Nonlinear System Identification and Control for Autonomous Robots

Laurens Bliek

Master of Science Thesis

TU Delft Delft University of Technology

Delft Institute of Applied Mathematics (DIAM)

Nonlinear System Identification and Control for Autonomous Robots

Master of Science Thesis

For the degree of Master of Science in Applied Mathematics at Delft University of Technology

Laurens Bliek

November 7, 2013

Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS) · Delft University of Technology

The work in this thesis was supported by Almende B.V. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Institute of Applied Mathematics (DIