsymbol{x}_k = \phi\left(t_k\,, t_{k-1}\,, \boldsymbol{x}_{k-1}\right) \tag{2.7}$$

We will refer to this special case of the evolution function as *evolution operator* $\Phi : \Omega \times \mathrm{T} \to \Omega$. Then, we can write a discrete model of an autonomous dynamic system as (2.8).

$$\boldsymbol{x}_k = \Phi\left(\boldsymbol{x}_{k-1}\,, t_{k-1}\right) \tag{2.8}$$

2.2.2 Dynamic system models

In practice, we often deal with systems that are not autonomous and have an external input. For this reason, we must extend our description of dynamic systems by a set of all possible inputs $\Psi \subseteq \mathbb{R}^m$, where m is the number of external inputs and $\boldsymbol{u}(t) \in \Psi$ is a specific set of inputs applied to the system at any given time instant $t \in T$. Then, we can expand the models (2.5) and (2.8) by an additional argument, acquiring a model of the dynamics of a *controlled* system (2.9). Also, to further simplify the notation, we will be marking arguments which are functions of time, e.g. $\boldsymbol{x}(t)$, simply as $\boldsymbol{x}$ as the time dependence is obvious in most cases.

$$\dot{\boldsymbol{x}} = f\left(\boldsymbol{x}\,, \boldsymbol{u}\,, t\right) \tag{2.9}$$

Assuming a known time signal of the input $\boldsymbol{u}$, the evolution function (2.10) can also be a solution to the IVP of (2.9).

$$\phi\left(t\,, t_0\,, \boldsymbol{x}_0, \boldsymbol{u}_0\right) : \mathrm{T} \times \mathrm{T} \times \Omega \times \Psi \to \Omega \tag{2.10}$$

and we can also define the evolution operator for a discrete system with an input as (2.11).

$$\boldsymbol{x}_k = \Phi\left(\boldsymbol{x}_{k-1}\,, \boldsymbol{u}_{k-1}\,, t_{k-1}\right) \tag{2.11}$$

Apart from models (2.9) and (2.11), which we refer to as *forward models*, it is useful in some applications, especially in control design, to use an inverse of the forward models, i.e. the continuous and discrete *inverse models* in the form of (2.12) and (2.13), respectively.

$$\boldsymbol{u} = f^{-1}\left(\boldsymbol{x}\,, \dot{\boldsymbol{x}}\,, t\right) \tag{2.12}$$

$$\boldsymbol{u}_k = \Phi^{-1}\left(\boldsymbol{x}_k\,, \boldsymbol{x}_{k-1}\,, t_{k-1}\right) \tag{2.13}$$

2.2.3 Additional models

Together with the basic system models and dynamic systems themselves, there are a number of additional models being used for convenience in practice. Often, it

is useful to separate the output of the system from its state. Generally, it can be described by Equation (2.14).

$$\boldsymbol{y} = g\left(\boldsymbol{x}, \boldsymbol{u}, t\right) \tag{2.14}$$

where $\boldsymbol{y}$ is the output of the system corresponding to state $\boldsymbol{x}$ and input $\boldsymbol{u}$. Often, the output is simply a subset of state $\boldsymbol{y} \subseteq \boldsymbol{x}$. The output often represents one or more quantities that we are interested in tracking or controlling.

Furthermore, we can also use a measurement model to link the quantities that we actually measure with the state, using Equation (2.15).

$$\boldsymbol{z} = h\left(\boldsymbol{x}, t\right) \tag{2.15}$$

where $\boldsymbol{z}$ is the measurement vector corresponding to state $\boldsymbol{x}$. The measurement can also be a subset of the state vector $\boldsymbol{z} \subseteq \boldsymbol{x}$.

A very important special case of all the above mentioned models is a linear system. Assuming that the system at hand is linear, time-invariant, and continuous, the general model of the system dynamics (2.9), the model of the output (2.14), and the model of the measurement (2.15), can be transformed into linear case equations (2.16), (2.17), and (2.18), respectively, forming the so-called state space model of a dynamic system.

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{2.16}$$

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \tag{2.17}$$

$$\boldsymbol{z} = \boldsymbol{H}\boldsymbol{x} \tag{2.18}$$

Similarly, for linear time-invariant discrete systems, the discrete dynamic model (2.11) with input and measurement models can be transformed into (2.19), (2.20), and (2.21), respectively, forming the so-called discrete state space model of a dynamic system.

$$\boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_{k-1} + \boldsymbol{G}\boldsymbol{u}_{k-1} \tag{2.19}$$

$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k \tag{2.20}$$

$$\boldsymbol{z}_k = \boldsymbol{H}\boldsymbol{x}_k \tag{2.21}$$

Lastly, by adding the assumption that the model of a continuous system (2.9) and a discrete system (2.11) is not dependent on time, but $\boldsymbol{F}$ and Φ depend on a set of parameters $\boldsymbol{b} \in \mathbb{R}^p$ instead, where p is the number of parameters, both can be rewritten in the form of (2.22) and (2.23), respectively. Both will be useful in the later chapters.

$$\dot{\boldsymbol{x}} = f\left(\boldsymbol{x}\,, \boldsymbol{u}\,, \boldsymbol{b}\right) \tag{2.22}$$

$$\boldsymbol{x}_k = \Phi\left(\boldsymbol{x}_{k-1}\,, \boldsymbol{u}_{k-1}\,, \boldsymbol{b}_{k-1}\right) \tag{2.23}$$

Also, when talking about parameters, it is useful to note a special case of models *linear in parameters*. This is a weaker condition than a linear system, it allows for nonlinearities, but the parameter $\boldsymbol{b}$ must only occur as an argument of the linear functions. [5]

Typical examples of functions linear in parameters are a linear model (2.24), a polynomial model (2.25) or a nonlinear model (2.26).

$$\dot{x} = \boldsymbol{x}^T \boldsymbol{b} \tag{2.24}$$

$$\dot{x} = b_1 + b_1 x + b_3 x^2 + ... \tag{2.25}$$

$$\dot{x} = b_1 \sin x_1 + b_2 \exp x_2^2 \tag{2.26}$$

On the other hand, examples of models that do not fall into this category are (2.27), (2.28), and (2.29).

$$\dot{x} = b_1 x^{b_2} \tag{2.27}$$

$$\dot{x} = b_1 + b_2 \sin b_3 x \tag{2.28}$$

$$\dot{x} = b_1 \exp b_2 x_1^{b_3} \tag{2.29}$$

This property can be defined by fulfilling the condition (2.30). A model of a dynamic system $\boldsymbol{F}\left(\boldsymbol{x}\,, \boldsymbol{u}\,, \boldsymbol{b}\right)$ is linear in parameters exactly when

$$\frac{\partial f\left(\boldsymbol{x}\,, \boldsymbol{u}\,, \boldsymbol{b}\right)}{\partial b_i} \neq \boldsymbol{0} \wedge \frac{\partial^2 f\left(\boldsymbol{x}\,, \boldsymbol{u}\,, \boldsymbol{b}\right)}{\partial b_i^2} = \boldsymbol{0}\,, \forall b_i \in \boldsymbol{b} \tag{2.30}$$

The advantage of models linear in parameters is in the great simplification in the process of parameter estimation. It allows the use of the least square method and its variants (LS, WLS, RLS), gaining the best estimate in terms of the MSE criteria in a single calculation without the need for iterative optimisation methods [5].

2.3 Kalman filter

The *Kalman filter* was developed by several mathematicians and physicists in the late 1950s and is named after the Hungarian mathematician and engineer Rudolf E. Kálmán, who published his now famous paper [6] in 1960. The algorithm was first

used in aircraft and spacecraft navigation applications. Nowadays, it is a standard method for dynamic system state filtering and estimation, and sensor fusion in various fields of research and engineering.

The filter falls into the category of Recursive Bayesian filters and combines a model of the system and measurement data to create the best possible estimation of the state vector of the system. We can find many detailed descriptions of the algorithm, for example, in [6, 7, 8, 9, 10].

In the common implementation, the Kalman filter (KF) is based on a model of a linear discrete dynamic system with continuous state space, which can be described by (2.19) and a model of the measurement process as described by (2.21). Both of these models are assumed to be imprecise, and the imprecision is modelled by Gaussian zero-mean noise. Specifically, a noise term is added to both models as in (2.31) and (2.32),

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{G}_k \boldsymbol{u}_{k-1} + \boldsymbol{w}_k \tag{2.31}$$

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \tag{2.32}$$

where $\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{Q}_k, 0)$ represents the *process noise* term and $\boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{R}_k, 0)$ represents the *measurement noise* term, where $\boldsymbol{Q}_k = cov(\boldsymbol{w}_k)$ is the process noise covariance and $\boldsymbol{R}_k = cov(\boldsymbol{v}_k)$ is the measurement noise covariance.

The Kalman filter iteratively predicts the behaviour of the system based on the process model and corrects the prediction using the measurement. Obviously, the noise terms are not known, which means that both the process prediction and the measurement have some degree of uncertainty. It is very important to note, here ,that the noise terms may represent an unknown or unmodeled effect that corrupts the results, an inherently stochastic disruption effect, or an imperfection in the model itself. In Equation (2.31), both the previous state $\boldsymbol{x}_{k-1}$ and the current state $\boldsymbol{x}_k$ are the true (unknowable) states of the system, which means that the noise term $\boldsymbol{w}_k$ can also be viewed as a complement to the rest of the equation to make the model fit.

In practice, the true states are not known, and we work with the best estimates we have, together with a probability density function in the form of a covariance matrix, which represents the possible spread or the level of confidence we have in our estimates. All probabilities are assumed to be Gaussian.

Various notations are used to describe KF. In this thesis, we will use the hat symbol over a variable symbol (e.g. $\hat{\boldsymbol{x}}$) to mark that it is an estimate of the true value and expand the lower indexing notation by the number of the iteration the estimate is based on. For example, $\hat{\boldsymbol{x}}_{i|j}$ represents an estimate of the value of $\boldsymbol{x}$ in iteration i based on information from iteration j. Typically, we use either information from the previous iteration ($j = i - 1$), which is called the *a priori* or the *prior* estimate, or information from the actual iteration ($i = j$), which is called the *a posteriori* or the *posterior* estimate.

Then, every state estimate that we work with is a tuple (2.33),

$$\left(\hat{\boldsymbol{x}}_{i|j},\, \hat{\boldsymbol{P}}_{i|j}\right) \tag{2.33}$$

where $\hat{\boldsymbol{x}}_{i|j}$ is the estimate of the state vector $\boldsymbol{x}_i$ and $\hat{\boldsymbol{P}}_{i|j}$ is the estimate of the covariance matrix $\boldsymbol{P}_i$ that describes the confidence of the state estimate in the form of a Gaussian *probability density function* (PDF). We can also say that the true state position in the state space is described by $\boldsymbol{x_i} \sim \mathcal{N}(\hat{\boldsymbol{P}}_{i|j},\, \hat{\boldsymbol{x}}_{i|j})$ with $\mathcal{N}$ as the PDF. To simplify the notation, we will drop the hat symbol on the covariance matrix estimate and simply refer to it as $\boldsymbol{P}_{i|j}$, as is common in most texts, as we never deal with true covariance, only with its estimate, making the estimate symbol obsolete.

Specifically, the Kalman filter algorithm is as follows. We start with an initial guess of the state estimate and the covariance matrix $(\hat{\boldsymbol{x}}_{0|0},\, \boldsymbol{P}_{0|0})$. Then, in every iteration k of the tracking process (usually every time we get a new measurement), we perform the *prediction step* to acquire the *a priori* estimate of the state vector and its covariance according to (2.34) and (2.35).

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{P}_k \hat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{G}_k \boldsymbol{u}_{k-1} \tag{2.34}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^T + \boldsymbol{H}_k \tag{2.35}$$

After that, we use the measurement z_k to perform *the update step* to acquire the *a posteriori* estimates. First, we calculate the Kalman gain $\boldsymbol{K}$ according to (2.36) and then perform the updates (2.37) and (2.38), where I is the identity matrix of the appropriate order.

$$\boldsymbol{H}_k = \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T \left(\boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T + \boldsymbol{R}_k\right)^{-1} \tag{2.36}$$

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{H}_k \left(z_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_{k|k-1}\right) \tag{2.37}$$

$$\boldsymbol{P}_{k|k} = \left(I - \boldsymbol{H}_k \boldsymbol{H}_k\right) \boldsymbol{P}_{k|k-1} \tag{2.38}$$

After the update, we acquire the new estimates $(\hat{\boldsymbol{x}}_{k|k},\, \boldsymbol{P}_{k|k})$, which can be used again in the next iteration $k+1$.

Notice that the only parameters (apart from the system and measurement models described by $\boldsymbol{F}$, $\boldsymbol{G}$ and $\boldsymbol{H}$) that we need to set are the process and measurement covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. They are usually set as constant, which results in the Kalman gain $\boldsymbol{K}$ and the estimate covariance $\boldsymbol{P}$ rapidly approaching stable optimal values. However, the algorithm allows all the matrices $\boldsymbol{F}$, $\boldsymbol{G}$, $\boldsymbol{H}$, $\boldsymbol{Q}$, $\boldsymbol{R}$ to change values between the iterations as the conditions of the system change. In such a case, $\boldsymbol{K}$ and $\boldsymbol{P}$ do not stabilise, but adjust at each step according to the changing parameters. This property will be used in later chapters.

Generally, correctly setting both the process and the measurement noise covariances is the main problem when implementing KF in a practical implementation. The measurement noise covariance $\boldsymbol{R}$ can usually be measured, estimated, guessed,

or read from a sensor datasheet. Correctly setting the process noise covariance $\boldsymbol{Q}$ is another issue, and the question *How to set the process noise covariance for a Kalman filter?* is probably one of the most googled questions by students, engineers, and researchers implementing their first filters [7].

We can find several *ad hoc* methods to set $\boldsymbol{Q}$ in some special cases where the perfect Kalman conditions are met, for example in [7, 6], also several adaptive autotuning algorithms were proposed, for example in [11, 12, 13, 14, 15]. However, in most practical applications, the $\boldsymbol{Q}$ matrix is set experimentally by intuition. This problem is described and developed in more detail in Section 4.1.

The Kalman filter is proven to be the optimal state estimator in the least mean square sense, however, only if all of the initial assumptions are met. In practice, it is often used, even if some of them are not met precisely. For example, some sensors (e.g., an encoder) do not produce Gaussian noise, most systems contain at least some nonlinearities, etc., the KF can still be used in most applications as long as the violations are not overly significant.

For situations when we deal with significant nonlinearities, there are modifications to the Kalman filter, namely the Extended Kalman Filter (EKF), which works with a more general system and measurement model in the form of Equations (2.39) and (2.40) and allows f and h to be nonlinear.

$$\boldsymbol{x}_k = f\left(x_{k-1},\, u_{k-1}\right) + \boldsymbol{w}_k \tag{2.39}$$

$$\boldsymbol{z}_k = h\left(x_{k-1}\right) + \boldsymbol{w}_k \tag{2.40}$$

The filter algorithm is similar to the linear version. The only change is in the prediction step, where we can directly use the nonlinear model of the system as Equation (2.41).

$$\hat{\boldsymbol{x}}_{k|k-1} = f\left(\hat{\boldsymbol{x}}_{k-1|k-1},\, \boldsymbol{u}_{k-1}\right) \tag{2.41}$$

Then, wherever the system and measurement matrices $\boldsymbol{F}$ and $\boldsymbol{H}$ are required throughout the calculations, they are acquired as the Jacobian matrix of the nonlinear functions according to (2.42) and (2.43), respectively.

$$\boldsymbol{F}_k = \frac{\partial f\left(\hat{\boldsymbol{x}}_{k-1|k-1},\, \boldsymbol{u}_{k-1}\right)}{\partial \boldsymbol{x}} \tag{2.42}$$

$$\boldsymbol{H}_k = \frac{\partial h\left(\hat{\boldsymbol{x}}_{k-1|k-1}\right)}{\partial \boldsymbol{x}} \tag{2.43}$$

The EKF is the simplest and generally computationally faster extension of the linear KF that is suitable for nonlinear systems, see [16] for more details.

For more complex problems with more significant nonlinearities, the Unscented Kalman Filter (UKF) is often used as an alternative. The UKF relies on heuristi-

cally chosen Sigma points which propagate through the nonlinear model and only then are used to construct the covariance. As opposed to the linearisation used in EKF, this method, sometimes called the *Unscented transformation*, is known to give much better results for highly nonlinear systems, at the cost of computational and implementation complexity. For more details on the Unscented Kalman filter, see [17, 7, 18].

Lastly, both the EKF and the UKF fail when the system is nonlinear in such a way that the probability distributions of the state estimates propagated through the system dynamics function are not unimodal, or otherwise unfit to be modelled by Gaussians. The most computationally expensive estimator for such nonlinear systems is the particle filter, which drops most assumptions and simulates the propagated PDFs numerically using Monte Carlo-like methods. Again, for more details on the Particle filter, see [7].

2.4 Simultaneous Estimation

The purpose of *simultaneous estimation* is, in the course of the measurement process, striving to estimate both the states and the parameters of a model of a dynamic system at the same time. Assuming a model in the form of (2.22) or (2.23), we target the state vector $\boldsymbol{x}_k$ and the parameters $\boldsymbol{b}_k$, iteratively estimating them as best as possible based on the known input u_k and the measurement vector z_k. We also consider the structure of the model, the function f for continuous or the Φ for discrete systems, to be completely known.

Typically, a suitable variant of the Kalman filter is used for the estimation of the state vector, outputting the estimate of the state vector $\hat{x}_{k|k}$ together with its covariance matrix $\boldsymbol{P}_{k|k}$, as described in Section 2.3 or, in more detail, regarding the choice of the most suitable method for a specific task, in [19, 20].

There are many various methods for *parameter estimation* (PE), which would iteratively adjust the estimate based on new data points (online). In case of a system which is linear or linear in parameters, for example, the recursive least squares method (RLS) can be used (see [5] for further details on RLS and other PE methods).

In the case where we need to estimate both the states and the parameters, there are two distinct approaches that can be used. These are *joint estimation* and *dual estimation*, which we outline in further detail in the subsequent sections. Regardless of the chosen approach, all the applications utilise some of the Kalman filter variants in different configurations. Common characteristics, typical for the KF, also follow from this. It is always necessary to correctly set the critical tuning parameters, especially the process noise covariance.

2.4.1 Joint estimation

The simplest way to implement *simultaneous estimation* is *joint estimation*. As mentioned in the Introduction, while using this approach, we consider the parameters of the dynamic system to be estimated as constant states. This means that

we can create an extended state vector $\overline{\boldsymbol{x}}_k = \left[\boldsymbol{x}_k^T, \boldsymbol{b}_k^T\right]^T$ and reformulate the initial models (2.22) and (2.23) into (2.44) and (2.45), respectively.

$$\dot{\overline{\boldsymbol{x}}}_k = f\left(\overline{\boldsymbol{x}}_k, \boldsymbol{u}_k\right) \tag{2.44}$$

where $\dot{\overline{\boldsymbol{x}}}_k = \left[\dot{\boldsymbol{x}}_k^T, \boldsymbol{0}\right]^T$.

$$\overline{\boldsymbol{x}}_k = \Phi\left(\overline{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_{k-1}\right) \tag{2.45}$$

Similarly to any other system, we can use the KF to estimate the state vector of the system (2.44) or (2.45). The state extension, in most cases, causes the system to be highly nonlinear, which forces us to use one of the nonlinear Kalman filter versions. The nonlinearity occurs even if the initial system was linear, as the parameters now considered states usually are in product with the original states, causing a significant nonlinearity.

The extension of the state vector is simple to implement, however, it has significant drawbacks. The most important one is the great variation between the original states and the parameters dynamics. Usually, the extended system is *numerically stiff*, which makes it hard to simulate and raises numerical problems. Another issue is the exponentially increased computational complexity of the KF caused by the increase in the system order, which makes all the matrices of the rank $n+p$ instead of n.

2.4.2 Dual estimation

The second method, which is used more frequently, is *dual estimation*. In this variant, we do not alter the system model or state vector in any way, but implement two parallel Kalman filters. The first filter estimates the system states as usual. The second filter is based on the same system model as the first, but we swap what we consider states and parameters, i.e. the parameters are considered constant states, and the states are considered parameters that vary in time. In every iteration, both filters exchange information about their best estimates of the current states and parameters. Typically, two identical filters (EKF or UKF) are used.

A significant advantage of the n is the relatively lower computational complexity, thanks to working with systems of lower order and better numerical stability, avoiding the issues of joint estimation. However, there may be a disadvantage in not considering correlations between the estimates of the states and the parameters, which may cause slower or biased convergence or even instability, as was suggested in [21].

2.4.3 Example research applications

This section lists several important research papers that serve both as example applications of simultaneous estimation and support the claims and goals of this thesis. They can also be used as resources for more detailed descriptions of the

algorithms described above.

A double-scale and adaptive particle filter-based online parameter and state of charge estimation method for lithium-ion batteries

The publication [22] from 2018 illustrates the use of a dual Particle filter to simultaneously estimate the state of charge and the fixed model structure parameters of lithium-ion batteries. Figure 2.2 shows the scheme of the algorithm. Using this method, the authors were able to gain an algorithm capable of tracking the actual state of the charge of the battery while simultaneously adjusting the imprecise initial guess of the model parameters throughout the battery life cycle.

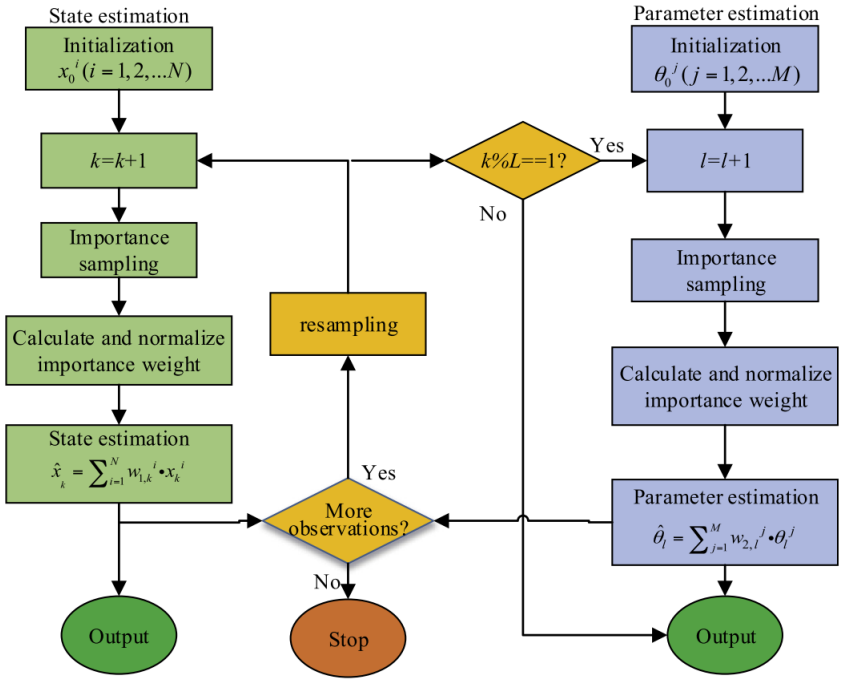

Fig. 2. Flowchart of the proposed double-scale D-PF estimator.

Figure 2.2: The scheme of the dual estimation of a lithium-ion battery equivalent circuit and its state of charge. Taken from [22].

Joint unscented Kalman filter for state and parameter estimation in Managed Pressure Drilling

The article [23] from 2013 describes a typical implementation of the joint Unscented Kalman filter for the estimation of states and parameters in a dynamic system. The example application is controlled high-pressure drilling for undersea deposits of raw materials. The authors used a hydraulic model of the entire system and a state vector extended by the model parameters. In several simulation examples, they illustrate the accuracy and precision of the estimation. Figure 2.3 shows the hydraulic scheme of the system and the simulation process.

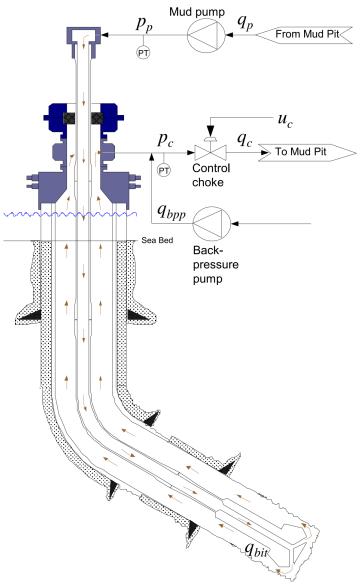

Fig. 1. Schematic of an MPD system, courtesy of Glenn-Ole Kaasa, Statoil.

(a)

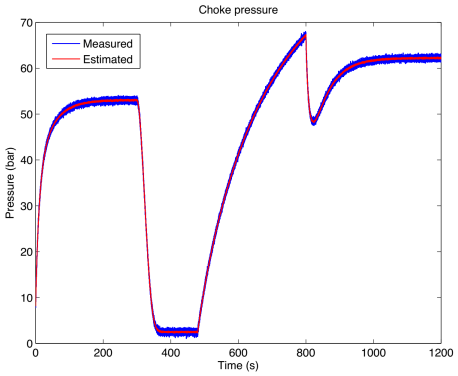

Fig. 3. Measured and estimated choke pressure.

(b)

Figure 2.3: Hydraulic scheme of the controlled high-pressure drilling process (a) and the head pressure simulation process (b). Taken from [23].

A Bayesian adaptive ensemble Kalman filter for sequential state and parameter estimation

In the paper [24] from 2018, the authors use dual estimation with a so-called *Ensemble Kalman filter* (EnKF), which is another modification of the UKF, and in both simulation and real-data experiments, they compare three different ways for the probability density function representation - data grid, normal distribution, and an ensemble of particles. First, they target high-dimensional problems, where other methods typically fail.

Lithium-Ion Battery Parameters and State-of-Charge Joint Estimation Based on H-Infinity and Unscented Kalman Filters

In the article [25] from 2017, the authors developed and successfully tested two UKF modifications (HIT-UKF and PSO-UKF) for te joint estimation of the lithium-Ion battery state of charge (SoC) and the level of degradation, which is a very actual task. They conclude that while the PSO-UKF method gives better results and converges faster, it is more than 200 times more computationally expensive.

2.5 Model approximation methods

When we work with a dynamic system in the form of (2.9), there is a question of finding the structure of the function f based on the measured data. Generally, this task falls into the field of *dynamic system identification* [5, 26].

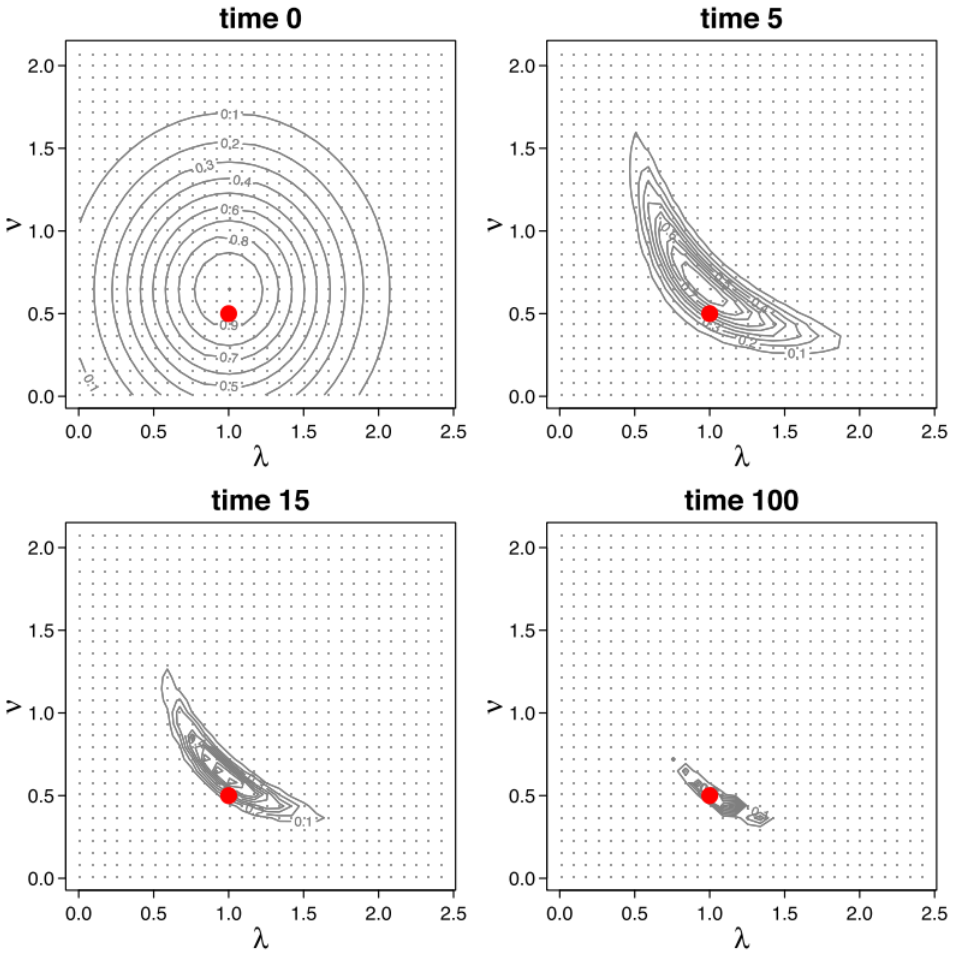

FIG. 6. For data simulated from the Lorenz-96 model (see section 4b), contours of the joint posterior distribution of the parameters λ and ν at time points: (top left) 0, (top right) 5, (bottom left) 15, and (bottom right) 100. The contour values are normalized to yield a maximum posterior density of 1 at each time point. The true value of $(\lambda, \nu) = (1, 0.5)$ is indicated by the red dot.

Figure 2.4: Convergence of the estimate of the posterior probability density function. Taken from [24].

In a case where acquiring the function based on the knowledge of the system using one of the general analytical modelling methods is not possible, we may choose to use one of many general approximation methods to construct a fitting model of the function based on measured data, usually the inputs and outputs of the system. As we mentioned earlier, these methods can be divided into *local* and *global* approximation methods. In the context of machine learning, these groups are called *lazy learning* and *eager learning* [5], where the difference is described based on the willingness of the method to generalise new information. Generally, we can say that local (lazy) methods are better suited for applications where we require adaptiveness and constantly acquire new data across the entire reachable state space of the system, which is exactly the situation that we are dealing with. The other advantages of local approximation methods may be the higher stability and the sometimes easier interpretation for humans.

These are the reasons that we decided to mainly use local approximation methods, specifically local linear models as these bring other benefits which will become clearer in later chapters.

When using the local approximation method to model a dynamic system, we try to approximate the general form of the model (2.22) or (2.23) using a predefined model structure in a limited part of the entire state space. There is a so-called kernel function assigned to every local model that determines the area of validity of the local model. Various kernel functions are available, usually we choose functions that are symmetrical, with single maximum at the centre of the validity area, approaching zero outside. Most commonly, we choose the Gaussian function because of its smoothness and the ability to acquire its derivative analytically. The kernel functions are usually defined using the *Mahalanobis distance* metric over the state space [27]. The *kernel* function is often called the *validity* function, *basis* function, or the *weight* function.

Figure 2.5 demonstrates the approximation of a one-dimensional nonlinear function using local linear modes with Gaussian weight functions.

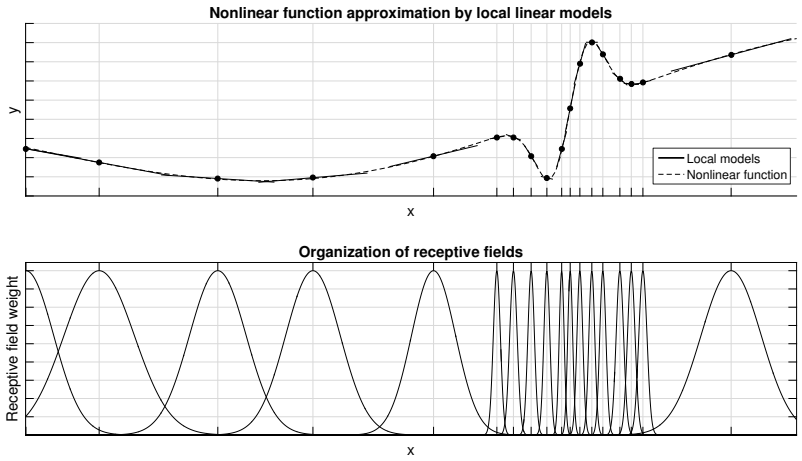

Figure 2.5: An example of approximation of a one-dimensional nonlinear function using local linear models with Gaussian weight functions using the RFWR algorithm. Taken from [28].

The structure of the local model is usually linear or at least linear in parameters, and therefore most approximation methods utilise some variant of the *Least Squares method* [5]. Various approximation methods differ mainly in the algorithm for the placement and shape optimisation of the validity functions. See [28] or [29] for more details. In Figure 2.6, we can see an example of the approximation of a two-dimensional nonlinear function using the RFWR (Receptive Field-weighted Regression) method, which uses a gradient optimisation method of a custom criteria function for the weight function, which it calls the receptive field, size and shape optimisation and, in Figure 2.7, the same function approximated using the LOLIMOT (Locally Linear Model Tree) method, which uses heuristic rules to place the new model and splits the state space using orthogonal cuts.

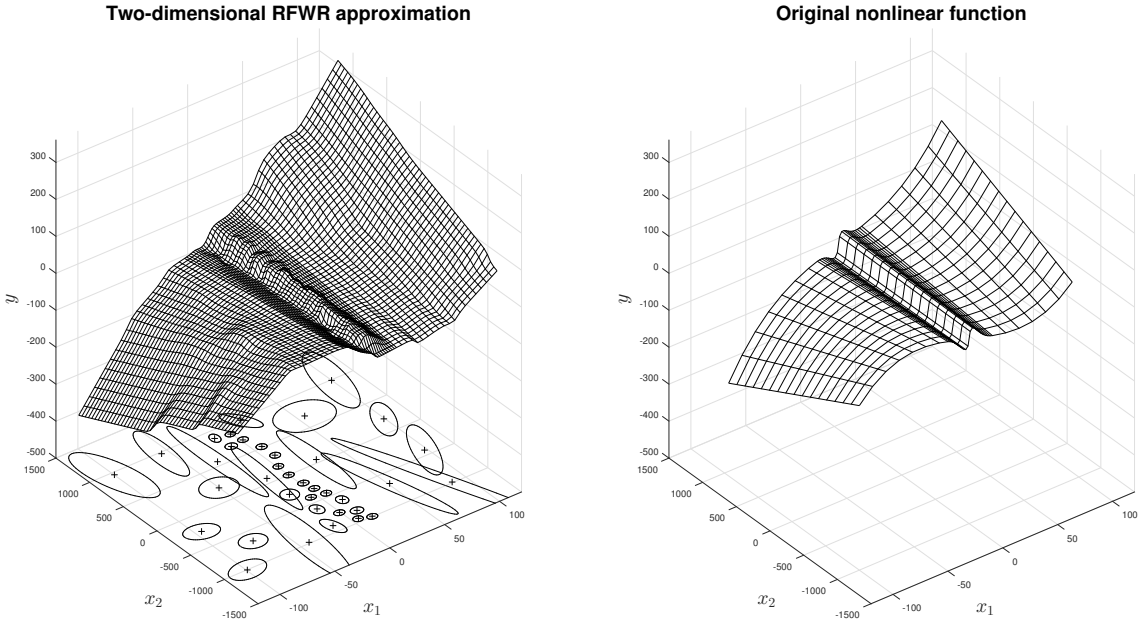

Figure 2.6: An example of the approximation of a two-dimensional nonlinear function using local linear models with Gaussian weight functions using the RFWR algorithm. Taken from [28].

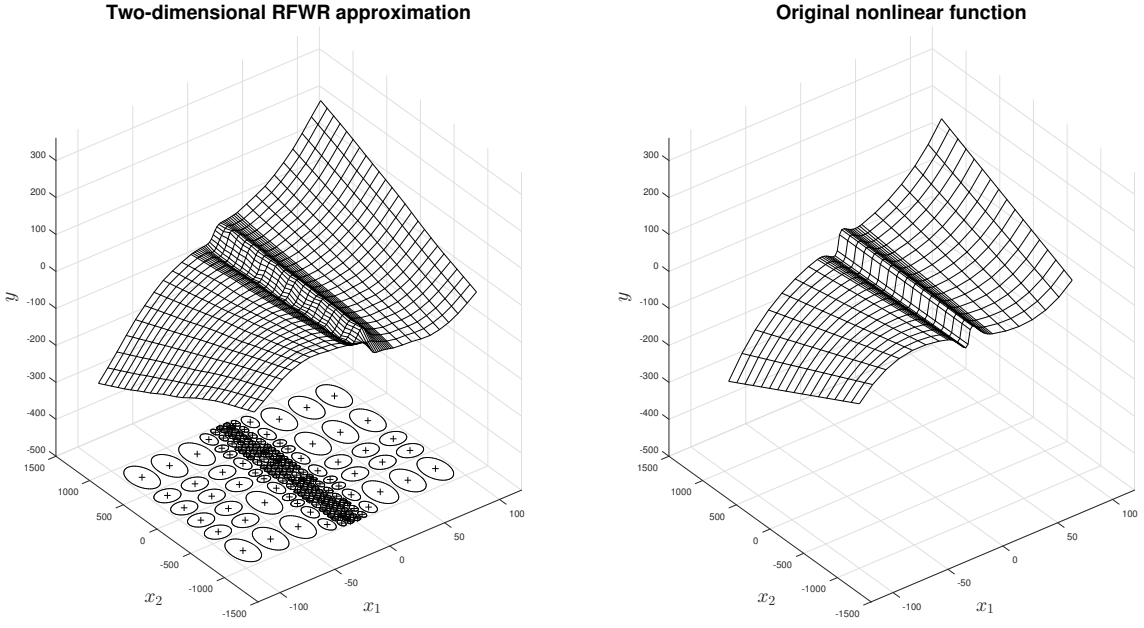

Figure 2.7: An example of the approximation of a one-dimensional nonlinear function using local linear modes with Gaussian weight functions using the LOLIMOT algorithm. Taken from [28].

2.5.1 Receptive field weighted regression

In this thesis, we will focus on the RFWR method with the further experiments. The original algorithm was first presented in [29] and then expanded for control

applications and higher-dimensional problems in [30].

A significant advantage of the RFWR method is that, thanks to the gradient optimisation of the receptive fields, it has the ability to approximate complex shapes using a relatively low number of local models. However, it has several tuning parameters which need to be set correctly, otherwise the approximation process may be unstable, imprecise, or on the contrary lead to overfitting.

A detailed description of the algorithm can be found in [29, 31, 32]. Here, we will only briefly describe the process to give an overview of the methods we will build upon in later chapters. As an example, we chose a simple scalar function (2.46) to approximate, but it can be applied in a similar way on whichever model described in the previous sections, for example (2.22) or (2.23).

$$y = f(\boldsymbol{x}) \tag{2.46}$$

The function f will be approximated (or replaced) by local models in the form of (2.47).

$$\hat{y}_i = \tilde{\boldsymbol{x}} \boldsymbol{b}_i \tag{2.47}$$

with

$$\tilde{\boldsymbol{x}} = \left[(\boldsymbol{x} - \boldsymbol{c}_i)^T, 1 \right] \tag{2.48}$$

where $\tilde{\boldsymbol{x}} \in \mathbb{R}^{n+1}$ is the input of the local model, $\hat{y}_i \in \mathbb{R}$ is the output of the local model (and the estimate of the value of the function f), $\boldsymbol{x} \in \mathbb{R}^n$ input of the function f, $\boldsymbol{c}_i \in \mathbb{R}^n$ the location of the centre of the validity function in the input space, and $\boldsymbol{b}_i \in \mathbb{R}^{n+1}$ the parameters of the local model i. Note that the local model structure (2.47) copies the structure of the first-order Taylor polynomial approximation of the function (2.46), it is a linear model, and the unity added to the vector of inputs corresponds to the constant coefficient of the Taylor polynomial (bias).

There is a validity function assigned to each local model that defines its weight (validity) w_i based on (2.49)

$$w_i = e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c}_i)^T \boldsymbol{D}_i (\boldsymbol{x}-\boldsymbol{c}_i)} \tag{2.49}$$

where $\boldsymbol{D}_i = \boldsymbol{M}_i^T \boldsymbol{M}_i$ is a symmetric positive definite matrix that induces the size and shape of the validity function of the local model based on the Mahalanobis distance [27] and $\boldsymbol{M}$ is a positive upper triangle matrix. In the algorithm, we usually work with $\boldsymbol{M}$ instead of $\boldsymbol{D}$ for numerical stability reasons [29].

At any given point in time, we may want to use the set of local models to produce the current best estimate of the function f approximation. To calculate the estimate $\hat{y}(\boldsymbol{x}_q)$ at any given point $\boldsymbol{x}_q$ in the input space, we use the weighted average of the local models supplied with the same input according to (2.50). $\boldsymbol{x}_q$ is often called

the *query point* of the output estimate.

$$\hat{y}(\boldsymbol{x}_q) = \frac{\sum_{i=1}^{I} w_i(\boldsymbol{x}_q)\hat{y}_i(\boldsymbol{x}_q)}{\sum_{i=1}^{I} w_i(\boldsymbol{x}_q)} \tag{2.50}$$

with I the number of local models.

The set of local models (total number, model parameters, and kernel parameters) is gradually adjusted with each new available data point $(\boldsymbol{x}_k\,, y_k)$ at each time step k.

The existing local model parameters are optimised using the recursive squares (RLS) method. First, the estimate covariance matrix $\boldsymbol{P}_{k-1}$ is updated according to (2.51), and then the parameters $\boldsymbol{b}_{k-1}$ are updated using (2.52).

$$\boldsymbol{P}_k = \frac{1}{\lambda}\left(\boldsymbol{P}_{k-1} - \frac{\boldsymbol{P}_{k-1}\tilde{\boldsymbol{x}}_k\tilde{\boldsymbol{x}}_k^T\boldsymbol{P}_{k-1}}{\frac{\lambda}{w_{i,k}} + \tilde{\boldsymbol{x}}_k^T\boldsymbol{P}_{k-1}\tilde{\boldsymbol{x}}_k}\right) \tag{2.51}$$

$$\boldsymbol{b}_k = \boldsymbol{b}_{k-1} + w_{i,k}\boldsymbol{P}_k\tilde{\boldsymbol{x}}_k\left(y_k - \tilde{\boldsymbol{x}}_k^T\boldsymbol{b}_{k-1}\right) \tag{2.52}$$

In addition to the quantities described above, there is also a forgetting factor $\lambda \in (0; 1\rangle$ used in (2.51) to favour more current data points, thus labelling the method as adaptive. With λ convergent with 1, the effect of forgetting, and thus the adaptivity speed, decreases. Usually, λ is set between 0.95 and 0.999, depending on the specific application. [32].

The spatial distribution of the local models is also updated with every new data point. The update process follows the gradient optimisation method of the weighted quadratic criterion function (2.53).

$$\boldsymbol{J}(\boldsymbol{x}, \boldsymbol{M}_{i,k}) = \frac{\sum_{i=1}^{I} w_{i,k}\left(y_k - \hat{y}_{i,k}\right)^2}{\sum_{i=1}^{I} w_{i,k}} \tag{2.53}$$

where $\boldsymbol{M}_{i,k}$, as mentioned above, is the upper triangular decomposition of the distance-inducing matrix $\boldsymbol{D}_{i,k}$ of the i-th local mode at the time step k.

Note that the value of the criteria function $\boldsymbol{J}$ is a function of $\boldsymbol{M}$ because the weight w is also a function of $\boldsymbol{M}$ when calculated through (2.49).

Then, the gradient optimisation of the criteria $\boldsymbol{J}$ to adjust $\boldsymbol{M}$ follows (2.54).

$$\boldsymbol{M}_{i,k+1} = \boldsymbol{M}_{i,k} - \alpha\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{M}_i} \tag{2.54}$$

where α is a damping factor.

The true gradient term in (2.54) is difficult to calculate analytically, as the update is performed incrementally and the entire data set is not available. As described in more detail in [29], the gradient term is calculated stochastically using an incremental implementation of leave-one-out cross-validation.

The final, similarly important part of the algorithm is the ability to add new and prune unnecessary local models according to simple rules:

1. If no existing local model has weight $w > w_{gen}$ when a new data point is acquired, a new model is created with its centre at that point in the input space.
2. In contrast, if more than one local model has weight $w_i, w_j > w_{prun}$, the model covering a smaller area is pruned.

Setting the parameters w_{gen} and w_{prun} can greatly influence the final number of local models. w_{gen} is usually set to 0.01, w_{prun} to 0.7. Setting both values to be larger leads to more models being used, although, it may lead to overfitting in extreme cases.

2.5.2 Example research applications

This section lists several important research papers that serve both as example applications of local approximation techniques and describe the algorithms used in this thesis in more detail.

Constructive Incremental Learning from Only Local Information

The paper [29] from the authors S. Schaal and Ch. G. Atkeson was the first to introduce the RFWR method mentioned in the Section 2.5. The authors base their research on their previous papers [31, 32] which deal with identification algorithms for local linear models and their use in control applications.

In Figure 2.8, we can see a comparison of the RFWR method with a neural network using sigmoidal activation functions. It shows that, given the same data set, the RFWR method is able to achieve a better fit. In particular, the global approximating network is not able to cope with an expanding data set after being trained for only a part of it.

Later, the author followed up with articles [33] and [30] that dealt with the RFWR method named Locally Weighted Projection Regression (LWPR) primarily aimed at stochastic systems with a very high number of dimensions in the state space.

Locally Weighted Regression Pseudo-Rehearsal for Online Learning of Vehicle Dynamics

Article [34] from 2019 utilises the LWPR method as a pre-learning step to adapt a global approximating neural network. The LWPR step is supposed to prevent the neural network from naively adapting to new, noise-corrupted data points and quickly forgetting the whole model when incrementally added data points come only

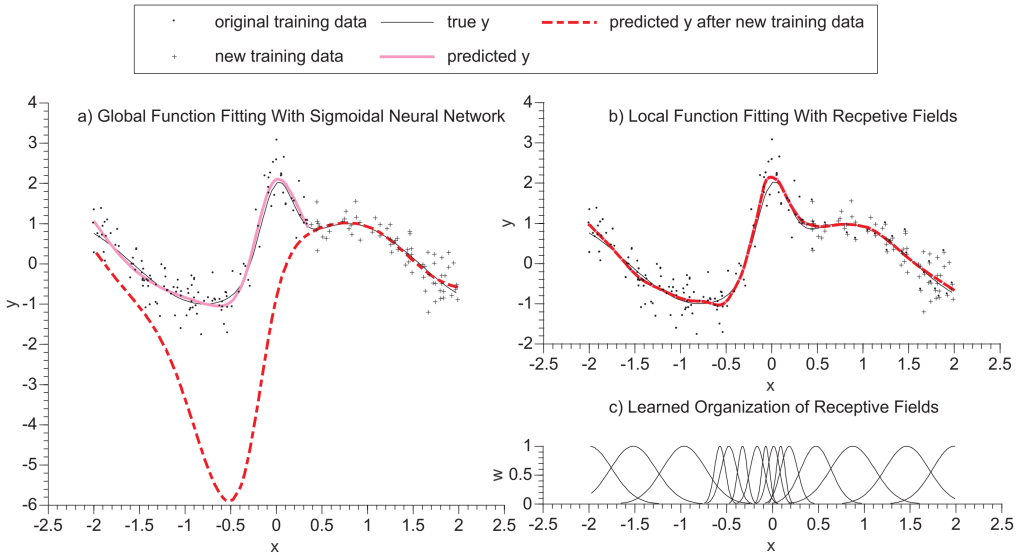

Figure 1: a) Results of function approximation of the function $y=\sin(2x)+2\exp(-16x^2)+N(0,0.16)$ with a sigmoidal neural network, b) results of function approximation by a local receptive field-based algorithm, fitting locally linear models in each receptive field (note that the data traces "true y", "predicted y", and "predicted y after new training data" largely coincide), c) the organization of the (Gaussian) receptive fields of b) after training.

Figure 2.8: Comparison of approximations of a nonlinear function using the RFWR method and a sigmoidal neural network. Taken from [29].

from a small area of the state space. The LWPR method uses the raw measured data and then the neural network is presented with simulated data points based on the LWPR model. Using such an interesting synthetic data generator enables the author to create a data set with a different placement of data points than the original one, which helps avoid overfitting in overexposed areas of the state space while preserving the accumulated knowledge. This application is a great example of using the advantages of both local and global approximation.

The authors apply the method to the modelling of the dynamics of a terrain four-wheeled vehicle (laboratory model). They were able to achieve the stable incremental adaptation of the neural network with the LWPR pre-learning to changing system dynamics more successfully when compared to a neural network alone.

A Hierarchical Bayesian Linear Regression Model with Local Features for Stochastic Dynamics Approximation

The article [35] from 2018 presents a combination of global stochastic models with local methods based on RFWR learning to model typical laboratory mechanical systems with stochastic inputs. The authors deal with the RFWR as a method that inspired the development of other highly specialised methods. Mainly, they aim at modelling the dynamic systems for optimal control design, especially in the *reinforcement learning* framework.

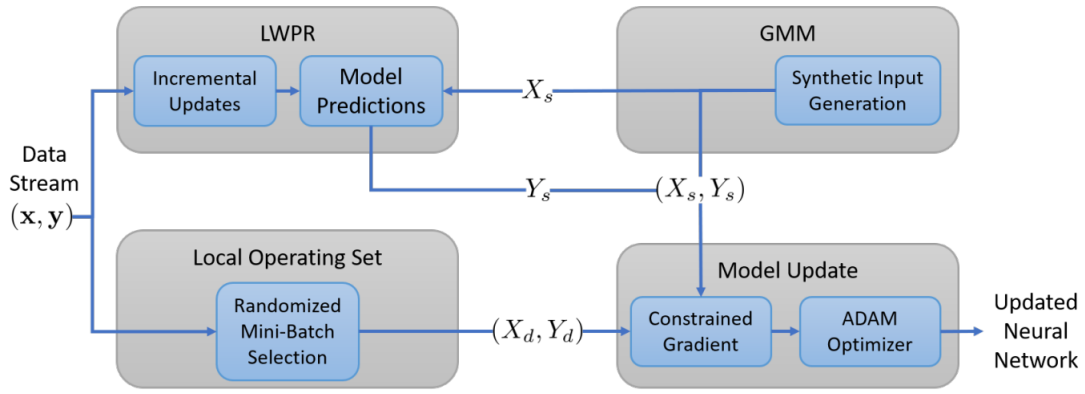

Fig. 1: LW-PR^2 Algorithm. The GMM produces synthetic input points which are combined with predictions from LWPR to create synthetic training pairs. These are combined with randomized mini-batches created from recently collected data in order to compute the constrained gradient update.

Figure 2.9: Block scheme of the algorithm - LWPR forming a database for a neural network to approximate vehicle dynamics. Taken from [34].

Incremental Receptive Field Weighted Actor-Critic

The paper [36] from 2013 describes the use of the RFWR method with dynamic systems with the continuous state space to model the *V-function*, a necessary part of the reinforcement learning framework to design an optimal control law.

3 Formulation of the thesis goals

The significance and need for the simultaneous estimation are outlined in Chapter 1 in relation to the typical control design process and related tasks. In Chapter 2, we described several existing variants of methods for the individual as well as the simultaneous estimation of states and parameters of dynamic systems, and local linear models of dynamic systems.

These methods are functional, however, none of them are universally applicable and there is no clear metric or methodology for the correct method choice. In practice, developers and scientists often choose the approach with which they have the most experience, regardless of the other methods. The existing methods also have significant disadvantages as described in Chapter 2, especially requiring intuition-based tuning of many application specific parameters, the implementation is often very layman unfriendly and deals with the estimation of the states and parameters (in terms of parameter tuning criteria) separately, even though they are closely intertwined.

Based on previous experiments and research, the goal of this thesis is to contribute to the solution of the dual estimation problem by developing a hybrid dual estimation method based on local approximation (RFWR for example) for the parameter estimation and the Kalman filter for the state estimation, instead of using the same algorithm for both estimation tasks. One of the main issues to deal with is the correct (preferably automated or adaptive) tuning of the Kalman filter's process noise covariance matrix so that the dual estimation is stable and performs well, at least for a specific, limited case of a typical mechatronic system, defined in Chapter 1.

It is important to note that we are not primarily aimed at improving the overall estimation quality, but to simplify the implementation process while keeping the quality good enough. That being said, however, improving the quality of the estimation will be a secondary target.

Specifically, we defined four goals for this research, which correlate with the goals set and approved at the State doctoral exam:

Survey and research of the mutual relationship of typical state and parameter estimation methods for dynamic systems

Chapter 2 describes the relationship of two common tasks that often occur while working with nonlinear dynamic systems - state estimation and parameter estimation. The first part of the thesis is to research, in detail, the most common methods and algorithms used for both these tasks with the goal of finding and exploring the possibilities of their mutual interconnection. We will mainly deal with methods from the Bayesian recursive filter category (variants of the Kalman filter) for the

state estimation task and with local linear approximation methods (LWL, RFWR, LOLIMOT, LWPR, etc.) for the modelling and parameter estimation task.

The interconnection of both tasks can be found, for example, in the imprecision or uncertainty of the estimation of parameters, which also translates into the states and output of the system. This influences the process noise of the system model, which is one of the most important effects that affects the implementation of the Kalman filter or variants. Some parameter estimation methods (such as Recursive least squares (RLS)) directly deal with the parameter estimation imperfection model in the form of Gaussian noise uncertainty. Specifically, questions arise concerning what is the effect of the known parameter uncertainty covariance on the process noise covariance in case of a linear system or what other effects influence it.

The output of this step will be an analysis of the signal and parameters occurring in both tasks and determining which of them may be used to better tie the said algorithms together.

Modification of the RFWR method for its use on typical mechatronic systems

The original algorithm described in [29] was successfully used in several practical applications mentioned in Section 2.5.2, however, it almost exclusively involves offline data processing. As Chapter 2 describes its potential in local linear approximation, the original algorithm has several disadvantages that limit its capabilities when applied in the context of the three tasks leading to the control design, especially on typical mechatronic systems.

Namely, these are:

- problematic stability while incrementally adapting to new data, e.g. when the data is asymmetrically localised with respect to the local model centre,
- vulnerability to overly local data,
- the validity function dimension is determined by the order of the system,
- possible inconsistencies between neighbouring local models.

The goal of this thesis is to modify the original RFWR algorithm in a way that it does not suffer from the above-mentioned issues and would be possible to tie the algorithm with a state estimation method resulting from the first goal. Specifically, these modifications will be made:

- combining local and global parameters to lower the dimension of the local models and validity functions,
- allowing separate dimension orders for the local models and the validity functions,
- improving the adaptation convergence stability while applying the algorithm incrementally,

- reducing the user defined parameters,
- allowing the interconnection if the algorithm with the state estimation methods according to the results of the first goal.

The output of this goal will be a modified adaptive RFWR algorithm for modelling nonlinear dynamic systems in the form of a Matlab library.

Hybrid method for the simultaneous modelling and state estimation of nonlinear dynamic systems

Following up on the previous goals, the third goal of this research is to further develop and implement the results from the first goal into the modified RFWR algorithm created in the second goal, resulting in a new hybrid method for both the modelling and state estimation of nonlinear dynamic systems. The main task is to achieve a reduction in the number of parameters a user needs to set and tune while simultaneously maintaining or even improving the overall performance of both the modelling and state estimation.

Again, the work will result in a Matlab library.

Case study on a real system

The algorithms developed in the second and third goals will need to be tested, first in a simulation and on a real system afterwards, which might reveal some possible shortcomings that may remain hidden when using simulated data (e.g., due to non-white noise or unexpected types of nonlinearity). Both algorithms will be tested throughout the development process on a suitable system (e.g., a rotary inverse pendulum, a magnetic ball levitation system or an automotive actuator, such as a throttle valve or an exhaust gas recirculation (EGR) valve) to ensure that the results are applicable in the real world.

These experiments will serve both as proof that the algorithms are working as intended and as an example implementation for a possible new user who intends to use the results in their own application or research.

4 Kalman filter with uncertain parameters

In this chapter, we deal with the Kalman filter as a well-known tool for the state estimation and tracking of dynamic systems with known parameters. First, in Sections 4.1 and 4.2, we study the algorithm under ideal conditions, following all the assumptions where optimality is guaranteed, and try to study and categorise the various sources of the estimate error and uncertainty which may occur in real applications.

Then, in Section 4.3, we introduce a modification to the state covariance prediction step of the Kalman algorithm, which is better suited for the specific situation where the dominant source of error is the uncertainty of the model parameters. The Section 4.4 further elaborates on the topic, and we suggest a partially empirical method for setting the process noise covariance in this very specific situation.

The last Sections 4.5 and 4.6 expands on the previous one by connecting the modified Kalman filter to a parameter estimation algorithm (RLS) and applying it to a linear model with local validity in the state space of a nonlinear system, and expand the algorithm into higher-dimensional space.

4.1 Kalman filter under ideal conditions

Generally, the KF can work with a system of arbitrary order, however, to make the results more intuitive, we started the research by looking at a first-order system of the simplest form (4.1). The system has a single parameter b, state x and input u. This trivial model may represent many real dynamic systems, such as the charge of a capacitor, the velocity of a point mass in one direction, the temperature of an object, and many others.

$$\dot{x} = bx + u \tag{4.1}$$

For the purpose of Kalman filtration, we need a discrete representation of such a system, which may be acquired by the Euler discretisation method [37], resulting in (4.2).

$$x = \boldsymbol{F}x_{k-1} + \boldsymbol{G}u_{k-1} \tag{4.2}$$

where $\boldsymbol{F} = 1 - T_s b$ and $\boldsymbol{G} = T_s$, where $T_s = 5 \cdot 10^{-3}$ s is the sampling period. We intentionally chose the sampling period as being very short, relative to the system

time constant, so that the discretisation effect is negligible. Nonetheless, this effect would not influence the results as we consider the discrete system (4.2) to be the reference in which we will compare our results to. The continuous version will be useful later.

Figure 4.1 shows sample signals of the system evolving over time. The input u was generated as random steps with both the value and the length of the steps generated from a uniform distribution.

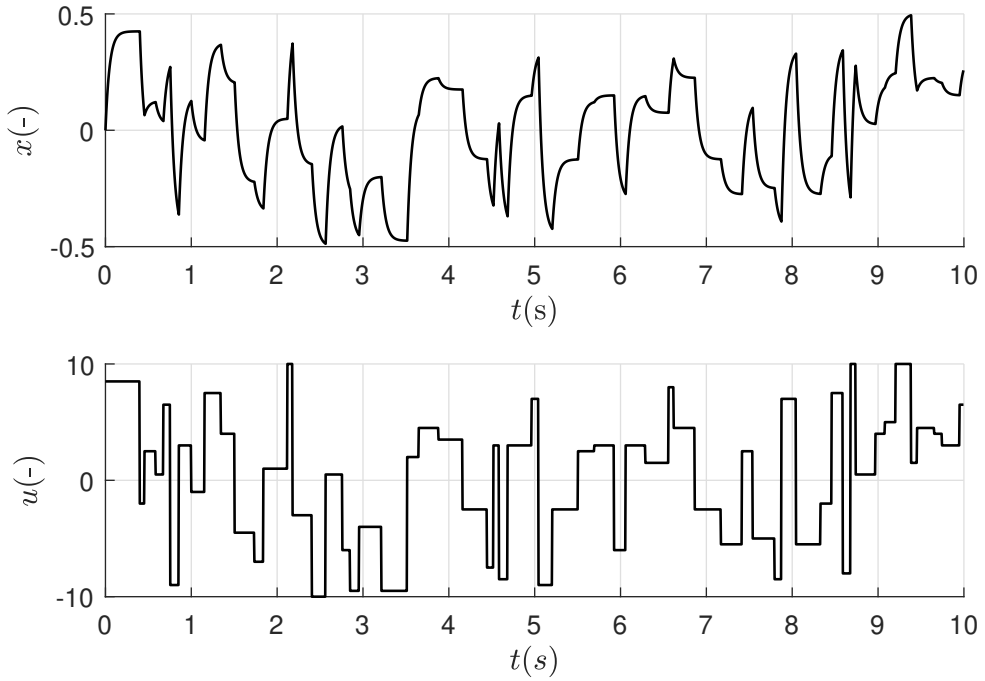

Figure 4.1: Sample signals of the system (4.2) evolving over time. Simulated with $T_s = 5 \cdot 10^{-3}$ s, $b = 20$, the value of u is generated from the uniform random distribution within the interval $\langle -10; 10 \rangle$ and the lengths of the input steps are generated from the uniform random distribution within the interval $\langle 0.05; 0.3 \rangle$ s.

For this very simple system, we implemented the Kalman filter to track the state x, according to Euations (2.34) through (2.38). To study the filter with ideal conditions, Gaussian noise was introduced as w in the input signal u and v in the measurement step according to (4.3) and (4.4).

$$\hat{x}_{k|k-1} = \boldsymbol{F}_k \hat{x}_{k-1|k-1} + \boldsymbol{G}_k u_{k-1} \tag{4.3}$$

$$z_k = x_k + v_k \tag{4.4}$$

with $w_k \sim \mathcal{N}(\boldsymbol{Q}_t, 0)$ and $v_k \sim \mathcal{N}(\boldsymbol{R}_t, 0)$ representing the process and measurement corruption noise, and $\boldsymbol{Q}_t = 9 \cdot 10^{-4}$ and $\boldsymbol{R}_t = 2.5 \cdot 10^{-3}$ are known Gaussian noise variances, respectively.

Since the corruption noise variances and the true state x signal are known in

this case, which is not true in most practical applications, it allows us to study the effect of imprecise $\boldsymbol{Q}$ and $\boldsymbol{R}$ settings on the filter performance. To evaluate the performance, we chose to use a common MSE metric defined as

$$M = \frac{1}{n}\sqrt{\sum_{k=1}^{n}\left(x_k - \hat{x}_{k|k}\right)^2} \tag{4.5}$$

With the above-described set up, we performed and evaluated the MSE for various $\boldsymbol{Q}$ and $\boldsymbol{R}$ settings independently, resulting in Figure 4.2. Each point on the depicted surface represents an individual test with different $\boldsymbol{Q}$ and $\boldsymbol{R}$ settings.

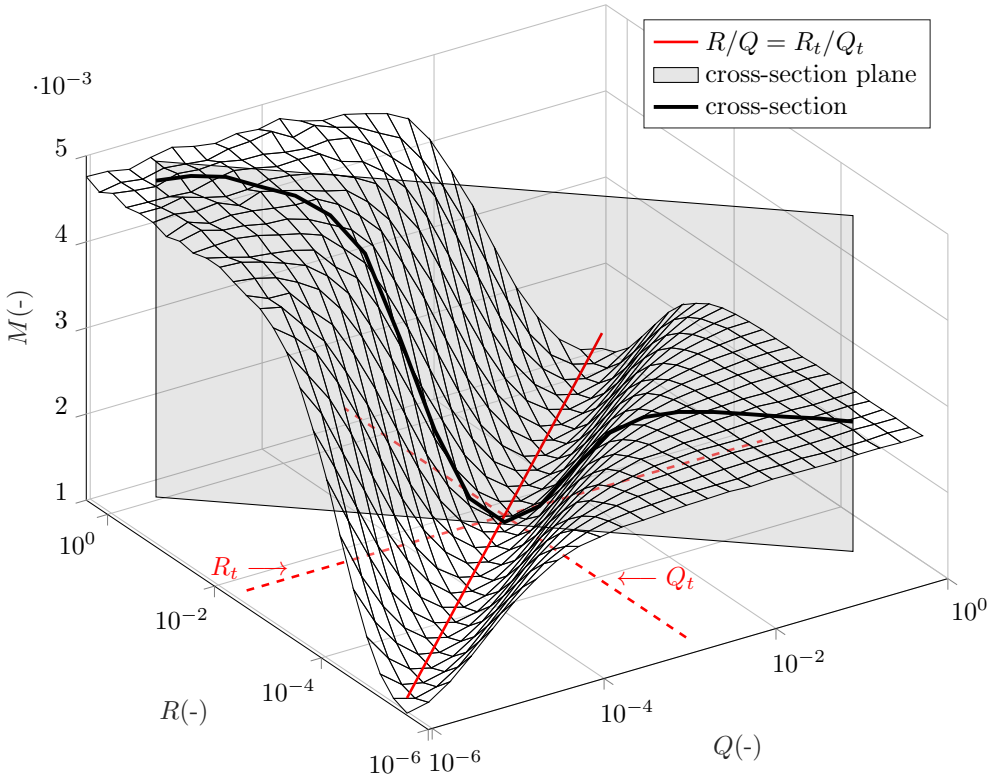

Figure 4.2: The Kalman filter performance measured as an MSE as a function of the $\boldsymbol{Q}$ and $\boldsymbol{R}$ values. Each point on the surface is the average of 50 simulations over a 30 second interval with the same settings. The figure also depicts the expected optimal point, the direction of symmetry and the cross-section plane that we work with in the subsequent simulations.

The shape of the M metric values forms a surface that reveals several important properties, which correspond with theoretical expectations. First of all, the optimal setting is achieved for $\boldsymbol{Q} = \boldsymbol{Q}_t$ and $\boldsymbol{R} = \boldsymbol{R}_t$, as expected. Also, it is known that, for the corresponding states, the performance of the Kalman filter depends on the ratio of $\frac{\boldsymbol{R}}{\boldsymbol{Q}}$, not on the specific values. In fact, we can see that the surface is symmetric along the line $\boldsymbol{R} = \boldsymbol{Q}\frac{\boldsymbol{R}_t}{\boldsymbol{Q}_t}$, which is also shown as a line if both the $\boldsymbol{Q}$ and $\boldsymbol{R}$ axes are

on a logarithmic scale.

The symmetry allows us to study the whole situation in a one-dimensional plot along the axis perpendicular to the optimal line, in effect looking at a cross-section of the surface shown in Figure 4.2. The line perpendicular to the optimal line would represent a hyperbola on a linear scale, allowing us to look for its specific shape in the form of (4.6), which has only one parameter k.

$$\boldsymbol{R} = \frac{k}{\boldsymbol{Q}} \tag{4.6}$$

To determine the parameter, we can specify a single condition as we want it to pass through the optimal point $(\boldsymbol{Q}_t; \boldsymbol{R}_t)$. Expressing from Equation (4.6), we get $k = \boldsymbol{R}_t\boldsymbol{Q}_t$. The resulting plot is shown in Figure 4.3. For better understanding, the plot shows both the $\boldsymbol{Q}$ and $\boldsymbol{R}$ axes, even though $\boldsymbol{R}$ is mapped on $\boldsymbol{Q}$ using (4.6).

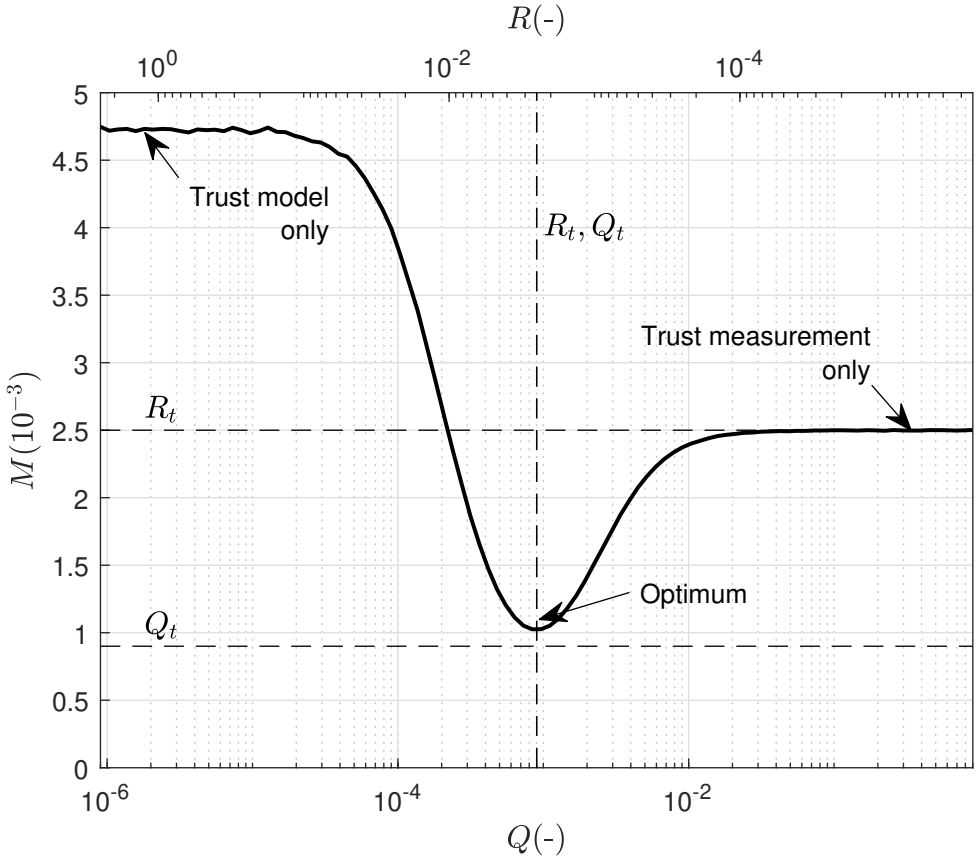

Figure 4.3: The Kalman filter performance measured as an MSE as a function of the $\boldsymbol{Q}$ and $\boldsymbol{R}$ values chosen along the axis of symmetry. The figure depicts the comparison of the performance metric to the true noise variance values and shows both the extreme and optimal cases of various $\boldsymbol{Q}$ and $\boldsymbol{R}$ settings.

First of all, the figure shows the optimal setting of $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$, which overlap on the horizontal axes thanks to the mapping, and corresponds with the best metric M

value acquired through the simulation as expected. The plot may be separated into three regions. For situations where $\boldsymbol{Q} << \boldsymbol{Q}_t$ while $\boldsymbol{R} >> \boldsymbol{R}_t$ the filter basically only considers the prediction step and the measurement is disregarded. In contrast, when $\boldsymbol{Q} >> \boldsymbol{Q}_t$ and $\boldsymbol{R} << \boldsymbol{R}_t$ the filter behaves like only the measurement matters and ignores the prediction value. Only in between those two extremes both the prediction and measurement values complement each other and the filter has the best possible performance.

It is important to note that the interval where the filter performs well is quite wide, which means that it is mostly important to choose the correct order for the $\boldsymbol{Q}$ and $\boldsymbol{R}$ values, fine-tuning them is hard to do in practice and does not bring high benefits.

In addition, since the MSE metric M has a similar meaning to the $\boldsymbol{Q}$ and $\boldsymbol{R}$ variances, it is interesting to compare it to the optimal values. As can theoretically be expected, the value of M converges to the value of $\boldsymbol{R}_t$ with an increasing $\boldsymbol{Q}$ (higher confidence in the measurement). On the other hand, the value of M is much higher than $\boldsymbol{Q}_t$ when the filter regards only the prediction model. This disparity is caused by the accumulation of the estimation error as the filter prediction becomes unstable or drifts away from the true state value when the measurement value is disregarded.

We can infer from the previous statements that even very noisy measurements can be beneficial to a good quality model in terms of long-term stability.

4.2 Sources of errors and uncertainties

Based on our experience and previous research, as well as the current scientific literature [18, 7, 38, 39, 40, 41], it is safe to say that of the two covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ representing the process and measurement noise, respectively, the process noise is the bigger issue. The measurement noise value can usually be estimated, measured, or acquired from the equipment datasheet. For this reason, and for the fact that only the ratio between $\boldsymbol{Q}$ and $\boldsymbol{R}$ matters, we will consider $\boldsymbol{R}$ to be known (at least in its order) throughout the rest of this thesis and focus on the methods on how to deal with $\boldsymbol{Q}$.

Before we discuss how to set $\boldsymbol{Q}$ correctly, we need to address various sources of imperfection and uncertainty that might cause it in the first place. We were able to identify several categories of effects that play a dominant role in practical applications.

The process noise can be caused by:

- a stochastic or an immeasurable input to the system
- a discretisation (or numerical integration) error
- a prediction model imperfection
  - wrong, incomplete, or imprecise model structure

– inaccurate or uncertain parameters

The first item, a stochastic or an immeasurable input to the system, is the most studied case. We can find several algorithms for tuning the $\boldsymbol{Q}$ matrix, for example, in [7]. There is usually nothing that can be done about this issue to diminish its influence, as it is mostly part of the application. Examples are wind blowing, human behaviour, Brownian motion, or quantum effects. In our case, we can consider this issue negligible as we try to model or measure every effect influencing the system that we try to track.

The effect of discretisation can be serious and is often disregarded, however, with most dynamic systems in mechatronics, it can be diminished by using higher order discretisation methods or shorter sampling periods. Again, in our case, this effect can be considered negligible if treated carefully.

Speaking of discretisation, when the Kalman filter is applied to a truly discrete system, not discretised, which contains an integration part, the error of the integration is not only negligible, but truly zero. However, it is advisable to set the corresponding element of the $\boldsymbol{Q}$ matrix to be low, but non-zero, to prevent disregarding the state measurement completely and causing potential instability or long-term drift. An example of such a system may be (4.7), where T_s is the sampling period of the discrete system and k_1 and k_2 are the system parameters.

$$\boldsymbol{x}_k = \begin{bmatrix} 1 & T_s \\ k_1 & k_2 \end{bmatrix} \boldsymbol{x}_{k-1} \tag{4.7}$$

The last item on the list is the prediction model imperfection, which can be further divided into two categories. The first category covers situations where the model structure simply cannot represent the system it is supposed to model well enough. For example, when a linear model is applied on a nonlinear system or when special kinds of nonlinearities, such as friction effects, are not considered in the model structure. This is an issue that is very hard to deal with, and setting the $\boldsymbol{Q}$ matrix correctly is left as an ad hoc implementation. However, in the case of our research where we decided to use local linear approximation methods, it is safe to assume that this effect becomes negligible once the approximation converges.

That leaves us with the second category of the last item - inaccurate or uncertain model parameters. This issue is often regarded as the same problem as the imperfect model structure and receives very little scientific attention. We were able to locate only a single paper [42] dealing with a similar problem, however, especially in the case of local models, this issue becomes the dominant effect causing the process noise. We also believe that this issue is the dominant effect far more often than not, except when stochastic or immeasurable effects are present. Furthermore, as opposed to most of the other issues, we will show, in the following sections, that the parameter uncertainty can be dealt with and offers a unique way to set the $\boldsymbol{Q}$ matrix accordingly.

4.3 Reformulating KF for uncertain parameters

In the previous section we summarised the effects causing the process noise and determined that the parameter uncertainty is the dominant effect that we should deal with. Similarly, as the Kalman filter algorithm deals with imprecisions in a stochastic way, we will consider the parameter uncertainties stochastic in the same Gaussian way. The reason for this assumption is that most parameter estimation algorithms use the Least squares method or one of its variants to determine the parameters, and this allows for the estimation of the uncertainty. We will expand on this idea in the following section.

First, consider an autonomous one-dimensional discrete system (4.8).

$$x_{k+1} = bx_k \tag{4.8}$$

where b is the only system parameter replacing the state matrix F, meaning, in this case, $F = b$.

The Kalman filter assumes the prediction model in the form (4.9), where w_k is the process noise, but it can also be seen as the complement to the model to make it fit perfectly.

$$x_{k+1} = bx_k + w_k \tag{4.9}$$

Assume that we use an estimated parameter $\hat{b}_k$ with an uncertainty represented by its variance $S_k = var(\hat{b}_k)$, meaning that the true value of $b \sim \mathcal{N}(S_k, \hat{b}_k)$. We can reformulate the model (4.8) into (4.10)

$$x_{k+1} = (\hat{b}_k + s_k)x_k \tag{4.10}$$

with $s_k \sim \mathcal{N}(S_k, 0)$ being the parameter noise in the same sense as the process noise w_k, meaning it is the complement to the perfect value. With this in mind, we can further expand the model according to (4.11).

$$\begin{aligned} x_{k+1} &= (\hat{b} + s_k)x_k \\ &= \hat{b}x_k + s_k x_k \\ &= \hat{b}x_k + w_k \end{aligned} \tag{4.11}$$

where $w_k = s_k x_k$.

This model corresponds to the original KF model (4.9) with the additional fact that the process noise w_k is now a function of the state x_k. This is an important and also expected fact when we consider the uncertainty of the parameters as the dominant source of the process noise. Figure 4.4 shows such a situation for a very simple,

one-dimensional system and graphically reveals that the assumption we arrived at - that the process noise depends on the actual value of the state, is correct.

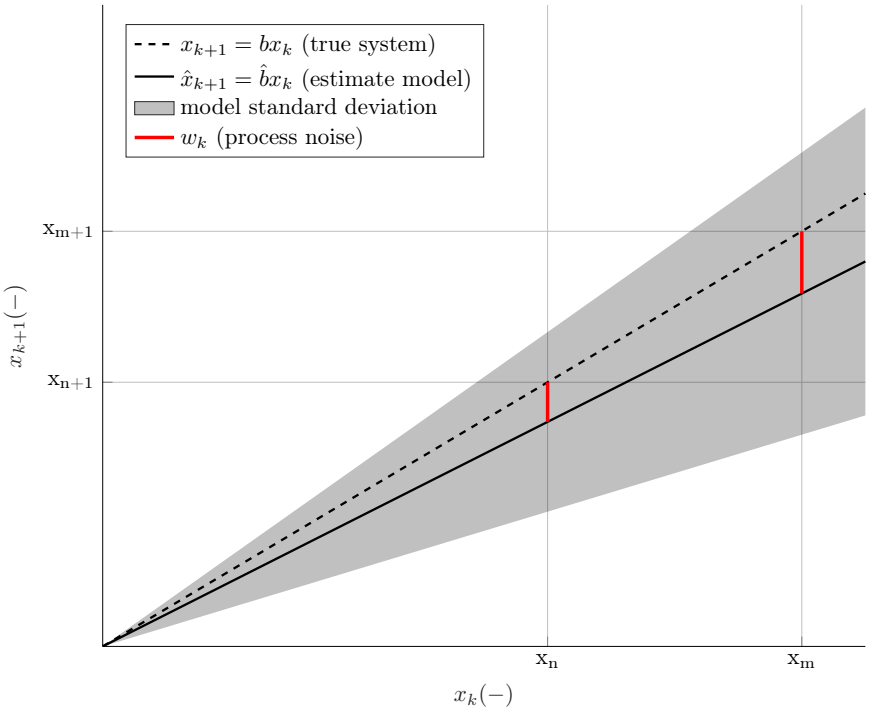

Figure 4.4: A one-dimensional discrete system with a single parameter b and a comparable estimated model of the system depicting the expected model standard deviation and resulting process noise.

The process noise covariance is used in the regular Kalman filter algorithm to predict the value of the state estimate covariance according to (2.35), assuming $w_k \sim \mathcal{N}(\boldsymbol{Q}_k, 0) \implies \boldsymbol{Q}_k = var(w_k)$. From this, we can expand the process noise covariance matrix $\boldsymbol{Q}_k$ as (4.12).

$$\begin{aligned} \boldsymbol{Q}_k &= var(w_k) \\ &= var(s_k x_k) \\ &= \boldsymbol{S}_k x_k^2 \end{aligned} \tag{4.12}$$

Using this expansion, we can modify the state estimate prediction step (2.35) as (4.13). This specific expansion only works for one-dimensional systems, we will deal with its expansion to higher dimensions in a later section, after experimentally verifying the modification.

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^T + \boldsymbol{S}_k x_k^2 \tag{4.13}$$

The estimate covariance prediction step in the KF basically expands the covari-

ance by an amount attributed to the process noise, thus keeping track with the state prediction step. First of all, (4.13) assumes that all the process noise comes from the parameter uncertainty and is covered by the term $\boldsymbol{S}_k x_k^2$, which depends on the actual state x_k as opposed to the regular term $\boldsymbol{Q}_k$. To verify that all the process noise indeed comes from the parameter uncertainty, we performed a stochastic simulation using the same system as in the previous experiment with $\boldsymbol{Q}$ set precisely to $\boldsymbol{R}_t$ and the covariance prediction made using (4.14). Since the system is not changing, we set $\boldsymbol{S}$ and $\boldsymbol{Q}$ as being constant during each individual simulation, thus dropping the subscript k in this and the subsequent equations throughout the rest of the chapter.

$$P_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^T + \boldsymbol{Q} + \boldsymbol{S} x_k^2 \tag{4.14}$$

We performed a number of simulations for each combination of $\boldsymbol{S}$ and $\boldsymbol{Q}$, while randomly picking the parameter estimate $\hat{b} \sim \mathcal{N}(B, \hat{b}0) \implies B = var(\hat{b})$, for each simulation. By averaging the KF performance for each $(\boldsymbol{S}, \boldsymbol{Q})$ combination, we arrive at Figure 4.5.

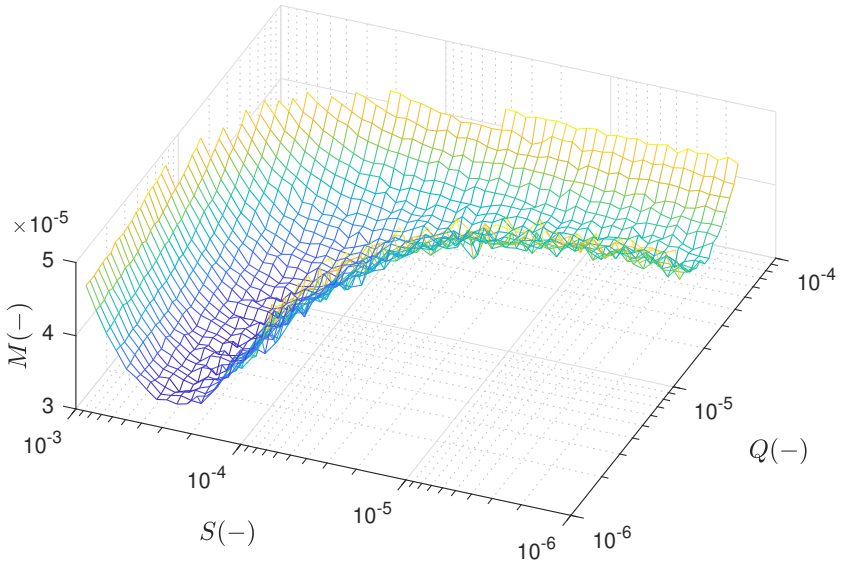

Figure 4.5: The simulation compares the effect of the original ($\boldsymbol{Q}$) and modified ($\boldsymbol{S}$) KF estimate covariance prediction step. Each point on the surface represents an average from 500 simulations with the randomly picked parameter b with a 10% standard deviation. A part of the surface with poor KF performance (high M values) is omitted to make it visible.

The simulation proves that both ways of representing the process noise are viable when used individually (we arrive at the optimal results when one of the parameters is relatively negligible) and that there is no clear optimum for a combination of both. Furthermore, the simulation suggests that the state-dependent prediction

variant may have slightly better performance.

Since we can be sure that the combination of both methods is not meaningful, we can now study and compare both methods independently in mode detail, in Figure 4.6. This figure corresponds to Figure 4.3, which represents a similar simulation but compares the two methods directly.

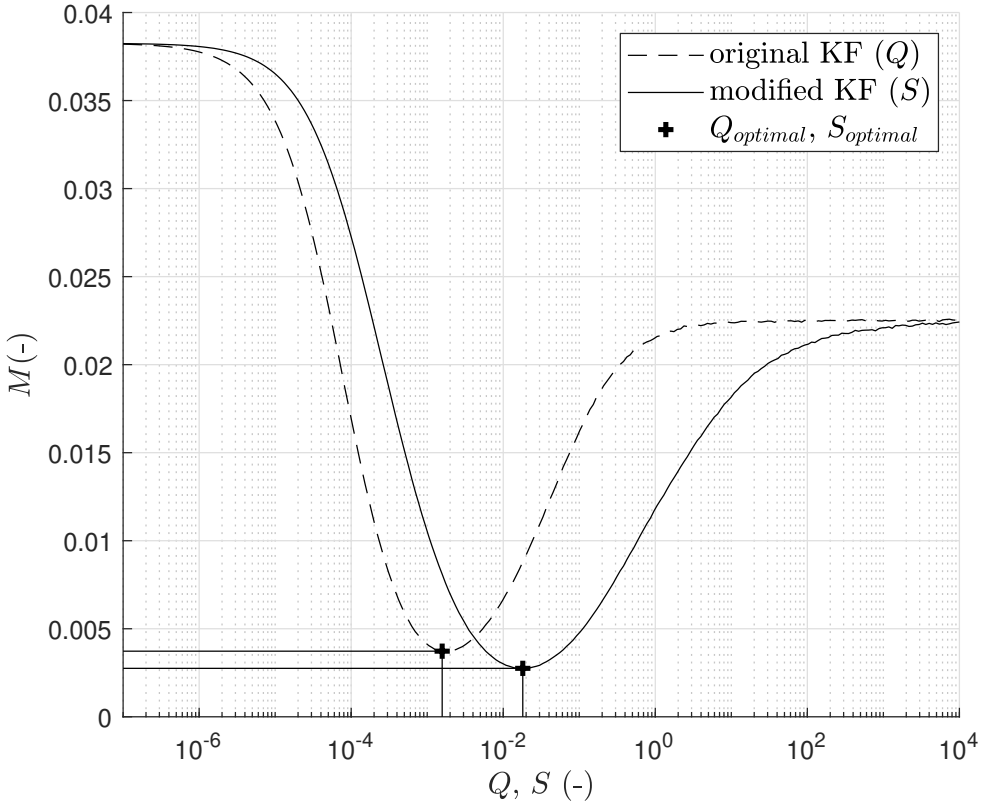

Figure 4.6: The simulation compares the effect of the original ($\boldsymbol{Q}$) and modified ($\boldsymbol{S}$) KF estimate covariance prediction step applied individually. Each point in the simulation represents an average of 25 simulations with the randomly picked parameter b with a 10% standard deviation. The results show that the modified version outperforms the original KF algorithm by about 35% in this specific scenario with uncertain parameters, with $\boldsymbol{S}_{optimal} = 2.75 \cdot 10^{-}3$ and $\boldsymbol{Q}_{optimal} = 3.73 \cdot 10^{-}3$.

The results show that with the correct setting, the modified state-dependent method for the KF estimate covariance prediction according to (4.13) performs comparably or even marginally better when set correctly as opposed to the original prediction method (2.35) in situations where the parameter uncertainty is the dominant cause of the process noise. The question that remains is how to set $\boldsymbol{S}$ correctly.

4.4 Setting the process noise covariance

In this section, we try to find the answer to the question of finding or estimating the optimal setting for the process noise covariance $\boldsymbol{S}$ when using the modified prediction method (4.13) introduced above.

In Section 4.3 we made the assumption that the imprecise estimation or stochastic uncertainty of the model parameters can be modelled as white Gaussian noise variance $\boldsymbol{S}$. Obviously, this is a simplification to a certain degree and this kind of assumption cannot always be met, however, there are several reasons to make it. First of all, it makes it possible to work with the uncertainty represented by a single number (or a matrix in a multidimensional case) and since it is the same assumption the Kalman filter framework makes (and guarantees optimality for), it makes it compatible with the rest of the KF algorithm. Second, often while estimating the model parameters, we also acquire a measure of how well the model fits the data, which may lead to an estimate of the correct setting for the $\boldsymbol{S}$ value. Especially in our case, when we chose to use the local linear approximation using LS and RLS for the parameter estimation, the model comes directly with the parameter uncertainty estimate. For the least squares method (LS), the uncertainty covariance estimate is known to be calculated according to (4.15),

$$var(\hat{b}) \approx \sigma^2 \left(X^T X\right)^{-1} \approx e^2 \left(X^T X\right)^{-1} \tag{4.15}$$

where σ^2 is the estimate error variance, which is theoretically unknown, hence, it is replaced by the estimate residual variance e^2 and $X^T X$ is the so-called cofactor matrix based on the data matrix X used for the LS estimate. See [43, 44, 45] for further details on the topic. Also, this parameter variance estimate corresponds to the P matrix in the RLS algorithm described in Section 2.5, specifically Equations (2.51) and (2.52), where the variance-covariance estimate is generalised for a multidimensional case and calculated iteratively, which will become useful later.

We may come to the conclusion that the best estimate for the value of $\boldsymbol{S}$ is $\boldsymbol{B} = var(\hat{b})$, however, we were able to disprove this conclusion experimentally, since an analytical solution to this issue does not exist. The proof of this claim will become clear from experiments in the subsequent sections.

Furthermore, we introduced an empirical formula which proved to be a very good estimate of the value of $\boldsymbol{S}$. This formula (4.16) is derived from the assumption that $\boldsymbol{S}$ must depend on the long-term variance w.r.t. zero of the signal we are tracking.

$$\hat{\boldsymbol{S}} = \frac{1}{2}\boldsymbol{B} E\left[x^2\right]^{-1} \tag{4.16}$$

Starting from the same simulation as in Figure 4.6, we estimate the value of $\boldsymbol{S}$ using the empirical formula (4.16), see Figure 4.7. We can see that the estimate is very close to the optimal value, at least in terms of the resulting filter performance. Table 4.1 summarises the results numerically.

As we concluded earlier, it is mostly important to tune the value of $\boldsymbol{S}$ (or $\boldsymbol{Q}$) in the correct order of magnitude, as the interval with the 20% best performing values spans roughly several (2-3) orders of magnitude. Note, that the KF performance with $\hat{S}$ still outperforms the KF with $\boldsymbol{Q}_{optimal}$.

Parameter	**Value** (-)	**KF performance** M (-)
$\boldsymbol{Q}_{optimal}$	$1.57 \cdot 10^{-3}$	$3.73 \cdot 10^{-3}$
$\boldsymbol{S}_{optimal}$	$1.80 \cdot 10^{-2}$	$2.57 \cdot 10^{-3}$
$\hat{\boldsymbol{S}}$	$1.60 \cdot 10^{-2}$	$2.75 \cdot 10^{-3}$

Table 4.1: Comparison of the KF performance with the optimal $\boldsymbol{Q}$ or $\boldsymbol{S}$ values with $\hat{\boldsymbol{S}}$ estimated using the empirical formula (4.16).

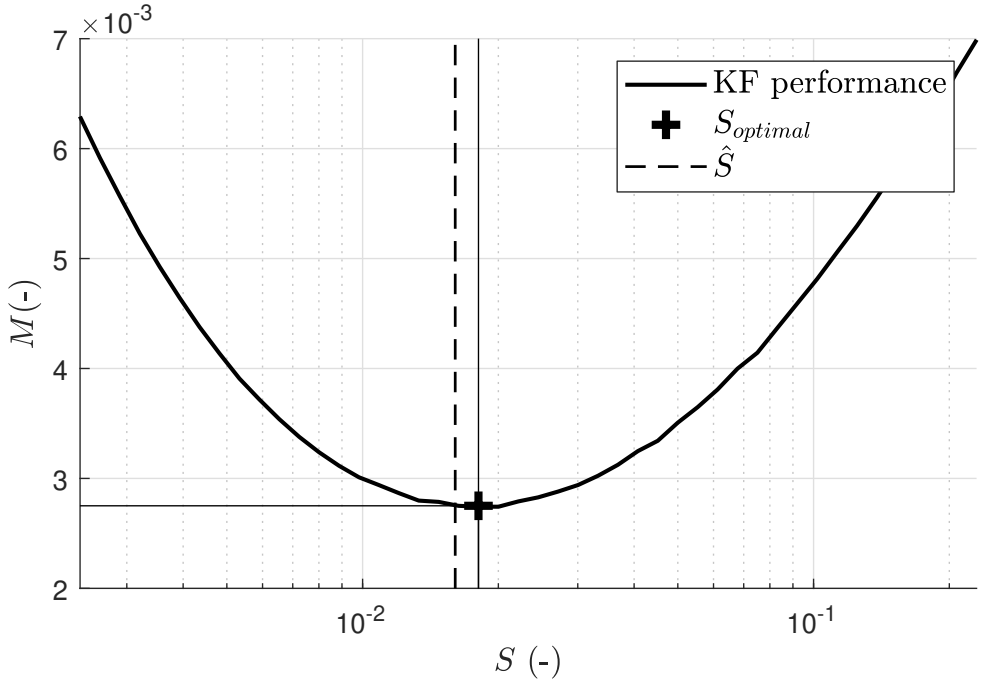

Figure 4.7: Using the same data as in Figure 4.6, the figure shows the estimate of $\boldsymbol{S}_{optimal}$ made using equation (4.16).

The previous experiments only test the numerical formula (4.16) in a single situation with set parameters. To be sure of its robustness and that there is no other quantity that should be included, we performed experiments with varying parameters, again comparing the performance of the modified KF with estimated values of $\boldsymbol{S}$ with the optimal settings, found using gradient optimisation.

Before describing the experiments, we need to add one more assumption, which limits the use of this modified KF (or, in fact, any KF version) to real situations. This assumption comes from the ratio between the noise and any given signal that we deal with, often called the signal-to-noise ratio, or the SNR metric. There are

various ways to calculate it, but we will define it here as (4.17). It is reasonable to say that we are only interested in situations where roughly $SNR \in \langle 1; 100 \rangle$. In situations where $SNR << 1$, the noise is effectively more significant than the signal we are trying to track, rendering the estimation task impossible. On the other hand, in situations where $SNR >> 100$, the noise is practically negligible, making the use of any filter obsolete. This applies to both the measurement and the process noise.

$$SNR = \frac{E\left[signal^2\right]}{E\left[noise^2\right]} \tag{4.17}$$

The first experiment, shown in Figure 4.8, studies the comparison of the optimal KF setting $\boldsymbol{S}_{optimal}$ and the estimated setting $\hat{S}$ acquired from (4.16) with respect to the varying parameter noise (the term s in (4.11)) and the amplitude A of the system input u . Otherwise, all the parameters remained similar to the previous simulations.

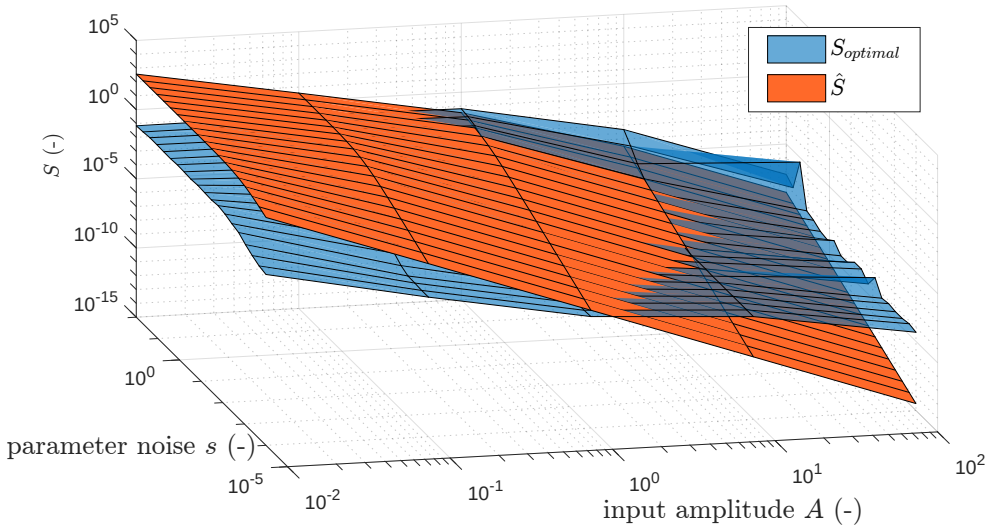

Figure 4.8: A simulation comparing the estimate and optimal value of $\boldsymbol{S}$ for the modified Kalman filter with varying parameter noise $\boldsymbol{S}$ and system input amplitude A. Each point on the surface represents the average value of 50 independent simulations. The optimal value $\boldsymbol{S}_{optimal}$ was found using a gradient search optimisation method and $\boldsymbol{R}$ was set optimally in each simulation.

The resulting shapes show that the formula (4.16) is a good estimate of $\boldsymbol{S}_{optimal}$ with varying parameters, except for situations that violate the reasonable SNR assumption, where the estimate undershoots for very large amplitudes and overshoots for very small amplitudes relative to the parameter noise.

The second experiment was carried out to rule out the potential influence of the measurement noise represented by the parameter $\boldsymbol{R}$ and its potential dependence on the parameter noise s. Figure 4.9 shows that the results are not dependent on the parameter noise, if $\boldsymbol{R}$ is set properly.

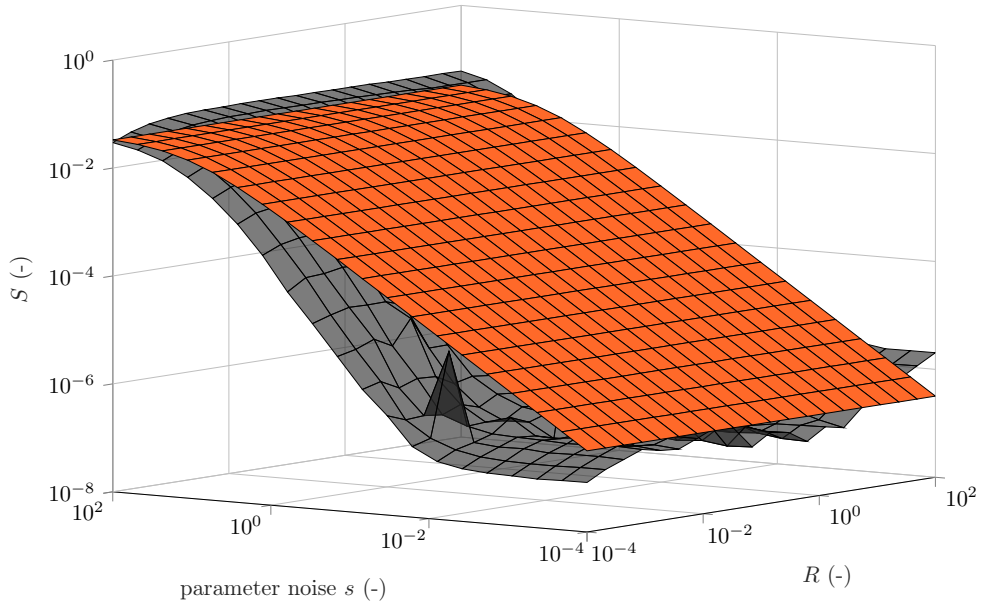

Figure 4.9: A simulation comparing the estimate and the optimal value of $\boldsymbol{S}$ for the modified Kalman filter with varying parameter noise s and the measurement noise $\boldsymbol{R}$. Each point on the surface represents the average value of 50 independent simulations. The optimal value $\boldsymbol{S}_{optimal}$ was found using a gradient search optimisation method and $\boldsymbol{R}$ was set optimally in each simulation.

The third experiment, shown in Figure 4.10 expands on the previous one by studying only the effect of the parameter noise s in more detail, providing not only the average values but also the 95% confidence intervals for both the optimal and the estimated $\boldsymbol{S}$ values.

This experiment demonstrates that the $\boldsymbol{S}$ value estimate slightly overshoots the true optimal values in the interval with a reasonable SNR. The overshoot, at 0-2 orders of magnitude should still provide good enough performance only slightly under-performing the optimal values while keeping on the more stable side, providing more confidence in the measurement relative to the process prediction.

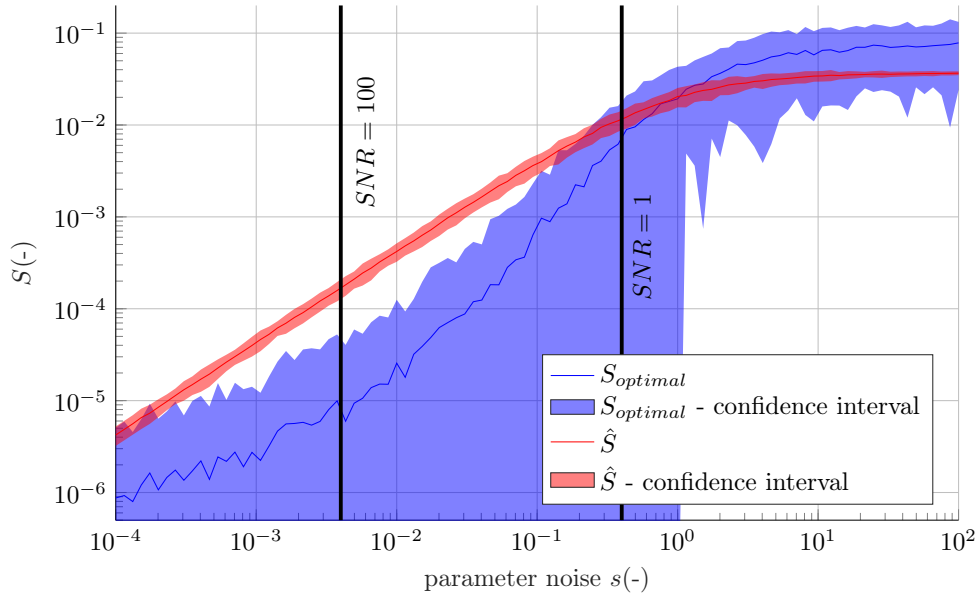

Figure 4.10: A simulation comparing the estimate and the optimal value of $\boldsymbol{S}$ for the modified Kalman filter with varying parameter noise s. Each point represents the average value of 100 independent simulations and its 95% confidence interval. The optimal value $\boldsymbol{S}_{optimal}$ was found using a gradient search optimisation method and $\boldsymbol{R}$ was set optimally in each simulation.

4.5 Dual estimation on a local linear model

The previous experiments were performed with a simple, one-dimensional dynamic system. In this section, we will first extend the modified KF presented earlier on a more general case and then interlink it with a parameter estimation model on a single local linear model of a nonlinear system to provide the basis for further development.

We start with a continuous, nonlinear, non-autonomous, first-order dynamic system described by an ODE (4.18).

$$a_0\dot{x} + a_1 x + a_2 x^3 = u \tag{4.18}$$

where a_0 through a_2 are the system parameters, x is the system state, $\dot{x}$ the state time domain derivative and u the system input. To approximate the system with a local linear model, a single model would take the form of (4.19).

$$\dot{x} = b_0 + b_1(x - c) + b_2 u \tag{4.19}$$

where a_0 trough a_2 are the local model parameters and c the position of the local model centre in the state space of x.

Generally, we require that the system be written in the form consistent with (2.31). To achieve this, we use the Euler discretisation method, setting $x_{k+1} \approx$

$x_k + \dot{x}_k T_s$ with T_s the sampling time in (4.19) acquiring (4.20).

$$\begin{aligned}
x_{k+1} - c &= x_k - c + \dot{x} T_s \\
x_{k+1} - c &= x_k - c + (b_0 + b_1(x_k - c) + b_2 u_k)\, T_s \\
x_{k+1} - c &= x_k - c + T_s b_0 + T_s b_1 (x_k - c) + T_s b_2 u_k \\
x_{k+1} - c &= (1 + T_s b_1)(x_k - c) + T_s(b_0 + b_2 u_k) \\
\downarrow \quad & \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
x^*_{k+1} = \quad & \boldsymbol{F} \quad x^*_k \quad + \quad \boldsymbol{G} \quad u^*_k
\end{aligned} \tag{4.20}$$

With the simple substitutions $\boldsymbol{F} = 1 + T_s b_1$, $x^* = x - c$, $\boldsymbol{G} = T_s$ and $u^* = b_0 + b_2 u$, the Kalman filter framework can be applied to track the state of the system as usual. The same approach can be applied to systems with a higher number of dimensions.

Using the same processes as in the previous sections, we can propagate the parameter inaccuracies $\beta \sim \mathcal{N}(\boldsymbol{B}, 0)$ through the system as (4.21).

$$\begin{aligned}
x^*_{k+1} &= (1 + T_s(\hat{b}_1 + \beta_1))(x_k - c) + T_s(\hat{b}_0 + \beta_0 + (\hat{b}_2 + \beta_2) u_k) \\
x^*_{k+1} &= \hat{\boldsymbol{F}} x^*_k + \hat{\boldsymbol{G}} u^*_k + Ts\beta_0 + T_s\beta_1 x^*_k + T_s\beta_2 u_k \\
x^*_{k+1} &= \hat{\boldsymbol{F}} x^*_{\boldsymbol{k}} + \hat{\boldsymbol{G}} u^*_k + w_k
\end{aligned} \tag{4.21}$$

where the term $w_k = Ts\beta_0 + T_s\beta_1 x^*_k + T_s\beta_2 u_k$ represents the process noise generated by the parameter inaccuracies in the same way as in (4.11). Then, again using the same thought processes as in the previous sections, the process noise covariance (or variance in this case) can be calculated as (4.22), while assuming β_0 through β_2 are independent. Assuming independence in this case is no doubt a simplification, as with most parameter estimation methods, the parameter inaccuracies will be correlated, however, in most practical cases the covariance will be negligible compared to the own parameter variances.

$$\begin{aligned}
var(w_k) &= var(Ts\beta_0 + T_s\beta_1 x^*_k + T_s\beta_2 u_k) \\
var(w_k) &= var(Ts\beta_0) + var(T_s\beta_1 x^*_k) + var(T_s\beta_2 u_k) \\
var(w_k) &= T_s^2 var(\beta_0) + T_s^2 x^{*2}_k var(\beta_1) + T_s^2 u_k^2 var(\beta_2)
\end{aligned} \tag{4.22}$$

Finally, using the empirical formula (4.16) in a piecewise manner, we can calculate the process noise covariance estimate $\hat{\boldsymbol{S}}$ for each of the terms in (4.22) separately

and predict the state covariance according to (4.23).

$$\boldsymbol{P}_{k|k-1} = \hat{\boldsymbol{F}}\boldsymbol{P}_{k-1|k-1}\hat{\boldsymbol{F}}^T + \hat{\boldsymbol{S}}_0 + \hat{\boldsymbol{S}}_1 x_k^{*2} + \hat{\boldsymbol{S}}_2 u_k^2 \tag{4.23}$$

where $\hat{\boldsymbol{S}}_0$ through $\hat{\boldsymbol{S}}_2$ are the respective covariance estimates of the propagated parameter inaccuracies. Note that the term corresponding to $\hat{\boldsymbol{S}}_0$ does not contain any state or input and remains constant, only changing when a new parameter estimate is available. $\hat{\boldsymbol{S}}_0$ through $\hat{\boldsymbol{S}}_2$ are, with respect to the empirical formula (4.16) calculated according to (4.24).

$$\begin{aligned}
\hat{\boldsymbol{S}}_0 &= \frac{1}{2} T_s^2 B_0 \\
\hat{\boldsymbol{S}}_1 &= \frac{1}{2} T_s^2 B_1 E[x^{*2}] \\
\hat{\boldsymbol{S}}_2 &= \frac{1}{2} T_s^2 B_2 E[u^2]
\end{aligned} \tag{4.24}$$

To test these conclusions, we performed a simulation experiment comparing the state tracking and parameter estimation accuracy using the above-described approach for Kalman filtering and a common Recursive least squares parameter estimation as a benchmark. Therefore, the RLS method was applied on the raw data and also as part of the hybrid state / parameter estimation approach with the modified Kalman filter as is depicted in Figure 4.11.

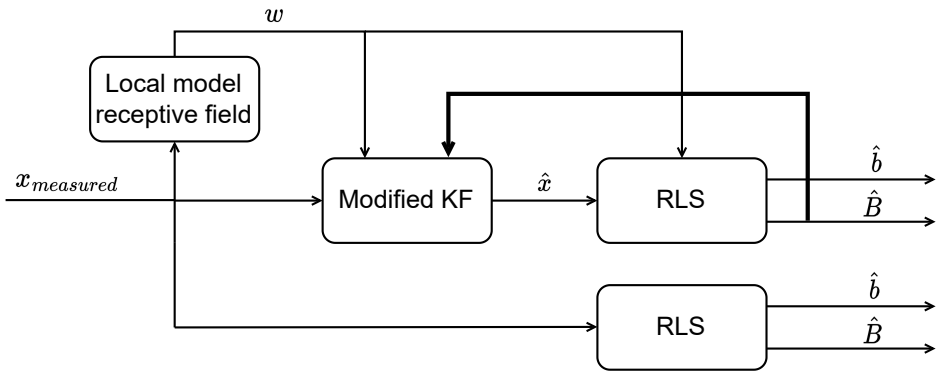

Figure 4.11: The diagram depicts the experiment comparing the hybrid state/parameter estimation using the modified KF approach together with RLS for the parameter estimation and the RLS method on its own. x represents the state of the system, w represents the actual local model weight, b represents the local model parameters and B represents the local model parameter bias covariance.

The experiment simulates the behaviour of a nonlinear dynamic system in the

form of (4.18) with a randomised input signal, the same as in the previous sections. We apply the hybrid estimation to a local linear model in the form of (4.19). The validity interval in the state space of the dynamic system of the local linear model is described by a Gaussian weight function (2.49), the same as with any single local model used in the RFWR algorithm described in Section 2.5.1. In this case, $\boldsymbol{D} = 1$ and $c = 5$.

The weight w in any given state x is used to discount our belief in the overall model prediction. Specifically, (4.23) becomes (4.25). The actual model weight is also used in the weighted version of the RLS algorithm, which is also further described in Section 2.5.1, specifically (2.51) and (2.52).

$$\boldsymbol{P}_{k|k-1} = \hat{\boldsymbol{F}}\boldsymbol{P}_{k-1|k-1}\hat{\boldsymbol{F}}^T + \frac{1}{w}\left(\hat{\boldsymbol{S}}_0 + \hat{\boldsymbol{S}}_1 x_k^{*2} + \hat{\boldsymbol{S}}_2 u_k^2\right) \tag{4.25}$$

To compare the parameters estimated using both approaches, we use a metric described by (4.26), which takes the squared relative error of each of the parameters of the local model into account. In this configuration, the best possible estimate would have the metric value $M_p = 0$. We acquired the ideal parameters by means of analytical linearisation.

$$M_p = \sum_{i=0}^{m}\left(1 - \frac{\hat{b}_i}{b_i}\right)^2 \tag{4.26}$$

Figure 4.12 depicts the tracking of the state x of the system by the modified KF with hybrid parameter estimation. We can see that outside the validity region the KF does not trust the model prediction at all (this corresponds to the $\boldsymbol{R} << \boldsymbol{Q}$ situation with the ideal KF described in Section 4.1).

However, inside the validity region, the filter closely follows the true signal value with a minimal tracking error. This behaviour, when the filter prefers and follows the measurement outside the validity region of the local model is expected and beneficial as it ensures the stability of the tracking task in regions with low model validity or high parameter inaccuracy.

In Figure 4.13, we can see the comparison of the local model parameters, overlaid on the $(\dot{x}, x)$ space, acquired through both approaches. Clearly, the hybrid approach achieves the better results, almost completely eliminating the estimation bias even though the SNR is significant.

Furthermore, to compare the results as they evolved over time, Figure 4.14 shows the value of the M_p metric calculated using (4.26) every time a new datapoint was measured for both methods. Initially, the metric values were high for both the approaches, but as the parameter estimate precision was growing, the hybrid KF approach started taking the model prediction into account, arriving at a much superior result in the end.

The last figure concerning this experiment is Figure 4.15, which shows the signal of the cumulative process noise variance at any given time (without the weight

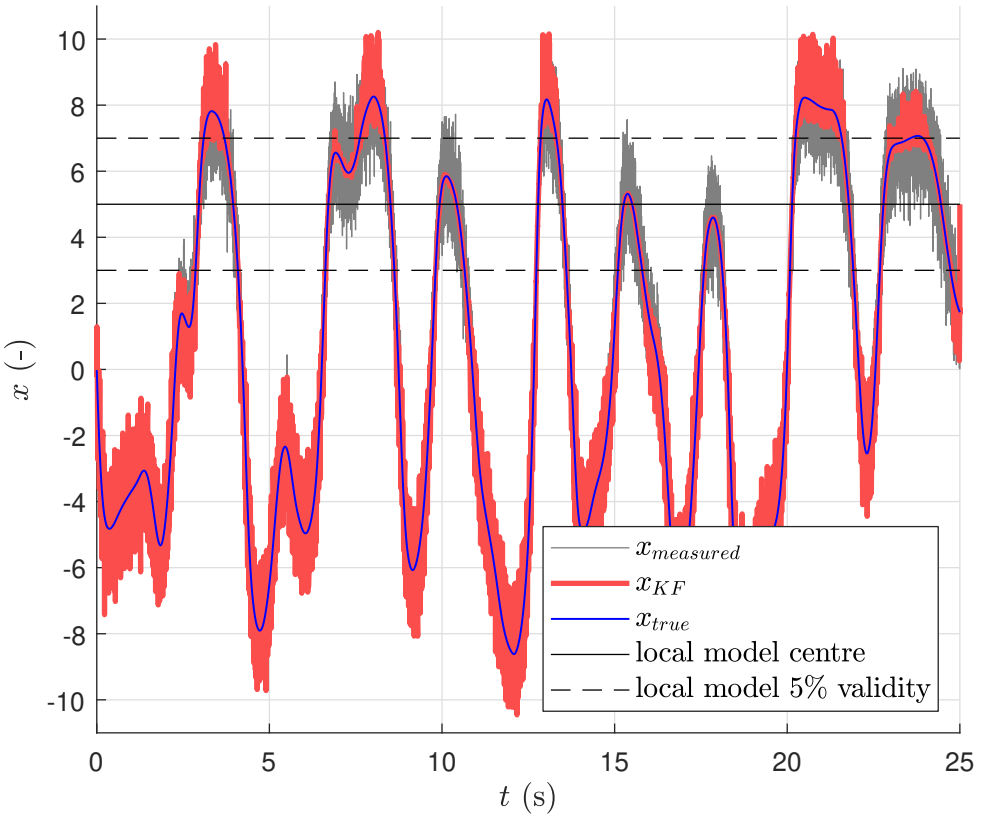

Figure 4.12: State estimation of a nonlinear system using a hybrid KF with online parameter estimation using RLS. We only estimate the parameters of a single local linear model described by $\boldsymbol{D} = 1$ and $c = 5$. Outside the validity region of the local model the KF does not take the process prediction into account and strongly prefers the measurement signal, ensuring stability.

discounting). The plot demonstrates that the model inaccuracy estimate drops over time. Also, the periodical rises in the variance value correspond with the actual state being outside of the local model validity range.

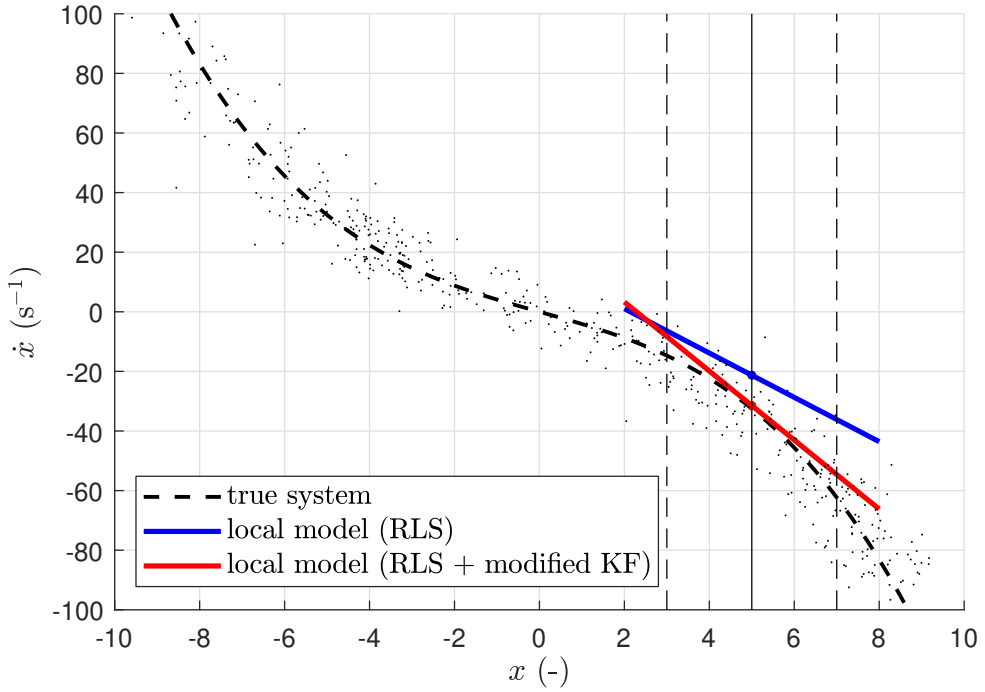

Figure 4.13: Local model parameters acquired using the two different approaches. The true nonlinear system dynamics is represented by the dashed line, the actual noisy datapoints are also depicted here.

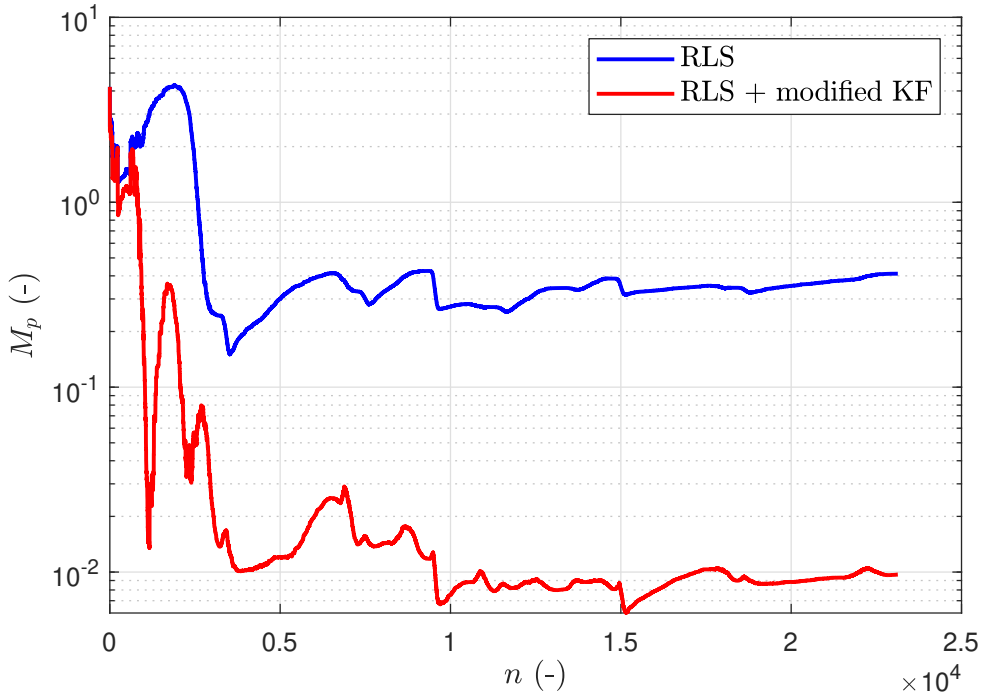

Figure 4.14: Comparison of the online parameter estimation process using the RLS alone and the hybrid KF + RLS state and parameter estimation. Only datapoints with $w > 0.001$ are plotted here as the estimation process essentially stops for $w < 0.001$, meaning the datapoint is outside of the local model validity range.

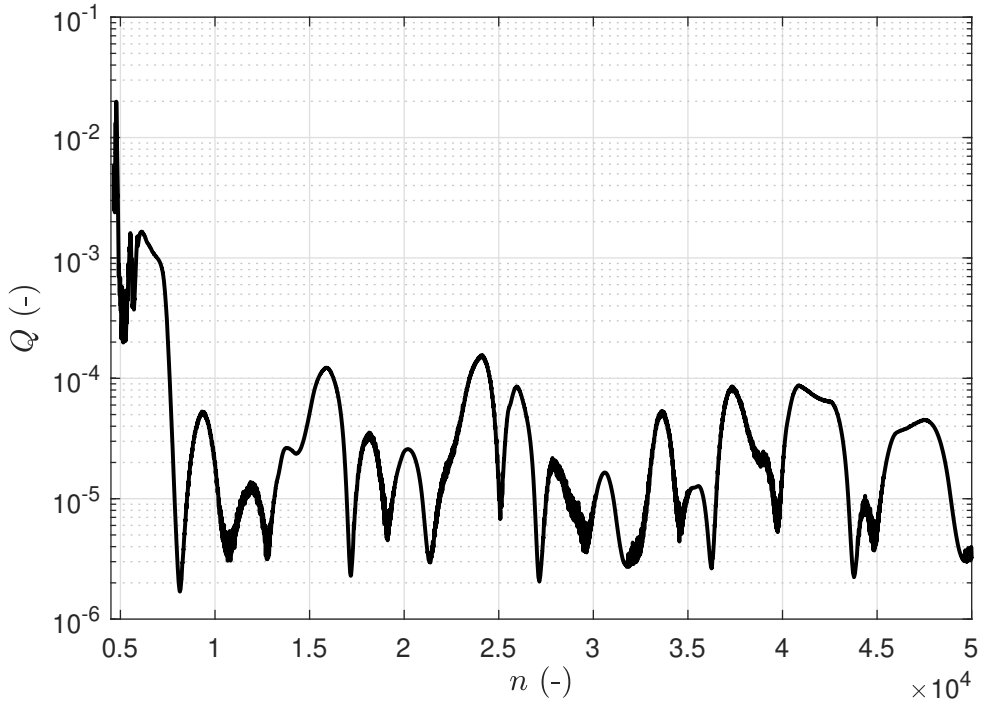

Figure 4.15: The plot shows the cumulative process noise from all the sources, as described in Equation (4.25) evolving over time. The decreasing trend represents the overall improvement in the model parameters while the occasional rises represent the increased model variance due to the actual state being outside the validity region of the local model.

4.6 Dual estimation in multiple dimensions

The last experiment in this section extends the previous one by working with a dynamic system of a higher order, specifically described by (4.27).

$$a_1\ddot{x} + a_2\dot{x} + a_3\dot{x}^3 + a_4x + a_5x^3 + a_6x^5 = u \tag{4.27}$$

with $a_1 = 0.03$, $a_2 = 0.04$, $a_3 = 0.0008$, $a_4 = 20$, $a_5 = -50$, $a_5 = 40$. These specific parameters generate a system with non-monotonic nonlinearity with respect to the state x. In this case, we work with a local linear model in the form (4.28).

$$\ddot{x} = b_0 + b_1(\dot{x} - c_1) + b_2(x - c_2) + b_3u \tag{4.28}$$

With a system of a higher order, there is one more step to consider before arriving at a general formulation of the modified KF framework suitable for systems with dominant parameter uncertainty, e.i., the multidimensional state space. To develop a proper formulation, we start with an n-dimensional autonomous discrete system in the form (4.29)

$$\boldsymbol{x}_{k+1} = F\boldsymbol{x}_k \tag{4.29}$$

where $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ are n-by-1 vectors and $\boldsymbol{F}$ is an n-by-n square matrix. Assuming that the system is written in canonical form, $\boldsymbol{F}$ has the shape of (4.30), meaning that the state vector $\boldsymbol{x}$ is a sequence of derivatives.

$$\boldsymbol{F} = \begin{bmatrix} 1 & T_s & 0 & 0 & \dots & 0 \\ 0 & 1 & T_s & 0 & \dots & 0 \\ \vdots & & \ddots & & & \vdots \\ 0 & \dots & 0 & 0 & 1 & T_s \\ f_1 & f_2 & & \dots & & f_n \end{bmatrix} \tag{4.30}$$

Using the same thought process as with the one-dimensional case (4.11), we can now expand the prediction step of the KF algorithm in the same way as for a higher-dimensional case in (4.31).

$$\begin{aligned} \boldsymbol{x}_{k+1} &= (\hat{\boldsymbol{F}} + s)\boldsymbol{x}_k \\ &= \hat{\boldsymbol{F}}\boldsymbol{x}_k + s\boldsymbol{x}_k \\ &= \hat{\boldsymbol{F}}\boldsymbol{x}_k + \boldsymbol{w}_k \end{aligned} \tag{4.31}$$

where $\hat{\boldsymbol{F}}$ is the state matrix estimate, $\boldsymbol{w}_k = s\boldsymbol{x}_k$ is an n-by-1 vector of process noises

and s is an n-by-n parameter noise matrix.

Since $\hat{\boldsymbol{F}}$ is constructed through the discretisation of the continuous form of the system, and assuming the discretisation error is negligible, the parameter estimation noise takes the form of (4.32).

$$s = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \\ s_1 & s_2 \quad \dots & s_n \end{bmatrix} \tag{4.32}$$

Then, the covariance matrix $\boldsymbol{Q}$ can be derived from (4.33).

$$\begin{aligned} \boldsymbol{Q}_k &= cov(\boldsymbol{w}_k) \\ &= cov(s\boldsymbol{x}_k) \\ &= \begin{bmatrix} 0 & \dots & & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & \boldsymbol{x}_k^T \boldsymbol{S} \boldsymbol{x}_k \end{bmatrix} \end{aligned} \tag{4.33}$$

where $\boldsymbol{S} = diag([\boldsymbol{S}_1, \boldsymbol{S}_2, \dots, \boldsymbol{S}_n])$, with $\boldsymbol{S}_1$ through $\boldsymbol{S}_n$ are the variance estimates of the corresponding noise terms s_1 through s_n calculated using the empirical formula (4.16), respectively.

We can see that the only non-zero element in the $\boldsymbol{Q}_k$ matrix used in the prediction step of the KF corresponds to the specific case with one input and one state described in (4.25).

In this manner, we can calculate the actual process noise covariance matrix with the parameter noise that only influences the highest-order derivative state in the state vector. This can also be seen intuitively, as the other states are calculated using integration, which does not bring any parameter noise to the system.

However, it is advisable not to leave the other diagonal elements as exactly zero, but to use a very small number, with respect to the corresponding elements of $\boldsymbol{R}$, to ensure the higher stability and prevent the long-term drift of the higher-order states even if they are being measured. Also, these elements can be nonzero if the other process noise causing effects categorised in Section 4.2 are not negligible, especially the effect of discretisation.

Applying these findings to the multidimensional case used in our experiment, the overall process noise covariance used in the prediction step at each step is calculated through (4.34).

$$\boldsymbol{Q}_k = \begin{bmatrix} q_{11} & 0 \\ 0 & \hat{S}_0 + \hat{S}_1 x_{1k}^{*2} + \hat{S}_2 x_{2k}^{*2} + \hat{S}_3 u_k^2 \end{bmatrix} \tag{4.34}$$

where $\hat{\boldsymbol{S}}_0$ through $\hat{\boldsymbol{S}}_3$ correspond to the optimal variance estimates calculated using the empirical formula (4.16) for the corresponding parameters b_0 through b_3 in the local linear model in the form (4.28).

Using (4.34), the prediction step is performed through (4.35).

$$\boldsymbol{P}_{k|k-1} = \hat{\boldsymbol{F}}\boldsymbol{P}_{k-1|k-1}\hat{\boldsymbol{F}}^T + \frac{1}{w}\boldsymbol{Q}_k \tag{4.35}$$

As we commented earlier, the term $\boldsymbol{Q}_{11}$ is not set as zero. In this experiment, it is set as $\boldsymbol{Q}_{11} = 10^{-4}$ to account for the discretisation error, while the measurement noise covariance is set to $\boldsymbol{R} = \begin{bmatrix} 10^{-3} & 0 \\ 0 & 10^{-1} \end{bmatrix}$ corresponding to the artificially generated measurement noise during the simulation experiment.

Throughout the experiment, the local model centre was placed at $c_1 = 0.85$ and $c_2 = 5$, with the corresponding Gaussian validity function described by the size inducing matrix $\boldsymbol{D} = \begin{bmatrix} 100 & 0.4444 \\ 0 & 10^{-1} \end{bmatrix}$.

Figures 4.16 and 4.17 demonstrate the results of the experiment, similar to the one-dimensional case. We can see that the hybrid approach was able to stably achieve better results, while requiring no parameters to be set manually.

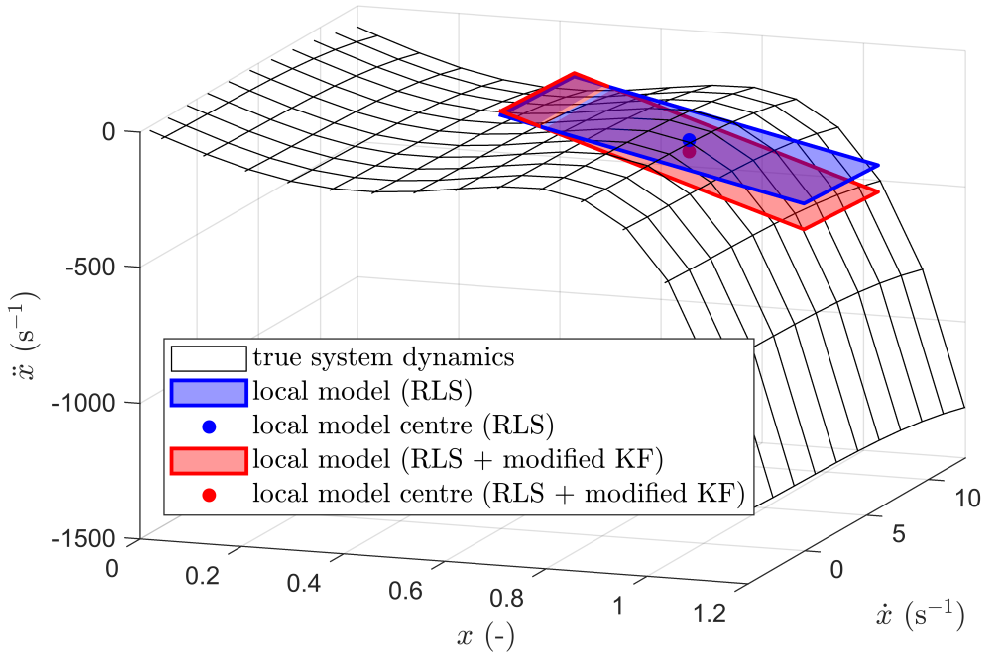

Figure 4.16: Local model parameters acquired using the two different approaches. The true nonlinear system dynamics is represented by the transparent mesh.

The results show that the simultaneous state and parameter estimation approach using a hybrid KF with the modified process covariance prediction step and the RLS algorithm is viable and can provide significant benefits over simply applying the RLS

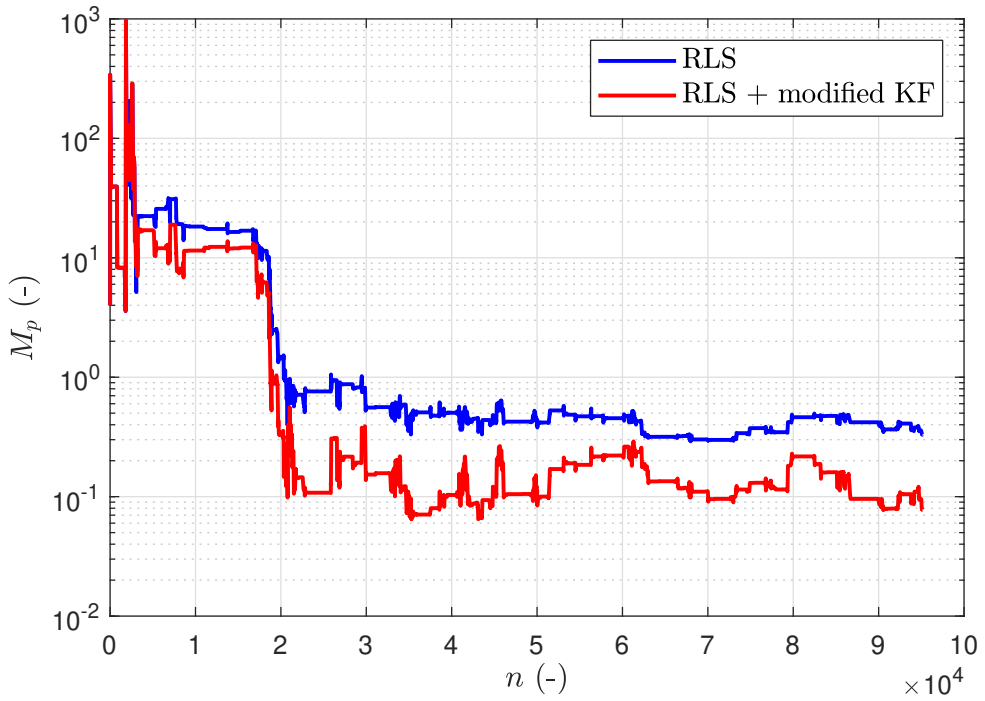

Figure 4.17: Comparison of the online parameter estimation process using the RLS alone and the hybrid KF + RLS state and parameter estimation.

method on the raw data and over the other dual estimation methods by not requiring manual parameter tuning.

4.7 Chapter summary

In this chapter, we first studied in Section 4.1 the Kalman filter under ideal conditions with all theoretical assumptions met to demonstrate the performance of the algorithms with correct and incorrect settings and to come to an important conclusion that the interval with optimal setting usually spans over one or two orders of magnitude. Therefore it is not the exact value of the parameter that needs to be set, but only its correct order to achieve close-to-optimal behaviour.

Then, in Section 4.2, we studied the various sources of error which cause and influence the process noise and framed the exact situation we are dealing with when the dominant source is the parameter estimation error or uncertainty.

Further on, in Section 4.3, we reformulated the process covariance prediction step of the Kalman filter to better suit our case and showed in simulation experiments that it performs marginally better that the original algorithm.

Building upon previous findings, in Section 4.4, we describe an empirical formula to estimate the process noise variance in this specific scenario and show in statistical simulation experiments that it is able to provide a reasonably accurate and robust estimate.

Then in the last two Sections 4.5 and 4.6 we form the hybrid dual estimation

algorithm for a single local model and then expand the idea to a higher-dimension space, respectively. In both Sections the simulation experiments show that this parameterless hybrid dual estimation performs better than a simple recursive least squares method while not using any tunable parameter.

5 RFWR modification

In the previous chapter, we developed and experimentally verified a modified version of the Kalman filter suitable for hybrid dual estimation with local linear models. Before applying this modification at scale, in this chapter, we return to the original local modelling algorithm that we chose as our starting point, the Receptive Field Weighted Regression, to further develop and modify this method to be more suitable for use in the situations that we are dealing with in this thesis. The main topics of the research are summarised in Chapter 3, Section 3.

We implemented all the modifications to the RFWR algorithm as a part of a new function library developed for Matlab, which served to simplify the experiments and also represents a notable output of this research on its own, as it can be used in the future by other researchers and engineers.

The library is represented by a Matlab class called *rfdelib* that implements all the core functions and presents the user with public access methods for use and plotting. The name of the library (RFDE) stands as an acronym for Receptive Field Dual Estimation, which comes from the idea of merging the RFWR algorithm with the Kalman filter for dual estimation, presented in Chapter 6. In this chapter, we only describe a part of the library connected to the RFWR algorithm itself, as it can be used as an approximation tool on its own. These two Chapters 5 and 6 describe both the algorithms and their implementations in the library. The reason behind this kind of presentation is that it may also serve as a quick guide for a potential library user.

This chapter is divided into two sections. First, in Section 5.1, we present the modified RFWR algorithm implemented in the library, and then, in Section 5.2, we present experimental simulations that demonstrate the library functions.[1]

5.1 RFWR library for MATLAB

The main functionality of the library is accessible to the user through just three methods, presented with their inputs and outputs in Figure 5.1. Throughout the library, $\boldsymbol{x}$ and y are used to mark the input and the output of the set of local models that approximate a relation $y = f(\boldsymbol{x})$. The library is implemented assuming that y is a scalar, since there is no reason for multiple outputs to be simultaneously modelled by the same set of local models. In a case where more outputs are needed, the library can be initialised in several independent instances existing in parallel. The first method, the constructor bearing the name of the library, is used to initialise the

[1] In this chapter, we assume the reader is familiar with the basic concepts of (object-oriented) programming and Matlab programming language syntax.

object with user-defined parameters, while only the *dataInputSize*, which specifies the number of inputs in $\boldsymbol{x}$, is compulsory.

```
function h          = rfdelib      (dataInputSize, options)
function              learn        (x,y)
function [y,ConfI] = get_estimate  (x)
```

Figure 5.1: Main publicly accessible methods of the *rfdelib* Matlab class.

Next, there is the *learn* function, which takes both $\boldsymbol{x}$ and y as input, which have the meaning of a single corresponding data sample ($\boldsymbol{x}$, y) used to update the set of local models stored inside the object. The method has no output. This method implements the core functionality of the RFWR algorithm as described in [29], namely adding new and pruning obsolete local models, and updating the model parameters and their respective receptive fields. All the functions called inside the *learn* method are privately accessed methods of the *rfdelib* class. Figure 5.2 shows a simplified implementation of the learn method.

```
function    learn   (h,x,y)
    W = h.get_lm_weight(x,0,0); % calculate weight for every lm

    h.update_lm(x,y,W); % update lms using RLS and distance metric

    h.add_lm(x,y,W); % add new lm & rf if necessary

    h.prune_lm(W)% prune obsolete lm & rf if necessary
end
```

Figure 5.2: Implementation of the basic RFWR functionality in the *learn* method.

Lastly, there is the method *get_estimate*, which is used to calculate the actual y estimate corresponding to the input query point $\boldsymbol{x}$. The method also returns the size of the 95% confidence interval corresponding to the estimate. The confidence interval can be calculated using the LS parameter bias based on (4.15) or, since the parameters are being estimated using the RLS algorithm, we can utilise the covariance estimate $\boldsymbol{P}$ the algorithm provides and iteratively updates anyway. The covariance matrix $\boldsymbol{P}$ in (2.51) represents the variances of and covariances between the individual parameters with $var\left(\hat{b}_i\right) = \boldsymbol{P}_{ii}$ the variance of the i-the parameter itself, and $cov\left(\hat{b}_i, \hat{b}_j\right) = \boldsymbol{P}_{ij} = \boldsymbol{P}_{ji}$ the covariance of the two parameter estimates. The estimated variance of a single model output can then be calculated by propagating $\boldsymbol{P}$ to the output space, and we can calculate the standard error se_y by taking the square root of the variance as in (5.1).

$$se_y = \sqrt{var(\hat{y})} = \sqrt{\boldsymbol{x}^T \boldsymbol{P} \boldsymbol{x}} \tag{5.1}$$

Naturally, se_y corresponds to the 68% confidence interval and $2se_y$ to the 95% confidence interval of $\hat{y}$. The *get_estimate* method returns the weighted average of se_y according to the validity value of the local models for the given $\boldsymbol{x}$.

We experimented with various implementations of (5.1) to improve the performance, for example, by only working with diagonal elements of the matrices, but in the end, the original method proved to be the most robust, and the lower computational complexity of any simplification did not outweigh the loss in precision and numerical stability.

It is actually quite simple to use the library as is shown in the pseudocode presented in Figure 5.3.

```
LM = rfdelib(2); % initialise the LM object for two inputs

for i = % each datapoint index
    x(:,i) = get_system_input(); % measure or load input data
    y(i) = get_system_output(); % measure or load output data
    LM.learn(x(:,i),y(i))
end

xq = [0 0]; % the query point - the point of interest
yq = LM.get_estimate(xq); % calculate output estimate
```

Figure 5.3: Simple example of the library usage.

For user convenience, there are also publicly accessible methods which generate data for various possible visualisations describing the set of local models and the current approximation result, especially in lower-dimensional cases. These are summarised in Figure 5.4. If these methods do not suffice, there are also some 65 publicly accessible properties that contain all the data representing the set of local models and a number of more advanced parameters in case a user wants to tune them. For a detailed list of the class properties and method description, see the library code.

```
function [rfDataX,rfDataW]   = get_rf_data_1D                (points)
function [C,lmDataX,lmDataY] = get_lm_data_1D          (points,scale)
function data                = get_rf_data_2D          (points,scale)
function params              = get_local_params                  (xq)
function hndls               = init_plot_1D
                    (estX1Values, idealEstimateFcn,trState,msrState)
function hndls               = update_plot_1D           (xNew,yNew,i)
function hndls               = init_plot_2D
          (estX1Values,estX2Values,idealEstimateFcn,trState,msrState)
function hndls               = update_plot_2D                    (i)
function capture_plot_frame                                      ()
```

Figure 5.4: Convenience and plotting functions of the *rfdelib* Matlab class.

5.1.1 Local models and receptive fields

Compared to the original algorithm, the RFWR method was modified to be used incrementally in an online setting. The RLS learning algorithm is incremental by default, however, the approach for optimisation of the distribution of the receptive fields was supposed to be used on batch basis. To address this issue, we developed different optimisation strategies, which are further described in Section 5.1.2.

That being said, the original RFWR algorithm can be implemented in an incremental way, however, it turned out to be very hard to set the parameters correctly as the incremental model parameter estimation and receptive field distribution optimisation have low stability, especially with uneven data distribution in the input space. For example, with an incorrect parameter setting and a high data sampling rate, individual models would adjust to data points collected at one location much smaller than the local model validity region defined by the receptive field and would completely neglect past data gathered at different locations, sort of overfitting.

To address this issue, we added a datapoint buffer for every local model which is used to update the model parameter every time a new datapoint is collected, while the datapoints in the buffer are replaced by new ones on a random basis. Figure 5.5 depicts the specific algorithm of the data buffer update when a new datapoint is available.

```
function [dataBuffer,buffState] = buffer_update
                            (dataBuffer,xNew,yNew,w,buffState)
    if w > WAct/3
        if buffState < length(dataBuffer) % add if buff not full
            buffState = buffState + 1;
            pointer = buffState;
        else % change random entry (closest 1/5 of entries) if full
            xDist = sum((dataBuffer(:,1:end-1) - xNew).^2,2);
            [~,ind] = sort(xDist,'ascend');
            pointer = ind(randi([1  round(length(dataBuffer)/5)]));
        end
        dataBuffer(pointer,:) = [xNew yNew]; % write data into buffer
    end
end
```

Figure 5.5: Data buffer update method pseudocode.

Now, the *update_lm* method used inside the *learn* method when a new datapoint is collected follows the algorithm described by the pseudocode in Figure 5.6. This can increase the computational complexity, so the buffer size needs to be reasonable, but experiments proved that even buffer sizes between 15-30 significantly improve the learning stability, without any major performance issues.

Another major modification which has the potential to greatly simplify the use of the library is the introduction of the option to set apart the dimensions of the local models and their respective receptive fields. In effect, this causes the local model to distribute receptive fields along only a subset of the input dimensions. This may be

```
% calculate model weights corresponding to the new datapoint
W = get_weight(x);

for i = % every existing local model
    if W(i) > w_act % if the model is activated by this datapoint

        buffer_update(i,x); %update the buffer of the actual lm

        for j = % every datapoint in the buffer
            w = get_weight(x_buff(j));

            rls_update(); % update model parameter

            rf_update(); % update rf parameters
        end
    end
end
```

Figure 5.6: *update_lm* method pseudocode describing the use of the data buffer when a new datapoint is acquired.

useful in situations where we expect the nonlinearity in the approximated function to be dependent only on some of the input quantities and be linear in others. This greatly improves the performance and stability, especially when there is only one nonlinear dimension. Typically, with mechanical systems, we expect the system to be linear in velocity while being nonlinear in position, also with multi-domain systems, typically only some of them contain nonlinearities.

This ability is implemented in the form of *rfInputToggling* property, which is one of the optional constructor parameters. By default, this is a row vector of ones, with the length of *dataInputSize*. However, if we set the vector manually, for example as in Figure 5.7, the library starts to ignore the first and the third input when placing the local model receptive fields, behaving the same way as if the centre location along the ignored dimension was zero and the size was infinite for all the local models.

```
    % initialise the LM object for three inputs while only the second
    % input is considered in rf distribution
    LM = rfdelib(3,'rfInputToggling',[0 1 0]);
```

Figure 5.7: Initialisation of the library with three expected model inputs, but only one-dimensional receptive field distribution.

This behaviour presents a secondary issue. That is the fact that different local models can have different parameters along the ignored axes due to noise and uneven data distribution, even though we can expect them to end up being equal. We solved this issue by adding further functionality to the library, the so-called global parameter matching. When turned on (the function is turned off by default), this function slowly brings the parameters to a common value (average or user speci-

fied). This has the benefit of slowly sharing information between the models, while not disturbing the stability of the RLS update process. Figure 5.8 shows various examples of how to use this functionality.

```
% global parameter matching for the remaining input dimension
% towards the mean value of all lms
LM = rfdelib(2,'rfInputToggling',[0 1]); % initialise the LM object
LM.globalParamMatchingSourse = 'mean';

% global parameter matching for the remaining input dimension
% towards a specific value
LM = rfdelib(2,'rfInputToggling',[0 1]); % initialise the LM object
LM.globalParamMatchingSourse = 'value';
LM.globalParamMatchingSourse = 'value';
LM.globalParamMatchingValue = 0.01; % target value

% turn global parameter matching off at any time
LM.globalParamMatchingSourse = 'none';
```

Figure 5.8: Usage of the global parameter matching functionality

The third significant add-on is the generalisation of the model input vector. In the original RFWR algorithm, there is the assumption that the local models generate and estimate a parameter for each of the system inputs plus a bias parameter, which is implemented through expanding the input vector by a unit constant. The library treats the data input vector and the model input vector as two separate things, linked by a private method called *lm_input_shuffler*. By default, this method acts the same way as the original algorithm - expands the input vector by the number one to generate the bias parameter, however, it allows for much more customisation. Generally, the local models used inside the RFWR framework do not have to be only linear combinations of input quantities, it can take the form of any function of the inputs that is linear in parameters. For example, using a local model in the form of (5.2) instead of (4.19).

$$\dot{x} = b_0 + b_1(x - c) + b_2(x - c)^2 + b_3 u \tag{5.2}$$

The method is prepared to work with linear, quadratic and cubic terms and special terms used to model mechanical friction (the sign function, etc.) and allows the user to easily implement their own version of the local model form.

5.1.2 Receptive fields update algorithms

As we mentioned earlier, one separate modification to the original RFWR algorithm that we made is the addition of several different methods for optimising the distribution of the receptive fields.

Naturally, the first method that the *rfdelib* class implements is the original al-

gorithm described in [29]. This is the only part of the library taken from another source (the authors' implementation published in [46]) and is only modified to fit the library framework.

The original algorithm, based on estimating the true first- and second-order gradient of the quadratic cost function (2.53) as precisely as possible, is quite computationally heavy, requires a lot of tuning parameters and is better suited for batch (offline) learning applications. As opposed to this approach, we developed and implemented three other methods, each with different properties.

Heuristic update

The first method uses the analytical Jacobian matrix of the weight function (2.49) to be able to correctly adjust the elements of the distance inducing matrix M, however, it uses a simple heuristic decision rule to choose if the region of validity of the given local model should be made larger, smaller, or stay the same with respect to the actual datapoint and the local model's long-term performance. This method was first introduced in [28] and later improved. The heuristic is best described by Figure 5.9.

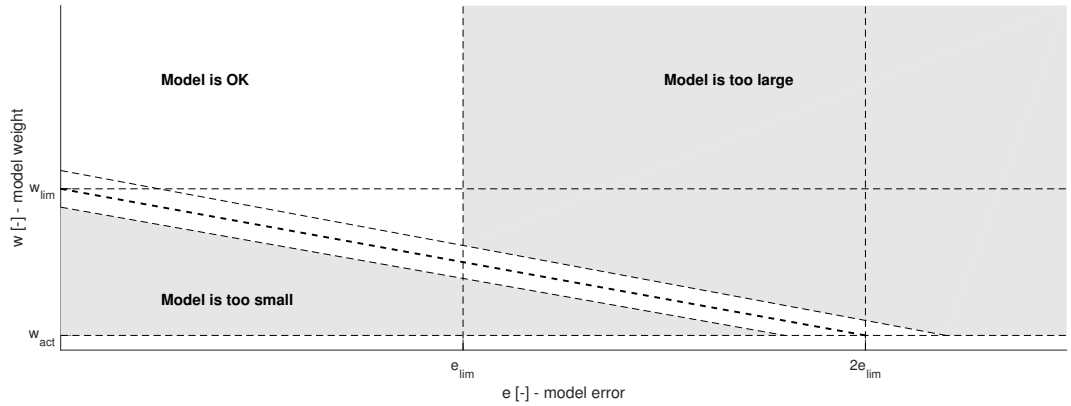

Figure 5.9: Heuristic decision function diagram. Taken from [28].

The heuristic is designed in a way which ensures that the receptive field updating effectively stops when the average model error with new datapoints achieves a reasonable value e_{lim} with a model weight w_{lim}, assuming the error will be smaller for larger weights (closer to the model centre) and larger for smaller weights, to prevent overfitting.

The algorithm is best described by its (redacted) code for each individual update made with a new datapoint depicted in Figure 5.10 for one and two-dimensional cases. The algorithm starts with the model error value $err = abs\,(y - \hat{y})$ with respect to the actual model prediction $\hat{y} = \boldsymbol{x}^T\hat{\boldsymbol{b}}$ and the actual datapoint y and uses the heuristic to decide on the action required. The other inputs to the private method are the direction vector $v_e = \boldsymbol{x} - \boldsymbol{c}$ from the RF centre $\boldsymbol{c}$ to the input datapoint $\boldsymbol{x}$, the actual model weight w and the actual state of the upper triangular decomposition $\boldsymbol{M}$ of the distance inducing matrix $\boldsymbol{D} = \boldsymbol{M}^T\boldsymbol{M}$.

```
function M = distance_update_heuristic (err,v_e,w,M)

    Wl = k_lim*err + w_lim; % Weight limit for current model error

    p = 0; % metrics update direction sign
    if w < 0.9*Wl % model is too small
        p = min([0.9*Wl-w w-WAct]);
    elseif w > 1.1*Wl && e > e_lim % model is too large
        p = -min([e-e_lim w-1.1*Wl w-WAct]);
    end

    dwdM = -w*v_e^2*M; % calculate weight gradient
    M = M + p*alph*dwdM; % perform metrics update
end
```

Figure 5.10: (Simplified) Implementation of the Heuristic Receptive Field update algorithm.

The method also uses the parameters (implemented as object properties) e_{lim} representing the error limit for updating the distance metrics, w_{lim} representing the weight limit for updating the distance metrics and $k_{lim} = -\frac{w_{lim}-w_{Act}}{2e_{lim}}$ representing the slope of the learning threshold line. The values $w_{lim} = 0.6$, e_{lim} depend on the specific case and need to be set appropriately by the library user, for normalised inputs, 0.1 is reasonable.

The method also uses the weight gradient, which is a simplification of the proper gradient of the criterion function as in (2.54), where $\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{M}} \approx \frac{\partial w}{\partial \boldsymbol{M}}$. This simplification can be made thanks to the fact that the criterion function $J(w)$ is a function of the weight w as well as other terms, which can be neglected at the cost of the loss of some precision, however, in gradient optimisation, we mainly care about the gradient direction, the size is not as important and can be compensated by the optimisation step size parameter α. See [29] and [33] for further details on this simplification. An important benefit of this simplification is that the gradient $\frac{\partial w}{\partial \boldsymbol{M}}$ can be calculated analytically as in (5.3) for a one-dimensional case where all the quantities become scalars, although it can be calculated for a higher-dimensional case as well. The *rfdelib* library implements one- and two-dimensional cases.

$$\begin{aligned}\frac{\partial w}{\partial \boldsymbol{M}} &= \frac{\partial \left(e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c})^T \boldsymbol{M}^T \boldsymbol{M}(\boldsymbol{x}-\boldsymbol{c})}\right)}{\partial \boldsymbol{M}} \\ &= \frac{\partial \left(e^{-\frac{1}{2}(x-c)^2 \boldsymbol{M}^2}\right)}{\partial \boldsymbol{M}} \\ &= \frac{\partial \left(e^{-\frac{1}{2}v_e^2 \boldsymbol{M}^2}\right)}{\partial \boldsymbol{M}} \\ &= -v_e^2 \boldsymbol{M} \left(e^{-\frac{1}{2}v_e^2 \boldsymbol{M}^2}\right) \\ &= -v_e^2 \boldsymbol{M} w\end{aligned} \tag{5.3}$$

Random update

The second update method that the library implements is based on the stochastic optimisation approach. It only works with diagonal distance inducing matrices which, on the one hand, do not allow for such precise receptive field optimisation and usually require a few more local models to cover the same input space, however, on the other hand, it requires much less computational power and brings more stability. In effect, the RF basis functions only scale along the RF input dimensions.

Every time there is a new datapoint, the method picks a random RF input axis and only updates the RF size along that axis. The algorithm chooses the size and direction of the update iteration based on the overall long-term performance of each local model. The local model performance is calculated as the long-term sum of the weighted squared estimation errors (WSE) with a forgetting factor λ, every update of the metric at each time step k is made through (5.4).

$$WSE_{k+1} = \lambda WSE_k + w \left(y_{k+1} - \hat{y}_{k+1}\right)^2 \tag{5.4}$$

Then, Figure 5.11 shows the (simplified) implementation of the update method used in the library.

The random update method performs the update on $\boldsymbol{C} = \boldsymbol{M}^{-1}$ instead of updating M directly, which corresponds to the direct size of the RF validity function since we use a Gaussian kernel. This also allows for the easy limit check of the minimal and maximal sizes directly in the RF input space domain compared to the domain of $\boldsymbol{M}$, which is much more intuitive and easier to set up.

Numerical update

The last update algorithm that the *rfdelib* class implements is a method using the numerical approximation of the criterion function $\boldsymbol{J}$ gradient. In this case, we work directly with the true criterion gradient $\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{M}}$ instead of the simplified $\frac{\partial w}{\partial \boldsymbol{M}}$, optimising the criterion (2.53). Naturally, we cannot calculate the true value of $\boldsymbol{J}$ on all the

```
function M = distance_update_random (M,WSE)
    n = randi([1 length(M)]);
    p = (WSE - WSE_lim)/WSE_lim;

    C = inv(M);
    C(n,n) = C(n,n) - p*alph*randomStepSize; % perform metrics update

    M = inv(Cut);
end
```

Figure 5.11: (Simplified) Implementation of the Random Receptive Field update algorithm.

datapoints since we want the method to work incrementally. However, the library has a data buffer for each of the local models (described in the Section 5.1.1) that brings up the synergy and allows us to calculate at least a partial estimate of $\boldsymbol{J}$ based on the values stored in each of the local buffers every time a new datapoint is acquired and the buffer is updated.

Again, the best way to describe the methods is through its implementation in the library shown in Figure 5.12.

```
function [M] = distance_update_numerical(M,DB,c,b,w,ID)
    x = DB(:,1:end-1); % read x data from local model buffer
    y = DB(:,end); % read x data from local model buffer

    % calculate estimation error for each of the datapoints
    xn = h.lm_input_shuffler(x - c);
    e = y - xn*b';

    m = det(M)/1000; % estimate M differential
    w1 = h.get_lm_weight(DB(:,1:end-1),ID,m);

    J0 = e'*diag(w)*e/sum(w)/M^2;
    J1 = e'*diag(w1)*e/sum(w)/(M+m)^2;

    dJdM = (J1 - J0)/m;

    dM = alph*dJdM*numericalStepSize;

    M = M - dM; % perform metrics update
end
```

Figure 5.12: (Simplified) Implementation of the Numeric Receptive Field update algorithm.

5.1.3 Further modifications

In the previous sections, we described the major modifications and extensions made on the original RFWR algorithm. There are also a number of smaller, yet important, extensions that the *rfdelib* library implements.

First, the library allows for three different variants regarding the setting of the initial size of a newly added RF. The three methods are a fixed pre-set size (original version), a random size, and the maximal size to intersect with the nearest existing RF to cover the vacant space.

Another modification which is minor in its implementation, but has significant consequences, is the use of the weighted version of the RLS parameter estimation algorithm for the local model parameters. This modification allows us to also consider the assumed accuracy of the input datapoint, which will become useful when implementing the hybrid dual estimation in Chapter 6. In addition, every parameter estimation iteration is calculated on a batch of data present in the data buffer of each local model, which significantly improves the stability.

Also, the library implements a number of ease-of-use methods that allow the user to easily get, set, save, or load all the necessary data describing the set of local models and their parameters as well as plot the results in a visually understandable way for the one- and two-dimensional RF cases. Figure 5.13 lists these publicly accessible methods.

```
function [rfDataX,rfDataW]   = get_rf_data_1D     (points)
function [C,lmDataX,lmDataY] = get_lm_data_1D     (points,scale)
function data                = get_rf_data_2D     (points,scale)
function params              = get_local_params   (xq)
function hndls               = init_plot_1D       (estX1Values,
idealEstimateFcn,trState,msrState)
function hndls               = update_plot_1D     (xNew,yNew,i)
function hndls               = init_plot_2D       (estX1Values,
estX2Values,idealEstimateFcn,trState,msrState)
function hndls               = update_plot_2D     (i)
function                       capture_plot_frame ()
```

Figure 5.13: Ease-of-use function of the *rfdelib* library.

5.2 Simulation experiments

In this section, we demonstrate the use of the *rfdelib* library in two simulation experiments. The first experiment in Section 5.2.1 works with a model of a nonlinear mechanical oscillator where the nonlinearity is only a function of one state, allowing us to distribute the receptive fields only along one axis. The second experiment in Section 5.2.2 shows the approximation of a two-dimensional nonlinear function.

5.2.1 Nonlinear oscillator

To test the behaviour of the *rfdelib* library, we use a second-order nonlinear system, which might represent a mechanical oscillator with a nonlinear spring. The system (5.5) is based on (4.27), extended by a second nonlinearity that spans only a part of the state space of the system, to force the algorithm to deal with nonlinearities on different scales. The system parameters are summarised in Table 5.1.

$$m\ddot{x} = b\dot{x} + k_1 x + k_3 x^3 + k_5 x^5 + k_6 e^{-\frac{(x-k_7)^2}{k_8}} + u \tag{5.5}$$

Parameter	Value
m	0.03
b_1	0.04
k_1	20
k_3	-50
k_5	40
k_6	7
k_7	0.6
k_8	0.001

Table 5.1: Table of the nonlinear oscillator parameters.

The system was simulated with a randomised input term u with a sampling rate $T_s = 10^{-3}$ s and we also added random noise with normal distribution to the system output to simulate an imperfect measurement. Figure 5.14 shows a trajectory in the state space of the system.

We used the noise-corrupted data to generate the system dynamics approximation using the *rfdelib* library with the numeric receptive field update method, default settings, and one-dimensional RF input space. Figures 5.15, 5.16, and 5.17 show the state on the system dynamics approximation with different amounts of datapoints. The figures depict both the shape and confidence interval of the approximation function and the spacing of the receptive fields.

In the first one, Figure 5.15, we can see that, with just 500 datapoints, a part of the receptive field input space ($x > 0$) is already covered by the well-learnt local models with narrow confidence intervals, however, the other part ($x < 0$) is covered by local models with very wide confidence intervals due to lack of datapoints in that region. Indeed, we can see that these models do not fit the true nonlinear function properly yet.

In the second one, Figure 5.16, we can see that the approximation is much better with 1500 datapoints, however, the distribution of receptive fields is not ideal and the highly nonlinear part of the function around $x \approx -0.6$ is not covered properly.

The last one, Figure 5.17, shows the situation with 6000 datapoints. It is clear that the distribution of the receptive fields is more optimised, meaning there are fewer, larger receptive fields in regions with less significant nonlinearities and the

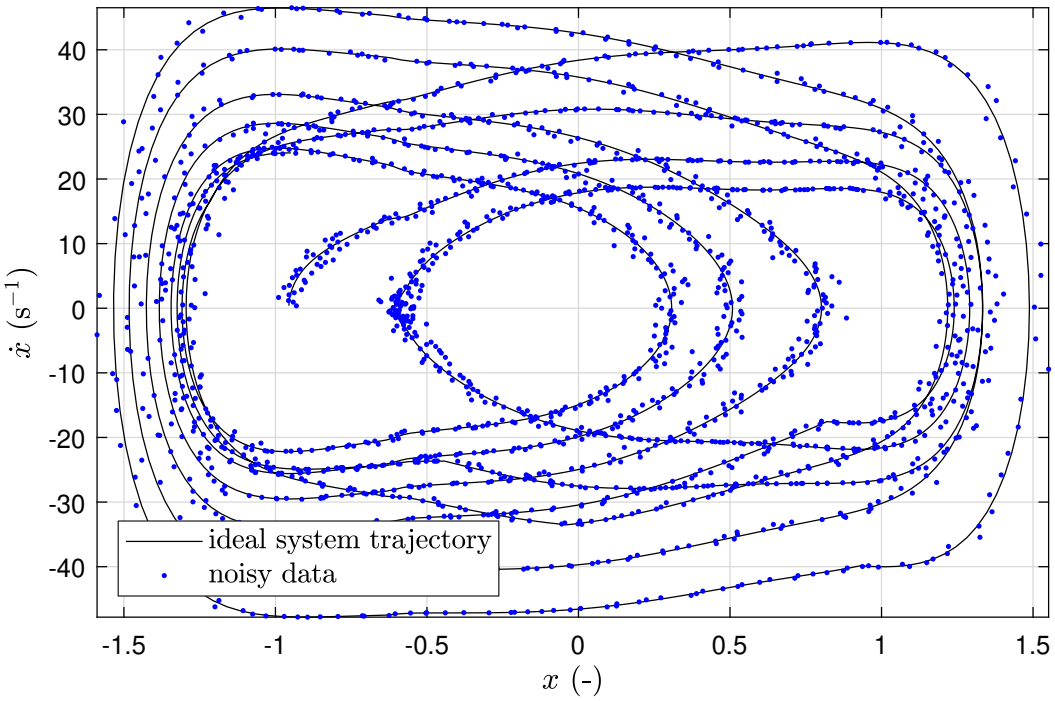

Figure 5.14: State space trajectory of the nonlinear system used for testing the *rfdelib* library, the plot shows both the ideal trajectory and noise-corrupted data used to generate the datapoints for the library to learn upon.

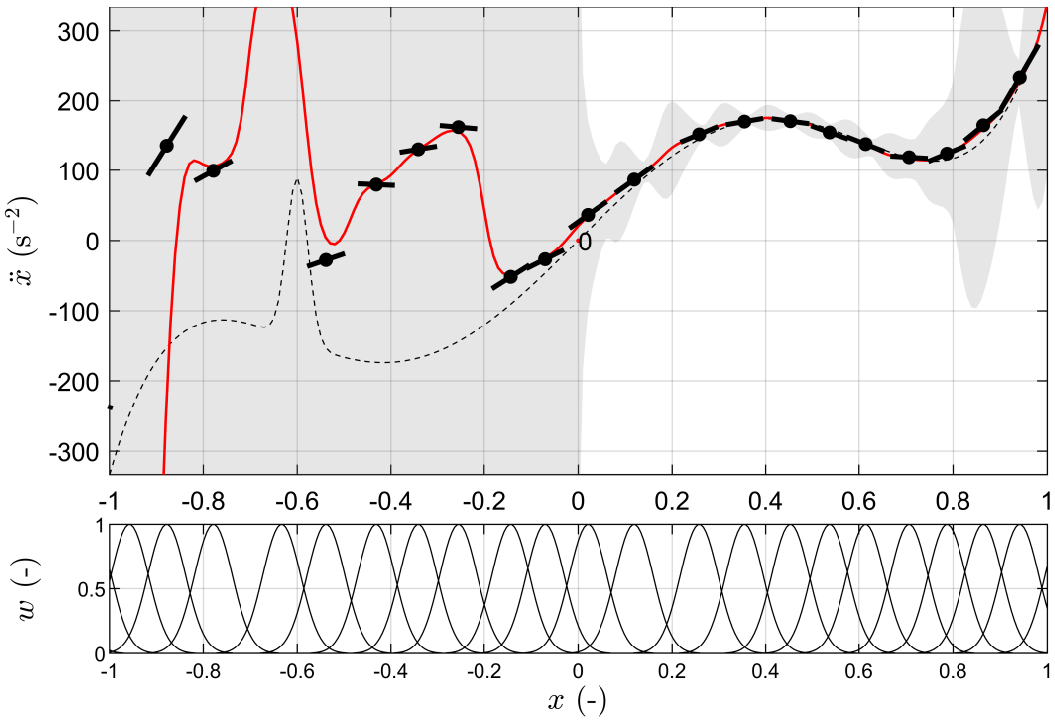

Figure 5.15: *rfdelib* approximation of the nonlinear system dynamics with 500 datapoints.

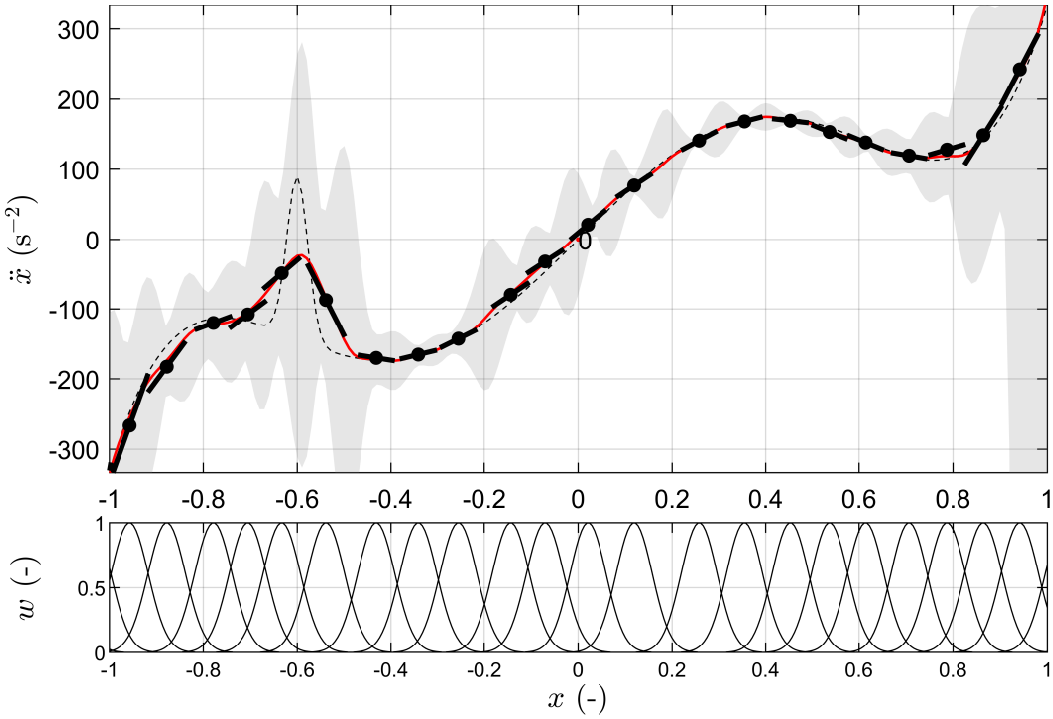

Figure 5.16: *rfdelib* approximation of the nonlinear system dynamics with 1500 datapoints.

local models begin to cluster more around $x \approx -0.6$ to fit the highly nonlinear part of the function. Also, the confidence interval overall is much narrower.

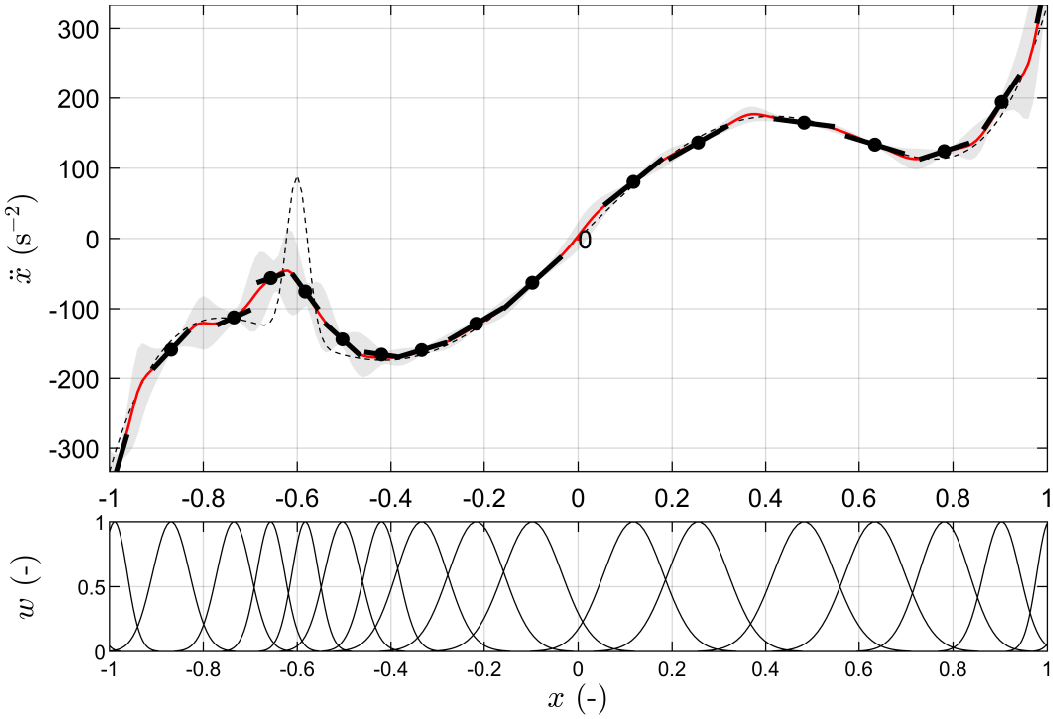

Figure 5.17: *rfdelib* approximation of the nonlinear system dynamics with 6000 datapoints.

5.2.2 2D nonlinear function

In this section, we demonstrate the function of the modified RFWR algorithm implemented in the *rfdelib* class on an example of a two-dimensional nonlinear function described by (5.6). In this case, the library is set up for a two-input local model with both inputs used for the RF distribution.

$$y = sin(5x_1) + x_2^2 \tag{5.6}$$

We generated uniformly distributed random points in the input space with $x_1, x_2 in < -1; 1 >$ and then used them to generate noise-corrupted datapoints to teach the approximation as in Figure 5.18.

```
fcn = @(x1,x2) sin(5*x1) + x2.^2;
x1 = linspace(-1,1,50)';
x2 = linspace(-1,1,50)';

LM = rfdelib(2,'rfInputToggling',[1 1]); % init library

% parameters
LM.confTarget = 0.1;
LM.alpha = 0.08;
LM.defaultDistanceUT = 0.2;
Ndata = 1e5;

LM.init_plot_2D(x1,x2,fcn,[],[]);

% teach fcn aproximation
for i = 1:Ndata
    x = rand(1,2)*2 - 1;
    y = fcn2D(x(1),x(2)) + randn*0.1;

    LM.learn(x,y);

    if mod(i,100) == 0
        LM.update_plot_2D([]);
    end
end
```

Figure 5.18: Code example for the two-dimensional nonlinear function approximation.

The results are summarised in Figures 5.19, 5.20, and 5.21. Each figure displays the approximated function, confidence interval estimation, and receptive field distribution corresponding to the incremental number of datapoints: 250, 750 and 3500, respectively. To validate the results, we calculated the $RMSE$ value between the approximated function and the true analytical function on a 50-by-50 grid in the input space. In Table 5.2, we can see that the $RMSE$ values decrease with an increasing number of datapoints.

Number of data-points	RMSE	Confidence interval size average	Number of local mod-els
250	0.233	0.745	22
750	0.149	0.322	36
3500	0.087	0.112	51

Table 5.2: Summary of the 2D function approximation by the RFWR algorithm with a different number of datapoints.

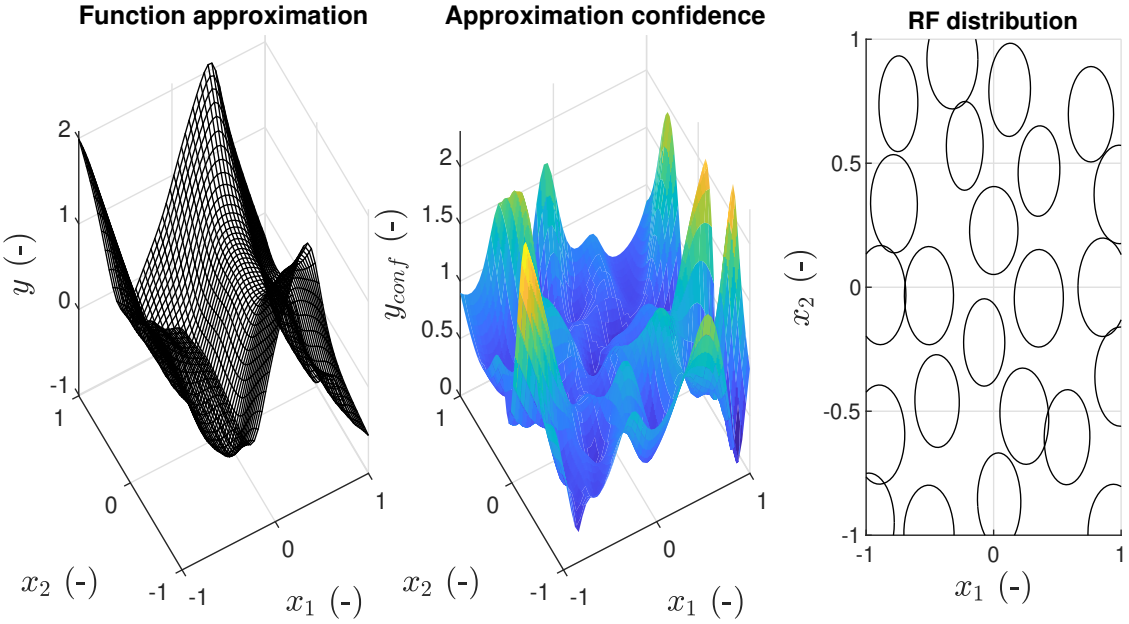

Figure 5.19: Nonlinear two-dimensional function approximation using RFWR after 250 datapoints with the approximation $RMSE = 0.233$.

Furthermore, Figure 5.22 shows a detailed comparison of the absolute error of the approximation of the function and the estimated confidence interval size maps depicted using the greyscale colourmap. Both maps are highly correlated which proves that the confidence interval estimation algorithm is working correctly.

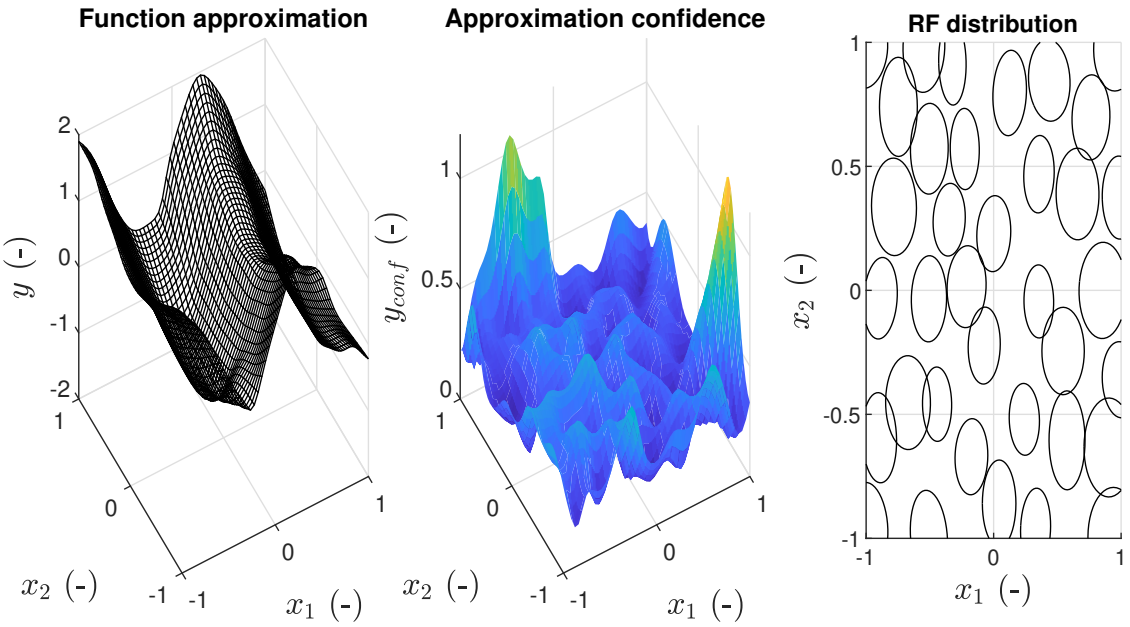

Figure 5.20: Nonlinear two-dimensional function approximation using RFWR after 750 datapoints with the approximation $RMSE = 0.149$.

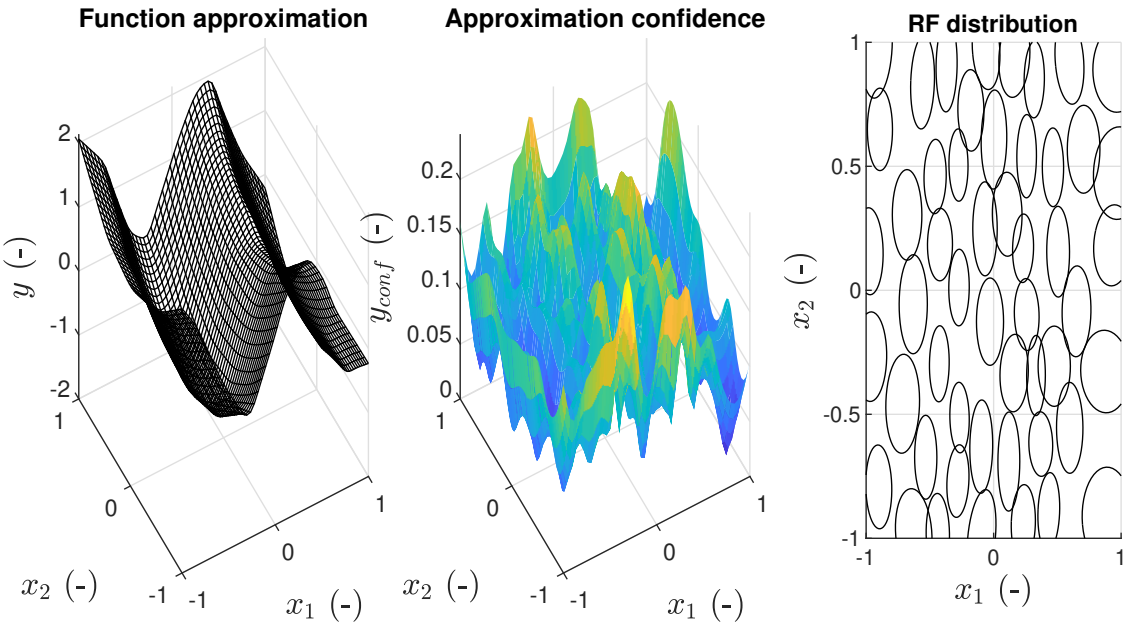

Figure 5.21: Nonlinear two-dimensional function approximation using RFWR after 3500 datapoints with the approximation $RMSE = 0.087$.

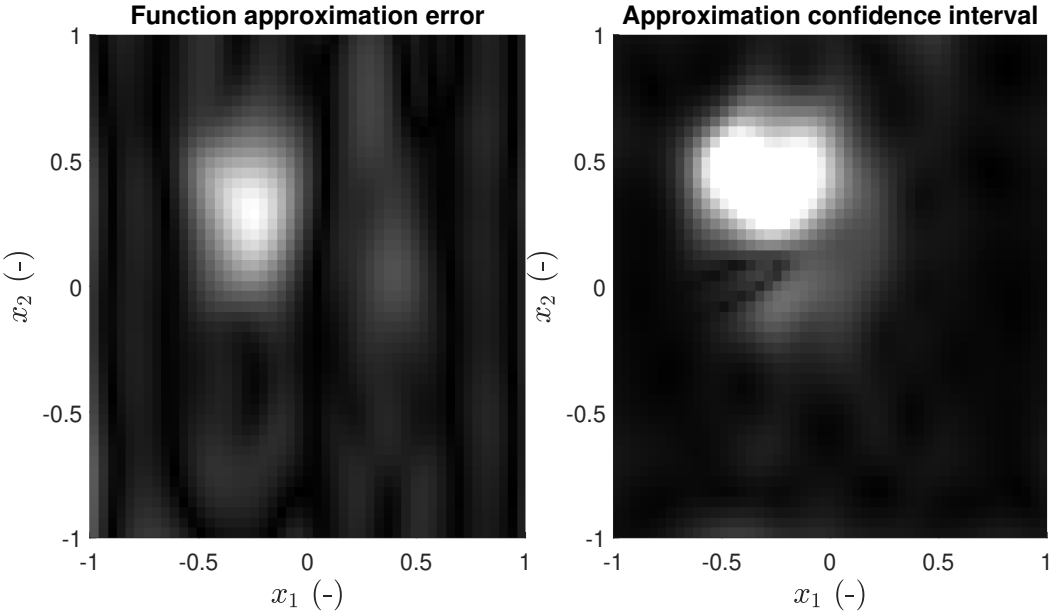

Figure 5.22: Nonlinear two-dimensional function approximation absolute error and confidence interval map comparison with 1125 datapoints, right after a new local model was added.

5.3 Chapter summary

In this chapter, we present the modification of the RFWR algorithm suitable for incremental, low-dimensional approximation tasks on multi-domain mechatronic systems together with a user library for the Matlab language which contains a number of modifications and improvements compared to the original RFWR algorithm described in [33]. The most important modifications are: several different methods for the receptive field spacing update, separating the receptive field and the local model state space, adding a stabilising data buffer for each local model etc.

Then, in Section 5.2 we demonstrate the performance of the algorithm in the one- and two-dimensional cases to prove that it is able to learn the system dynamics.

6 Dual estimation based on RFWR and Kalman filter

In this chapter, we describe the final extension of the *rfdelib* library introduced in the Chapter 5. The extension implements the ideas on the Kalman filter modification presented in Chapter 4. In terms of the various simultaneous estimation approaches described in Section 2.4, the combination of the RFWR algorithm for model and parameter estimation and the modified KF for state estimation forms a hybrid dual estimation algorithm, hence, the abbreviation RFDE in *rfdelib* that stands for Receptive Field Dual Estimation.

The RFDE implementation is described in Section 6.1, and then, in Sections 6.2 and 6.3, we present simulations and experiments with real dynamic systems that demonstrate the function of the library.

6.1 Hybrid dual estimation library for MATLAB

In Section 4.5, we presented the dual estimation with a single local linear model. The *rfdelib* library implements this algorithm separately for each existing local model in the RFWR structure. Building on the diagram in Figure 4.11 which presents the algorithm for a single, fixed, local model, Figure 6.1 presents a similar diagram for the RFDE algorithm, showing the interconnection between the RFWR local model set and the modified KF calculated for each of the local models.

The dual estimation function must be turned on when the library is initialised, see Figure 6.2 for a code example. Note that the library assumes that the system matrix F is in discrete canonical form with the system state forming a sequence of derivatives (discrete time continuous state space system), where all the states are being measured. The library also requires the user to set the discrete sampling time T_s, the measurement noise covariance matrix $\boldsymbol{R}$ and the default process noise covariance matrix $\boldsymbol{Q}_0$, which is added to the process noise covariance estimated based on the modified KF algorithm. The library allows for $\boldsymbol{Q}_0 = \mathbf{0}$.

There can be a discussion whether to use discrete or continuous state space, meaning whether the state vector should be a list of past states or a sequence of corresponding derivatives. Both approaches are valid and theoretically equivalent and the use of a discrete state space would lead to several simplifications in the implementation while linking the modified KF and the RFWR algorithms. However, previous experiments and publications on the topic, such as [47, 28] led us to the conclusion that the continuous state vector is better suited for parameter estimation and more often than not corresponds to the measurements made on the system. In

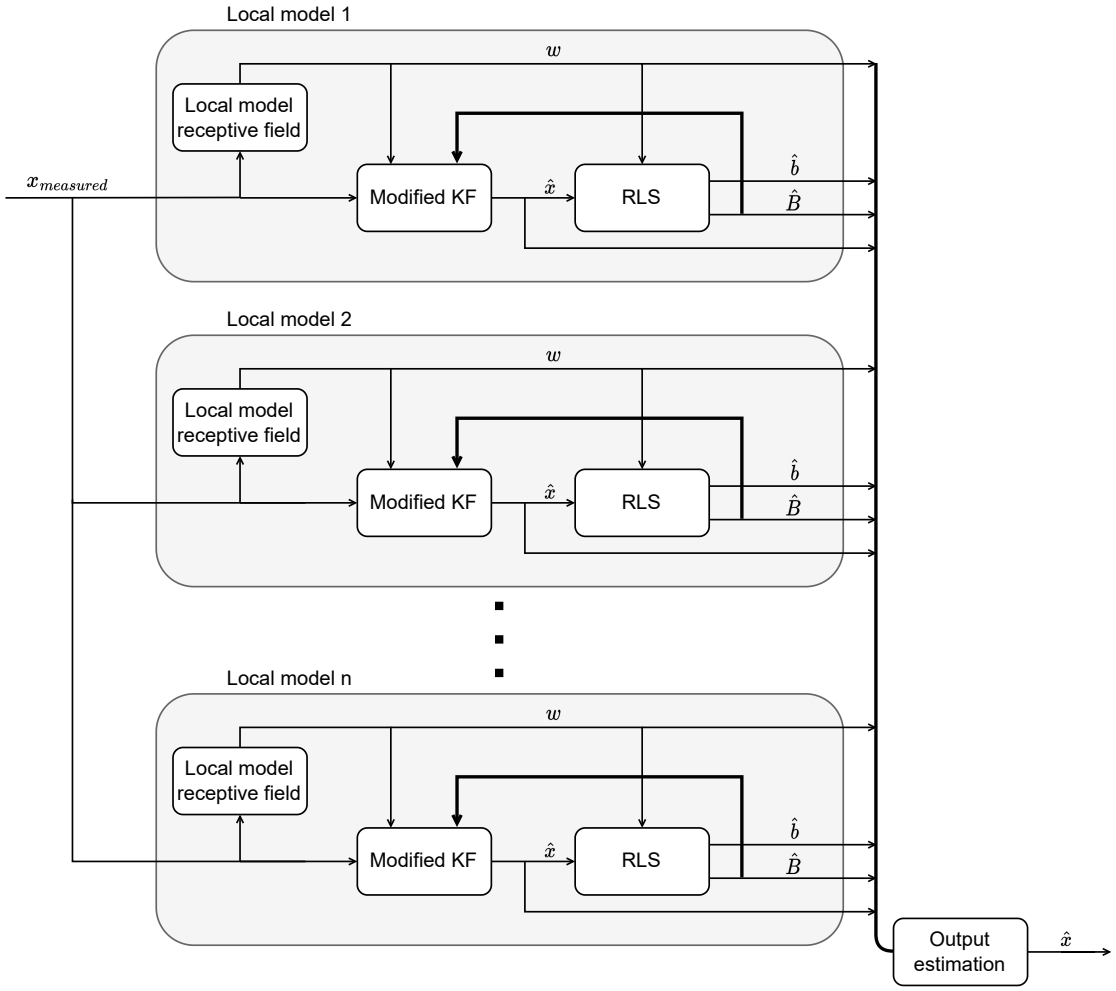

Figure 6.1: The diagram of the *RFDE* algorithm showing the interconnection between the RFWR local model set and the modified KF calculated for each of the local models. $\boldsymbol{x}$ represents the system state, w represents the actual local model weight, $\boldsymbol{b}$ represents the local model parameters and $\boldsymbol{B}$ represents the model parameter covariance.

```
LM = rfdelib(3,'rfInputToggling',[1 0 0],...
'dualEstimation','on','Ts',Ts,...
'kfMeasureNoiseCovariance',diag([r11 r22]),...
'kfProcessNoiseCovariance',diag([q11 q22]));
```

Figure 6.2: Example code demonstrating the initialisation of the *rfdelib* library to use the RFDE functionality.

RFDE, the RFWR local models are built upon continuous-time continuous-state space systems, however, the modified KF uses the Euler discretization of the corresponding local model.

The last assumption that the library requires is that the new datapoints come as a sequence of measurements in time, since the KF tracks the states of the system.

This is a requirement which is not necessary for the use of the RFWR algorithm in general as it performs the optimisation on a static basis without any internal states.

It is also important to consider the behaviour of a single KF with local validity. As the state being tracked goes out of the validity region described by the RF of the corresponding local model, the filter starts to give increasingly more weight to the measurement compared to the model prediction. From a certain point, the filter basically only passes the measurement without any correction, which ensures the stability of the filter while the system is in a state where the local model is not valid. The *rfdelib* even implements this behaviour in such a way that from the weight limit corresponding to the w_{act} RF activation limit it skips the KF prediction and correction step and only passes the measurement through to save computational power.

When a new datapoint is presented, *rfdelib* uses an extended *learn* method which implements the link between the state and parameter estimation. The new *learn* method code is shown in Figure 6.3.

```
function  learn (h,x,y)
    W = h.get_lm_weight(x,0,0);

    % update lm params using RLS and distance metric
    if h.dualEstimationOn == 1
        xKf = h.update_kf(x,W);
        h.update_lm(xKf,y,W);
    else
        h.update_lm(x,y,W);
    end

    h.add_lm(x,y,W); % add new lm&rf if necessary

    h.prune_lm(W)% pruning obsolete rf if necessary

    h.global_parameter_matching();

end
```

Figure 6.3: Implementation of the *learn* method in the *rfdelib* class with the RFDE functionality.

6.2 Simulation experiments

In this section, we present the results of a simulation experiment that corresponds directly to those presented in Section 5.2.1. We used the same system, the nonlinear oscillator, with the same parameters as in (5.5) but instead of using the modified RFWR algorithm presented in the Chapter 5, we applied RFDE to approximate the system dynamics.

Figures 6.4, 6.5, and 6.6 show the results of the simulation in different stages of the learning process, i.e. with 1480, 3860 and 7800 datapoints, respectively.

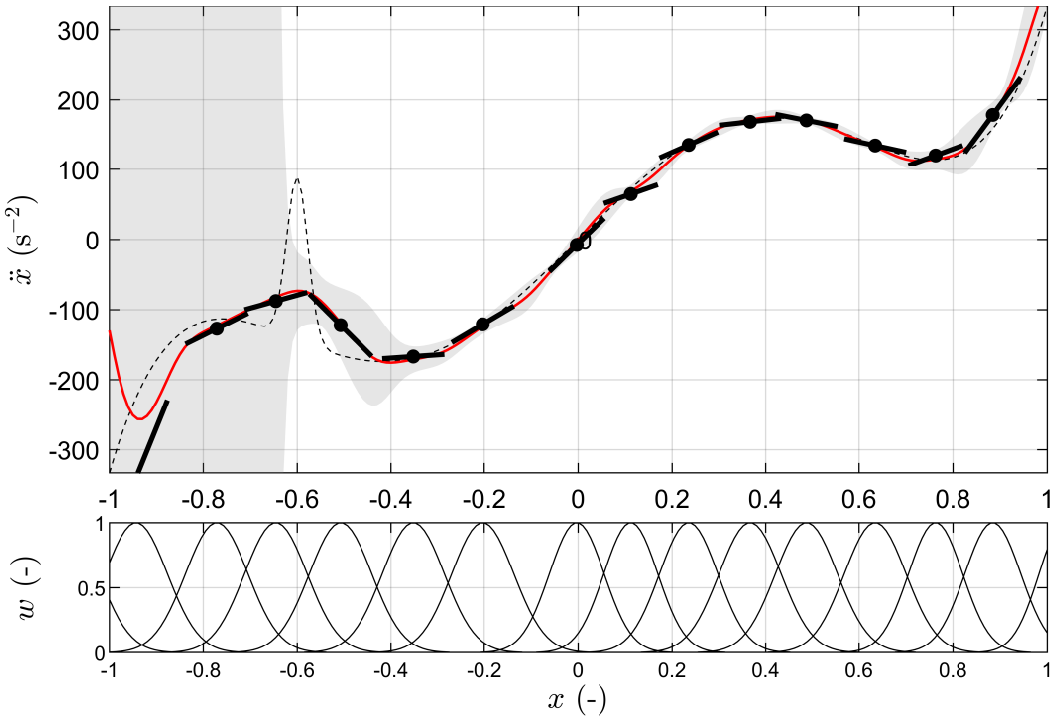

Figure 6.4: *rfdelib* approximation of nonlinear system dynamics with 1480 datapoints.

Lastly, Figure 6.7 shows the tracking signal of the x state of the system for the first 1 s of the learning process. We can see that in the beginning ($t < 0.55s$), the signal is noisy, faithfully following the measurement signal. After that, the *RFDE* algorithm was able to increase the precision and subsequently narrow the confidence interval in a few local models placed around $x \approx -0.7$ to such a degree that the KF started taking these models' predictions into account and the noise vanished while the signal closely followed the true state. In addition, the figure also shows the running $RMSE$ value of the KF tracking error, corresponding to the previous observation, the $RMSE$ value drops almost one order of magnitude when the precise models with high confidence are taken into account.

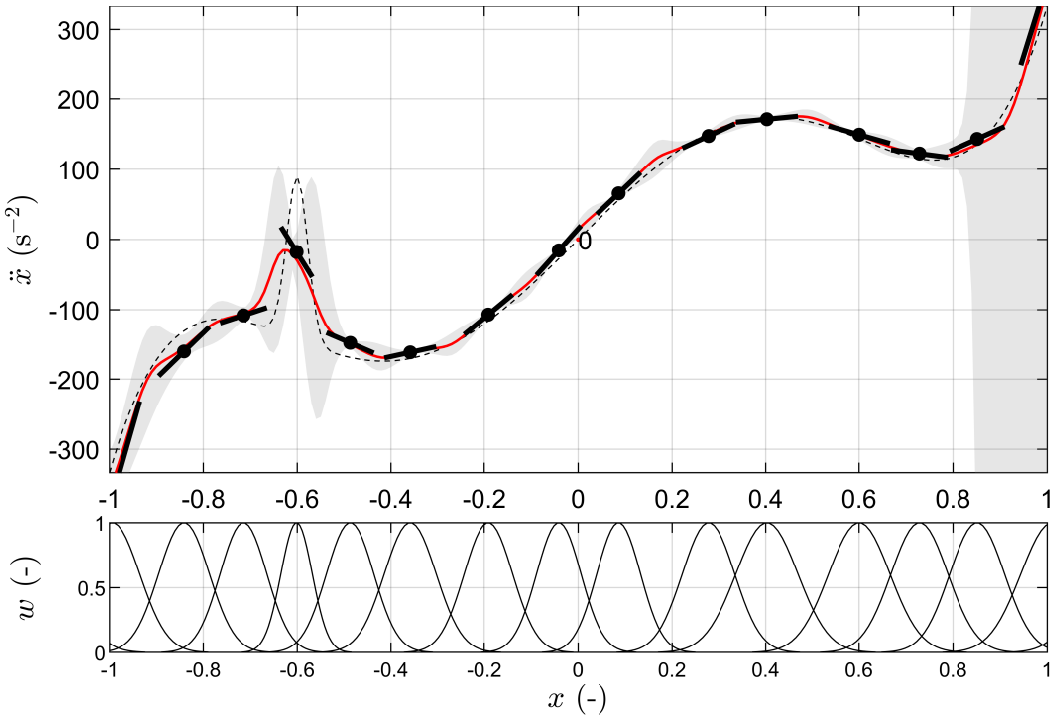

Figure 6.5: *rfdelib* approximation of nonlinear system dynamics with 3860 datapoints.

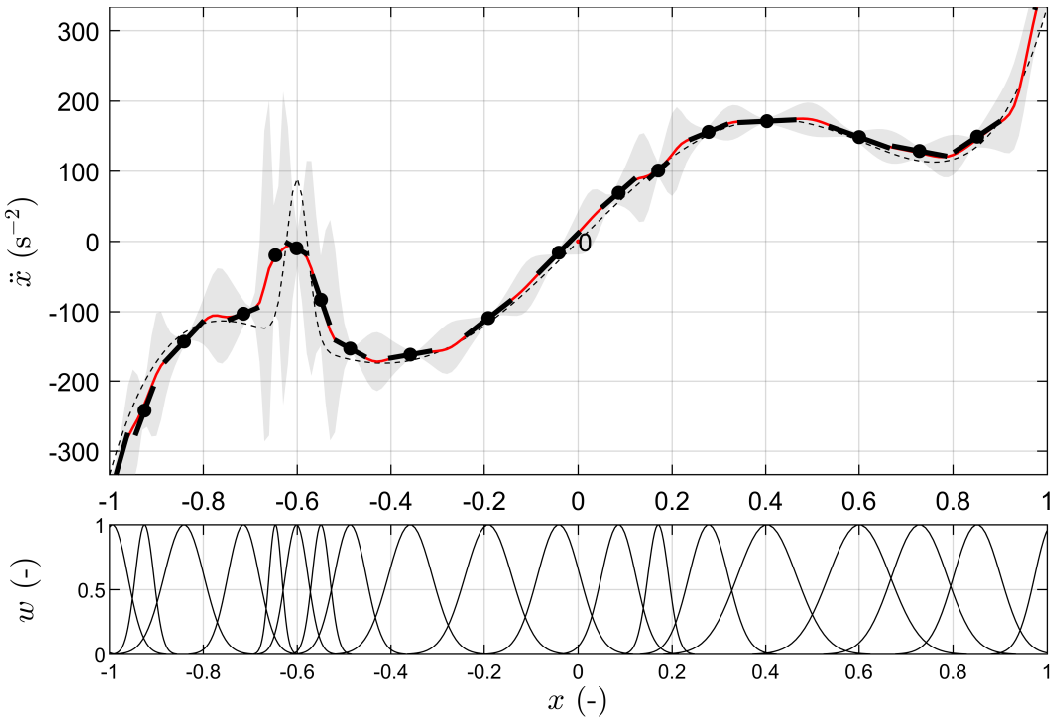

Figure 6.6: *rfdelib* approximation of nonlinear system dynamics with 7800 datapoints.

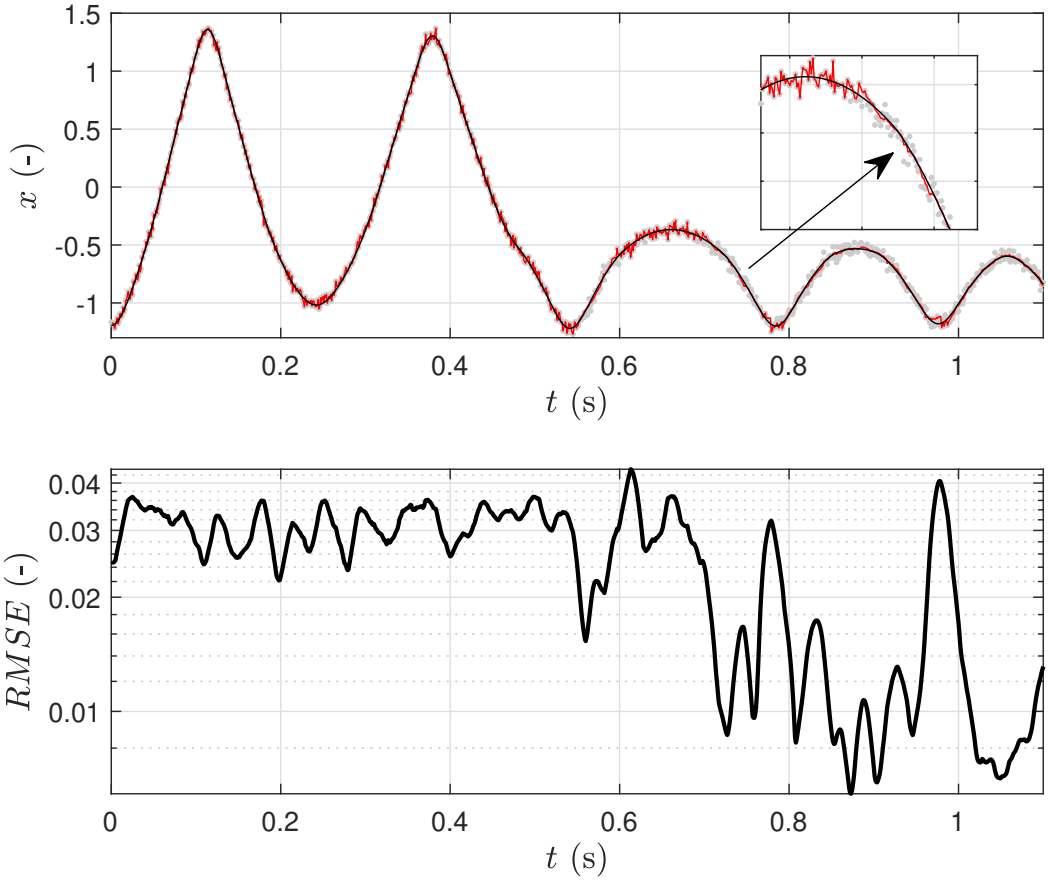

Figure 6.7: *rfdelib* tracking of the state x, showing the transition to dual estimation when the quality of local model around $x \approx -0.7$ improves significantly.

6.3 Case study - Magnetic manipulator

To test the *rfde* library under real conditions, we decided to perform a case study of hybrid simultaneous estimation on a laboratory model of a magnetic manipulator, the same one that was used in [47] and initially presented in [48].

The magnetic manipulator, shown in Figure 6.8, consists of a row of four coils and an iron ball. The position of the ball, measured using a laser distance sensor can be controlled by varying currents in the coils, which are driven by a specialised power electronic unit set through Matlab commands. The communication between the Matlab environment and the power control unit is carried out using the Humusoft MF634 PCIe IO card [49].

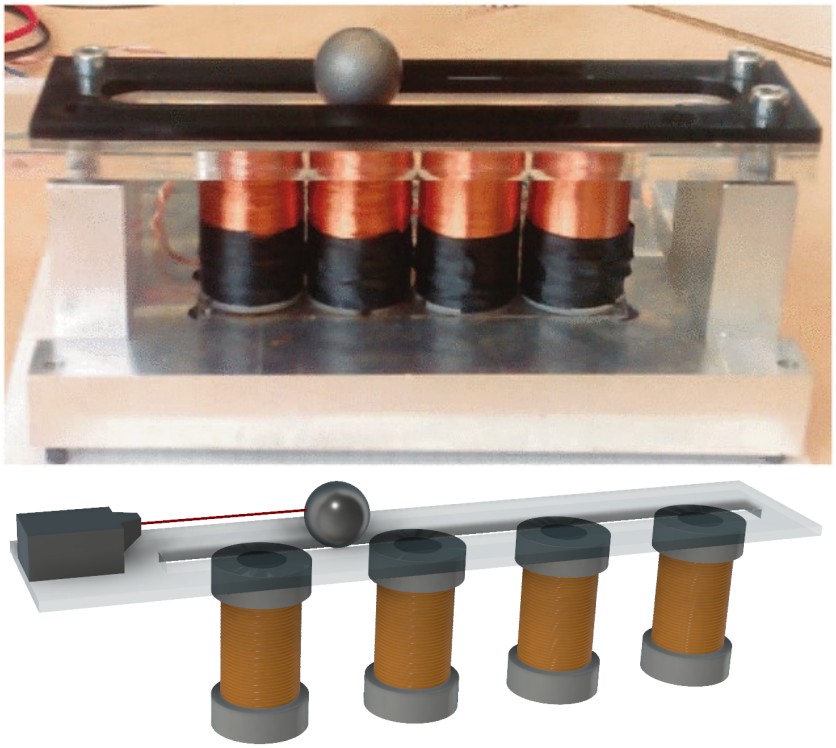

Figure 6.8: Photo and a 3D render of the magnetic manipulator with a row of coils and an iron ball in a linear pathway. Taken from [48] and [47], respectively.

The main dimensions and functional parameters of the manipulator are summarised in Table 6.1. When studying the dynamics of the system, it is important to create a model for a single coil, as the effect of multiple coils can be viewed as additive ([48, 47]), especially when the system is intended to be controlled in such a way that only a single coil is active at any time. For this reason, we focus on the simultaneous estimation of a system with the iron ball and a singe coil. This system can generally be described using an explicit state model (6.1).

$$\ddot{x} = f\left(x, \dot{x}, u\right) \tag{6.1}$$

Parameter	**Value**	**unit**
Ball mass	53	g
Ball diameter	20	mm
Distance between the coil edge and the ball position limit	20	mm
Distance between the coil centres	25	mm
Maximal coil current	0.6	A
Sampling period	0.005	s

Table 6.1: Magnetic manipulator parameters. [47]

where x is the position of the ball and u is the system input corresponding to the coil current.

According to [48], considering zero velocity, the force exerted on the ball in the longitudinal direction along the ball's pathway is highly nonlinear. The shape of the nonlinear function is depicted in Figure 6.9.

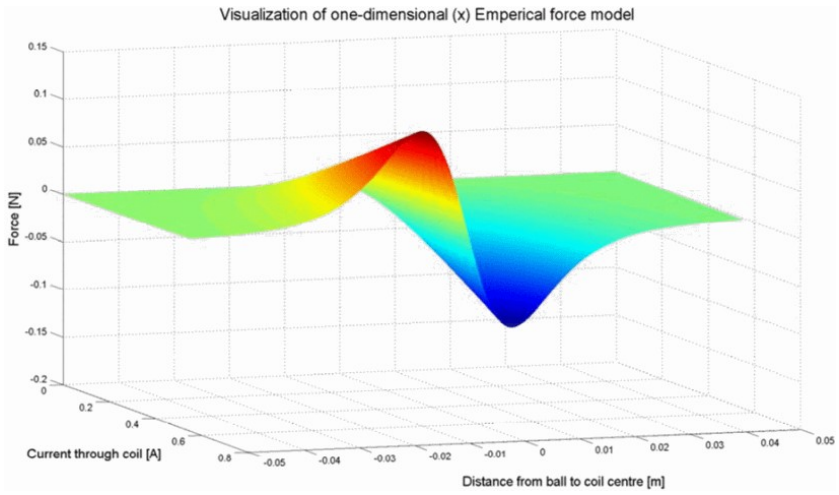

Figure 6.9: Nonlinear magnetic force exerted on the iron ball along the ball's pathway direction as a function of the position and current. Taken from [48].

Using the RFDE method, the system can be studied in multiple ways. Figure 6.9 shows that the system is highly nonlinear along the ball's position x and less significantly along the current u. Also, [47] shows that there is a less significant nonlinear dependence on the ball's velocity $\dot{x}$ too. In Section 6.3.1, we study various combinations of the receptive fields distributions in one and two dimensions.

To acquire a database for the subsequent experiments, we created a data set 60 s long with sampling period $T_s = 10^{-3}$ s containing the time, coil current, position, velocity, and acceleration of the iron ball. Figure 6.10 shows a 7 s interval of the ball's position x and the coil current u that was generated randomly with uniform random distribution of both the current level, where $u \in < 0; 0.6 >$ A, and the pulse length $T_{pulse} \in < 0.05; 1 >$.

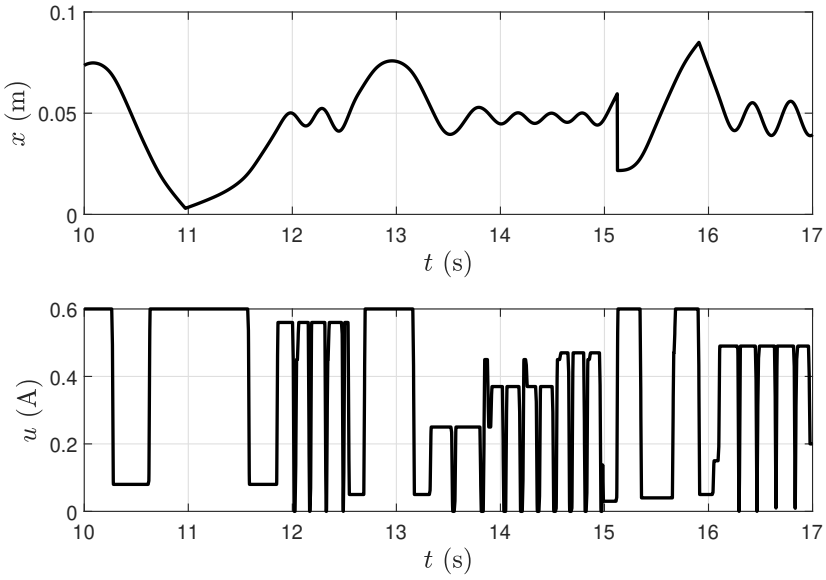

Figure 6.10: Part of the data-set used in the later experiments with RFDE. The Input signal u was generated using the random - random amplitude signal generation procedure with uniform random sampling.

6.3.1 Hybrid dual estimation with the Magnetic Manipulator

In this section, we present the results of the RFDE algorithm applied to the iron ball magnetic manipulator. In all the cases, the algorithm is set for three local model inputs (x, $\dot{x}$ and u) and one output ($\ddot{x}$), generating local models in the form of (6.2).

$$\ddot{x} = \boldsymbol{b}^T \boldsymbol{x} = b_0 x + b_1 \dot{x} + b_2 u + b_3 \tag{6.2}$$

where $\ddot{x}$ is the local model output: acceleration of the ball, $\boldsymbol{b}^T = [b_0, b_1, b_2, b_3]$ is the vector of parameters of any given local modal and $\boldsymbol{x}^T = [x, \dot{x}, u, 1]$ is the vector of local model inputs: position, velocity, and coil current, respectively, with respect to the centre of the corresponding receptive field.

First, we set the RFDE algorithm so that only the position of the ball is considered when plating the receptive fields of the local models, making the distribution one-dimensional. The result is shown in Figure 6.11 after learning with 7400 datapoints. We can see that the algorithm was able to adapt and to find the shape of the underlying nonlinear function similar to Figure 6.9, however, it contains a great deal of uncertainty and the shape of the function is not precise. The cause of this imprecision is that the ball velocity also brings a nonlinear effect to the system behaviour. According to [47], the effect is caused by the induced eddy currents generated in the iron ball material while it moves through the changing magnetic field.

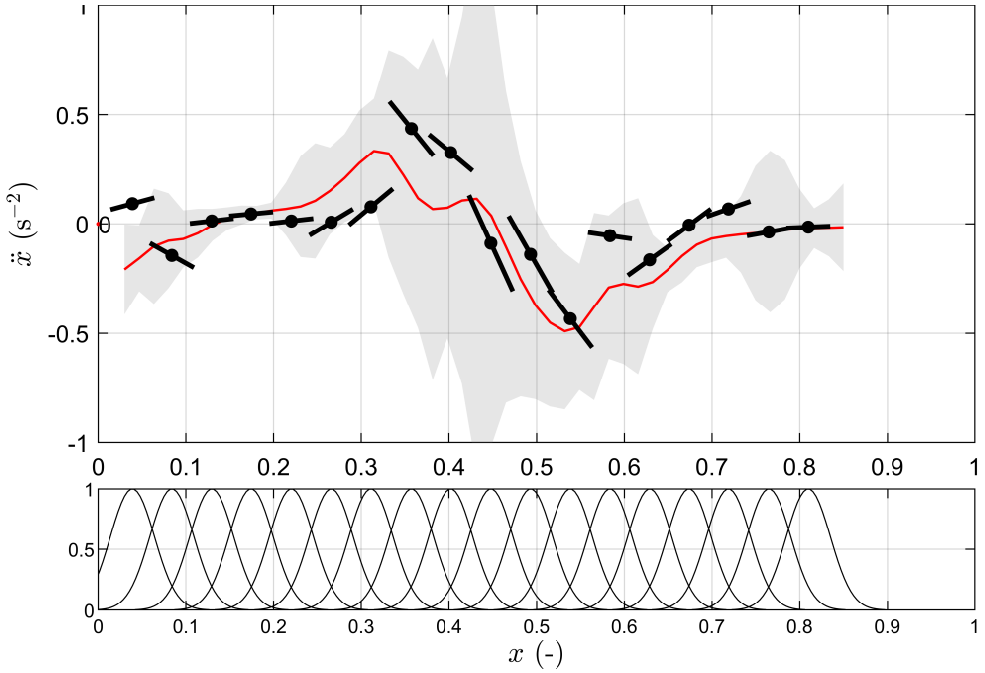

Figure 6.11: The model shape learned by the *RFDE* algorithm with one-dimensional RF distribution along the x dimension after 7400 datapoints. The figure shows the dynamic function shape for $u = 0.5$ A and also the distribution of the RFs.

In the second experiment, we set the RFDE algorithm so that both the ball's position and the velocity are considered when placing the receptive fields. The result is shown in Figure 6.12. The figure shows the shape of the dynamics of the system in the RF space for $u = 0.5$, as well as the size of the model confidence interval, after 17400 datapoints. Again, we can see that the characteristic nonlinear shape is present along the x axis, however, the coil effects are lower in higher velocities, representing the effects of the eddy currents. We can also see that the model is imprecise on the edges of the space, corresponding to wider confidence intervals, which is caused by the lack of enough datapoint in those regions. Also, the confidence interval size is much lower than in the one-dimensional case.

Lastly, Figure 6.13 shows the distribution of the receptive fields corresponding the the result shown in Figure 6.12.

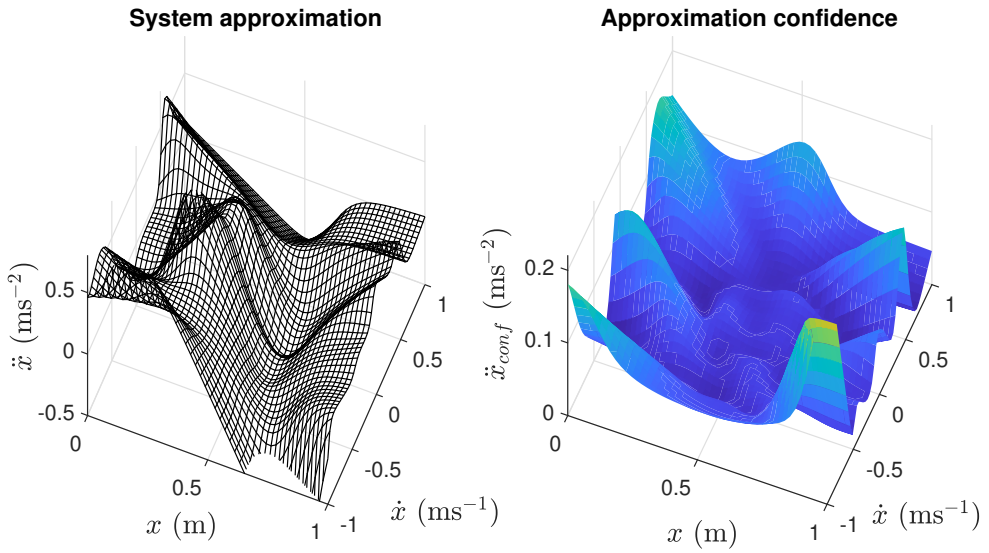

Figure 6.12: The model shape learned by the *RFDE* algorithm with two-dimensional RF distribution along the x and $\dot{x}$ dimensions after 17400 datapoints. The figure shows the dynamic function shape for $u = 0.5$ A and the corresponding size of the confidence interval.

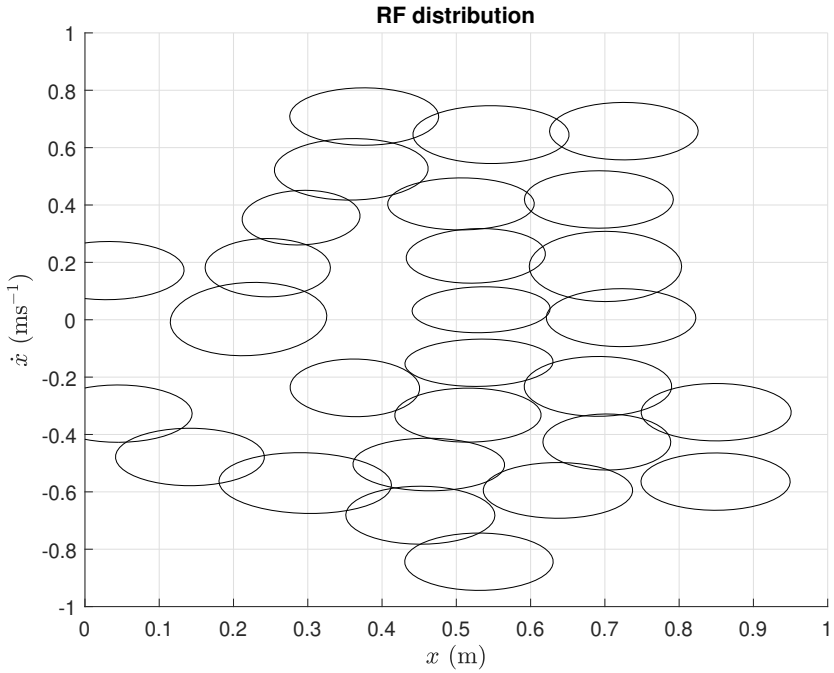

Figure 6.13: The model RF distribution learned by the *RFDE* algorithm with two-dimensional RF distribution along the x and $\dot{x}$ dimensions after 17400 datapoints.

6.4 Chapter summary

In this Chapter, we combined the results developed and presented in the previous two Chapters, the modified Kalman filter and the modified RFWR algorithm, to form the Hybrid dual estimation algorithm called the Receptive Field Dual Esti-

mation (*RFDE*). The algorithm is able to track the states of a dynamic system while continually learning the system dynamics with minimal requirements on user parameter tuning compared to other simultaneous estimation algorithms.

In Sections 6.2 and 6.3 respectively, we performed simulation and practical experiments to demonstrate the results and to show that the algorithm is able to robustly learn even from very noisy data.

7 Conclusion

The research described in this thesis was motivated by engineering experience and previous research, published in [50], concerning all three basic tasks described in the Introduction, that is, the modelling, state estimation, and control algorithm design for nonlinear dynamic systems. It turns out that, individually, these tasks are theoretically mostly solved and that the used methods work. However, their practical implementation raises many questions for which there are no clear answers. Also, while solving these tasks separately, we often arrive at suboptimal solutions due to the criteria used for the separate model parameter estimation, and the observer or controller tuning is defined as being unrelated even though they all contribute to the overall control quality.

The four main goals for this dissertation were defined in the presented thesis and approved at the State Doctoral Exam:

1. Research of the mutual relationships in typical algorithms used for estimating the states and parameters of dynamic system models.

2. Modification of the RFWR method for its use with typical mechatronic systems.

3. Developping a Hybrid method for the simultaneous modelling and state estimation of nonlinear dynamic systems.

4. Implementation and testing of the resulting algorithms in a real system.

Within the first goal, we mainly studied the uncertainty of dynamic system parameters estimated on the basis of noisy data. This uncertainty is one of the sources of estimation errors in state estimation algorithms. We identified it as being dominant in situations where the structure (ordinary differential equation) of the model can be reliably determined, we do not work with principally stochastic systems, and the discretisation error is also not significant due to the sampling period being orders of magnitude shorter than the natural time constants of the system in question. Chapter 4 further describes the use of our discoveries for the reformulation of the classical Kalman filter algorithm for this very specific situation. We were able to achieve recognisable improvement in the state estimation accuracy, but most importantly, we were able to find an empirical approach for determining the tuning parameter most critical for the Kalman filter's practical implementation - the process noise covariance. Thanks to this development, we were able to create a parameterless link between the modified Kalman filter and a method for the parameter estimation of a local linear model of an otherwise nonlinear system. This link forms a hybrid dual estimation approach. We were also able to demonstrate its functionality and

improved state and parameter estimation accuracy compared to that of a typical approach, thus fulfilling the first goal of the thesis.

Chapter 5 deals with the modification of the Receptive Field Weighted Regression (RFWR) originally described in [29, 31] and the development of a user library for MATLAB. We have rewritten the method into an incremental version to be used online without the need for batch data processing. This modification was motivated by the inherent potential of incremental methods to work as adaptive approximators in situations in which the system that we observe can change its behaviour. For example, due to the wear and tear of its parts, changing external conditions, etc.

Additionally, we expanded the modified RFWR algorithm by several useful elements. Most importantly, allowing for a separate number of dimensions in the local models and the corresponding receptive fields. This is useful in situations where the nonlinearity of the system is exclusively or dominantly dependent on a subset of the system states and when other system dynamics are presumed to be linear. Other modifications included minor adjustments, such as four different variants of the receptive field optimisation algorithm, an improved computational performance, a learning data buffer for each local model, calculation of the resulting approximation confidence interval, etc. Thus, the second goal can also be considered achieved.

Furthermore, in Chapter 6, we describe the *Hybrid Method for the Simultaneous Modelling and State Estimation of Nonlinear Dynamic Systems* built upon the results of previous research goals. This method, in the form of dual estimation, combines the RFWR and KF modified for the situation with uncertain parameters. We tested the hybrid dual estimation approach through several simulation experiments and finally on data measured in a real system of a magnetic manipulator, achieving the third and fourth research goal of the thesis.

7.1 Thesis achievements

- **Categorisation of error sources for the Kalman filter (KF) process model**
  In Chapter 4, we described and categorised the possible sources of errors that influence the KF process model, which may arise in common engineering applications. This methodology can be helpful in practical KF implementations and especially in tuning the values of the process noise covariance.
- **Modification of the KF for situations with inaccurate and uncertain process model parameters**
  Based on previous research, published in [51], we modified a part of the Kalman filter algorithm, specifically the state vector covariance prediction step, to better suit the situation where the dominant source of the error is inaccurate or contains uncertain process model parameters, especially parameters estimated based on noisy data. In our experience, this situation is far more common than is assumed, although very little scientific attention is paid to it. The results show that this modified KF gives better results in conditions with the

parameter uncertainty being the dominant source of the process noise.

- **Empirical approach for setting the process noise covariance**
  We found and experimentally verified an empiric formula to find the value of the process noise covariance for the Kalman filter in this very specific situation, greatly simplifying the practical implementations. It turned out that it is mostly important to set the correct order of the process noise covariance values and this empirical formula proved to give an a reasonably accurate and robust estimate.

- **Linking the KF and RLS algorithms for the dual estimation over a local linear model**
  We further modified the Kalman filter algorithm so that it can be used with local linear models that are linked to a parameter estimation method, for example, RLS. Mainly, this means implementing the KF in a way that it can work in a shifted state and input space corresponding to the centre of the local model validity function and using the validity function to weight the RLS estimate as well as the KF prediction.

- **Modification of the RFWR algorithm**
  We modified the RFWR function approximation method, which is based on the principle of local linear models described in [29], so that it is better suited to be used with common low-order mechatronic systems, which typically have a significant nonlinearity dependence on a subset of states, and the state space is formed by sequences of quantities in integral-derivative relationships. Some of these modifications were published in [47]. We also developed a user library for the MATLAB language.

- **Hybrid dual estimation**
  By combining the modified RFWR method and the modified Kalman filter, we developed a *Hybrid Method for the Simultaneous Modelling and State Estimation of Nonlinear Dynamic Systems*, which uses a separate Kalman filter for each local model whose parameters are estimated. We validated the functionality of this method both in simulation and experimentally, and further developed a user library for the MATLAB language, which allows for its simple and fast practical implementation.

7.2 Future research possibilities

The research described in this thesis can be further continued in several areas. First, the idea of a Kalman filter applied in situations with uncertain parameters could also be used with other filters from the Bayesian recursive filter family - the Extended Kalman filter (EKF), Unscented Kalman filter (UKF), or Particle filter (PF) - designed for nonlinear systems.

Second, there is the possibility for further research of the simultaneous estimation methods, especially applied directly on nonlinear systems, e.g., using the Particle

filter and a different (possibly global) approximation method or focussing on higher order systems [30].

Lastly, going back to the original motivation for this thesis, another research direction that comes to mind is linking the simultaneous estimation with the third basic task, i.e., designing the control algorithm. This would make the quality of the control process the only criterion for both the state and parameter estimation.

References

[1] BOHNER, Martin; PETERSON, Allan. Dynamic Equations on Time Scales. *Dynamic Equations on Time Scales*. 2001. Available from DOI: 10.1007/978-1-4612-0201-1.

[2] ELLNER, Stephen P.; GUCKENHEIMER, John. Dynamic models in biology. *Dynamic Models in Biology*. 2011, vol. 9781400840960, pp. 1–329. ISBN 9781400840960. Available from DOI: 10.2307/j.ctvcm4h1q.

[3] D.SC, Balth. van der Pol Jun. On "relaxation-oscillations". 2009, vol. 2, no. 11, pp. 978–992. ISSN 1941-5982. Available from DOI: 10.1080/14786442608564127.

[4] BOYCE, William E.; DIPRIMA, Richard C. Elementary differential equations and boundary value problems. [N.d.], p. 809. ISBN 978-0470458310.

[5] NELLES, Oliver. Nonlinear System Identification. *Nonlinear System Identification*. 2001. Available from DOI: 10.1007/978-3-662-04323-3.

[6] KALMAN, R. E. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*. 1960, vol. 82, no. 1, pp. 35–45. ISSN 0021-9223. Available from DOI: 10.1115/1.3662552.

[7] *Kalman and Bayesian Filters in Python by rlabbe*. [N.d.]. Available also from: http://rlabbe.github.io/Kalman-and-Bayesian-Filters-in-Python/.

[8] ANDERSON, Brian D. O.; MOORE, John B. (John Barratt). Optimal filtering. 2005, p. 357. ISBN 978-0486439389.

[9] SIMON, Dan. Optimal state estimation : Kalman, H [infinity] and nonlinear approaches. 2006, p. 526. ISBN 978-0471708582.

[10] SALYCHEV, O. S. (Oleg Stepanovich). *Applied inertial navigation : problems and solutions*. BMSTU Press, 2004. ISBN 5703823951.

[11] SALAU, Nina P.G.; TRIERWEILER, Jorge O.; SECCHI, Argimiro R.; MARQUARDT, Wolfgang. A New Process Noise Covariance Matrix Tuning Algorithm for Kalman Based State Estimators. *IFAC Proceedings Volumes*. 2009, vol. 42, no. 11, pp. 572–577. ISBN 9783902661548. ISSN 1474-6670. Available from DOI: 10.3182/20090712-4-TR-2008.00092.

[12] PONCELA, Marta; PONCELA, Pilar; PERÁN, José Ramón. Automatic tuning of Kalman filters by maximum likelihood methods for wind energy forecasting. *Applied Energy*. 2013, vol. 108, pp. 349–362. ISSN 0306-2619. Available from DOI: 10.1016/J.APENERGY.2013.03.041.

[13] CHEN, Zhaozhong; HECKMAN, Christoffer; JULIER, Simon; AHMED, Nisar. Weak in the NEES?: Auto-Tuning Kalman Filters with Bayesian Optimization. *2018 21st International Conference on Information Fusion, FUSION 2018*. 2018, pp. 1072–1079. ISBN 9780996452762. Available from DOI: 10.23919/ICIF.2018.8454982.

[14] MU, Tingting; NANDI, Asoke K. Automatic tuning of L2-SVM parameters employing the extended Kalman filter. *Expert Systems*. 2009, vol. 26, no. 2, pp. 160–175. ISSN 1468-0394. Available from DOI: 10.1111/J.1468-0394.2009.00469.X.

[15] CAI, Levi; BOYACIOGLU, Burak; WEBSTER, Sarah E.; VAN UFFELEN, Lora; MORGANSEN, Kristi. Towards auto-tuning of kalman filters for underwater gliders based on consistency metrics. *OCEANS 2019 MTS/IEEE Seattle, OCEANS 2019*. 2019. ISBN 9780578576183. Available from DOI: 10.23919/OCEANS40490.2019.8962573.

[16] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. Probabilistic robotics. [N.d.], p. 647. ISBN 978-0262201629.

[17] JULIER, Simon J; UHLMANN, Jeffrey K. A New Extension of the Kalman Filter to Nonlinear Systems. [N.d.].

[18] CHUI, C. K.; CHEN, G. (Guanrong). Kalman filtering : with real-time applications. 2009, p. 229. ISBN 3642099661.

[19] CAMPESTRINI, Christian; HEIL, Thomas; KOSCH, Stephan; JOSSEN, Andreas. A comparative study and review of different Kalman filters by applying an enhanced validation method. *Journal of Energy Storage*. 2016, vol. 8, pp. 142–159. ISSN 2352-152X. Available from DOI: 10.1016/J.EST.2016.10.004.

[20] PEI, Yan; BISWAS, Swarnendu; FUSSELL, Donald S.; PINGALI, Keshav. An Elementary Introduction to Kalman Filtering. *Communications of the ACM*. 2017, vol. 62, no. 11, pp. 122–133. ISSN 15577317. Available from DOI: 10.48550/arxiv.1710.04055.

[21] ASWAL, Neha; BHATTACHARYA, Baidurya; SEN, Subhamoy. Joint and Dual Estimation of States and Parameters with Extended and Unscented Kalman Filters. *Recent Developments in Structural Health Monitoring and Assessment — Opportunities and Challenges*. 2022, pp. 223–252. Available from DOI: 10.1142/9789811243011{_}0008.

[22] YE, Min; GUO, Hui; XIONG, Rui; YU, Quanqing. A double-scale and adaptive particle filter-based online parameter and state of charge estimation method for lithium-ion batteries. *Energy*. 2018, vol. 144, pp. 789–799. ISSN 0360-5442. Available from DOI: 10.1016/J.ENERGY.2017.12.061.

[23] MAHDIANFAR, Hessam; PAVLOV, Alexey; AAMO, Ole Morten. Joint unscented Kalman filter for state and parameter estimation in Managed Pressure Drilling. *2013 European Control Conference, ECC 2013*. 2013, pp. 1645–1650. ISBN 9783033039629. Available from DOI: 10.23919/ECC.2013.6669753.

[24] STROUD, Jonathan R.; KATZFUSS, Matthias; WIKLE, Christopher K. A Bayesian Adaptive Ensemble Kalman Filter for Sequential State and Parameter Estimation. *Monthly Weather Review*. 2018, vol. 146, no. 1, pp. 373–386. ISSN 1520-0493. Available from DOI: 10.1175/MWR-D-16-0427.1.

[25] YU, Quanqing; XIONG, Rui; LIN, Cheng; SHEN, Weixiang; DENG, Junjun. Lithium-Ion Battery Parameters and State-of-Charge Joint Estimation Based on H-Infinity and Unscented Kalman Filters. *IEEE Transactions on Vehicular Technology*. 2017, vol. 66, no. 10, pp. 8693–8701. ISSN 00189545. Available from DOI: 10.1109/TVT.2017.2709326.

[26] LJUNG, Lennart. System identification : theory for the user. 1999, p. 609. ISBN 978-0136566953.

[27] DE MAESSCHALCK, R.; JOUAN-RIMBAUD, D.; MASSART, D. L. The Mahalanobis distance. *Chemometrics and Intelligent Laboratory Systems*. 2000, vol. 50, no. 1, pp. 1–18. ISSN 0169-7439. Available from DOI: 10.1016/S0169-7439(99)00047-7.

[28] BRABLC, Martin. *Control Of Nonlinear Systems Using Local Approximation Methods*. Brno, 2016. PhD thesis. Brno University of Technology.

[29] SCHAAL, Stefan; ATKESON, Christopher G. Constructive Incremental Learning from Only Local Information. *Neural Comput.* 1998, vol. 10, no. 8, pp. 2047–2084. ISSN 0899-7667. Available from DOI: 10.1162/089976698300016963.

[30] VIJAYAKUMAR, S; D'SOUZA, A; SHIBATA, T; CONRADT, J; SCHAAL, Stefan. Statistical learning for humanoid robots. *AUTONOMOUS ROBOTS*. 2002, vol. 12, no. 1, pp. 55–69. Available from DOI: 10.1023/A:1013258808932.

[31] ATKESON, Christopher G.; MOORE, AW; SCHAAL, Stefan. Locally weighted learning. *ARTIFICIAL INTELLIGENCE REVIEW*. 1997, vol. 11, no. 1-5, pp. 11–73. Available from DOI: 10.1023/A:1006559212014.

[32] ATKESON, Christopher G.; MOORE, AW W; SCHAAL, Stefan. Locally weighted learning for control. *ARTIFICIAL INTELLIGENCE REVIEW*. 1997, vol. 11, no. 1-5, pp. 75–113. Available from DOI: 10.1023/A:1006511328852.

[33] SCHAAL, Stefan; ATKESON, Christopher G.; VIJAYAKUMAR, Sethu. Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning. *Applied Intelligence 2002 17:1*. 2002, vol. 17, no. 1, pp. 49–60. ISSN 1573-7497. Available from DOI: 10.1023/A:1015727715131.

[34] WILLIAMS, Grady; GOLDFAIN, Brian; REHG, James M.; THEODOROU, Evangelos A. Locally Weighted Regression Pseudo-Rehearsal for Online Learning of Vehicle Dynamics. 2019. Available from DOI: 10.48550/arxiv.1905.05162.

[35] PARSA, Behnoosh; RAJASEKARAN, Keshav; MEIER, Franziska; ASHIS, ·; BANERJEE, G; PARSA, B; RAJASEKARAN, K; MEIER, F; BANERJEE, A G. A Hierarchical Bayesian Linear Regression Model with Local Features for Stochastic Dynamics Approximation. 2018. Available from DOI: 10.48550/arxiv.1807.03931.

[36] LEE, Dong Hyun; LEE, Ju Jang. Incremental receptive field weighted actor-critic. *IEEE Transactions on Industrial Informatics*. 2013, vol. 9, no. 1, pp. 62–71. ISSN 15513203. Available from DOI: 10.1109/TII.2012.2209660.

[37] ZHANG, Lei; ZHANG, Pingwen; ZHENG, Xiangcheng. Error estimates for Euler discretization of high-index saddle dynamics. 2021. Available from DOI: 10.48550/arxiv.2111.05156.

[38] R. ANANTHASAYANAM, Mudambi. Tuning of the Kalman Filter Using Constant Gains. *Introduction and Implementations of the Kalman Filter*. 2019. Available from DOI: 10.5772/INTECHOPEN.81795.

[39] HOUTEKAMER, Peter L.; MITCHELL, Herschel L.; DENG, Xingxiu. Model Error Representation in an Operational Ensemble Kalman Filter. *Monthly Weather Review*. 2009, vol. 137, no. 7, pp. 2126–2143. ISSN 1520-0493. Available from DOI: 10.1175/2008MWR2737.1.

[40] DRÉCOURT, Jean Philippe; MADSEN, Henrik; ROSBJERG, Dan. Bias aware Kalman filters: Comparison and improvements. *Advances in Water Resources*. 2006, vol. 29, no. 5, pp. 707–718. ISSN 0309-1708. Available from DOI: 10.1016/J.ADVWATRES.2005.07.006.

[41] FITZGERALD, Robert J. Divergence of the Kalman Filter. *IEEE Transactions on Automatic Control*. 1971, vol. 16, no. 6, pp. 736–747. ISSN 15582523. Available from DOI: 10.1109/TAC.1971.1099836.

[42] XIE, Lihua; DE SOUZA, Carlos E. Robust Kalman Filtering for Uncertain Discrete-Time Systems. *IEEE Transactions on Automatic Control*. 1994, vol. 39, no. 6, pp. 1310–1314. ISSN 15582523. Available from DOI: 10.1109/9.293203.

[43] FARAWAY, J.J. *Practical Regression and Anova Using R*. University of Bath, 2002. Available also from: https://cran.r-project.org/doc/contrib/Faraway-PRA.pdf.

[44] HAYASHI, Fumio. Econometrics. 2000, p. 683. ISBN 978-0691010182.

[45] VAN DE GEER, Sara A; EVERITT, Brian S; HOWELL, David C. Least Squares Estimation. *Encyclopedia of Statistics in Behavioral Science*. 2005, vol. 2, pp. 1041–1045. ISBN 978-0-470-86080-9. Available also from: https://stat.ethz.ch/~geer/bsa199_o.pdf.

[46] VIJAYAKUMAR, Sethu. *Software (Sethu Vijayakumar)*. [N.d.]. Available also from: https://homepages.inf.ed.ac.uk/svijayak/software/.

[47] BRABLC, Martin; ZEGKLITZ, Jaň; GREPL, Robert; BABUŠKA, Robert. Control of Magnetic Manipulator Using Reinforcement Learning Based on Incrementally Adapted Local Linear Models. 2021. Available from DOI: 10.1155/2021/6617309.

[48] DAMSTEEG, Jan Willem; NAGESHRAO, Subramanya P.; BABUŠKA, Robert. Model-based real-time control of a magnetic manipulator system. *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*. 2018, vol. 2018-January, pp. 3277–3282. ISBN 9781509028733. Available from DOI: 10.1109/CDC.2017.8264140.

[49] *MF634 — Humusoft*. [N.d.]. Available also from: https://www.humusoft.cz/datacq/mf634/.

[50] SOVA, Václav. *Adaptivní řízení elektromechanických aktuátorů s využitím dopředného kompenzátoru založeného na více-modelovém přístupu*. Brno, 2018. PhD thesis. Brno University of Technology.

[51] NAJMAN, J.; BRABLC, M.; RAJCHL, M.; BASTL, M.; SPÁČIL, T.; APPEL, M. *Advances in Intelligent Systems and Computing*. Vol. 1044, Monte carlo based detection of parameter correlation in simulation models. 2020. ISBN 9783030299927. ISSN 21945365. Available from DOI: 10.1007/978-3-030-29993-4{_}7.

Author's publications

[1] BRABLC, M; SOVA, V; GREPL, R. *Adaptive feedforward controller for a DC motor drive based on inverse dynamic model with recursive least squares parameter estimation*. Ed. by MAGA, Dusan; BREZINA, Tomas. 2017. ISBN 978-8-0010-5883-1. Available also from: `http://www.scopus.com/inward/record.url?eid=2-s2.0-85015277645&partnerID=MN8TOARS`.

[2] RUBES, Ondrej; SMILEK, Jan; BRABLC, Martin; HADAS, Zdenek. Nonlinear redesign of vibration energy harvester: Linear operation test and nonlinear simulation of extended bandwidth. In: IEEE, 2016, pp. 737–742. ISBN 978-1-5090-1798-0. Available from DOI: `10.1109/EPEPEMC.2016.7752086`.

[3] MUSIL, Filip; BRABLC, Martin. *Modelling and Simulation of Vehicle Boot Door*. 2018. Available from DOI: `10.1007/978-3-319-65960-2_50`.

[4] RUBES, Ondrej; BRABLC, Martin; HADAS, Zdenek. Nonlinear vibration energy harvester: Design and oscillating stability analyses. *Mechanical Systems and Signal Processing*. 2018. ISSN 08883270. Available from DOI: `10.1016/j.ymssp.2018.07.016`.

[5] RUBES, Ondrej; BRABLC, Martin; HADAS, Zdenek. Verified nonlinear model of piezoelectric energy harvester. *MATEC Web of Conferences*. 2018, vol. 211. ISSN 2261236X. Available from DOI: `10.1051/MATECCONF/201821105005`.

[6] SOVA, V.; BRABLC, M.; GREPL, R. FPGA Implementation of Multiplierless Low-Pass FIR Differentiator. In: 2019. ISBN 9788021455443.

[7] RAJCHL, M.; BRABLC, M. Inverse Model Approximation Using Iterative Method and Neural Networks with Practical Application for Unstable Nonlinear System Control. In: 2019. ISBN 9788021455443.

[8] NAJMAN, J.; BRABLC, M.; RAJCHL, M.; BASTL, M.; SPÁČIL, T.; APPEL, M. *Monte carlo based detection of parameter correlation in simulation models*. 2020. ISBN 9783030299927. ISSN 21945365. Available from DOI: `10.1007/978-3-030-29993-4_7`.

[9] MATĚJÁSKO, Michal; BRABLC, Martin; APPEL, Martin; GREPL, Robert. Contactless fault detection of a dc motor direction of rotation using its stray magnetic field. *Machines*. 2021, vol. 9. ISSN 20751702. Available from DOI: `10.3390/MACHINES9110281`.

[10] STODOLA, Marek; RAJCHL, Matej; BRABLC, Martin; FROLÍK, Stanislav; KŘIVÁNEK, Václav. Maxwell points of dynamical control systems based on vertical rolling disc—numerical solutions. *Robotics*. 2021, vol. 10. ISSN 22186581. Available from DOI: `10.3390/ROBOTICS10030088`.

[11] BRABLC, Martin; ŽEGKLITZ, Jan; GREPL, Robert; BABUŠKA, Robert. Control of Magnetic Manipulator Using Reinforcement Learning Based on Incrementally Adapted Local Linear Models. *Complexity*. 2021, vol. 2021. ISSN 10990526. Available from DOI: `10.1155/2021/6617309`.

Publication type	**Count**	**Citations**
Scientific journal article Q1 (First quartil)	1	20
Scientific journal article Q2 (Second quartil)	3	2
Conference paper	7	16

Author's publication summary (SCOPUS database)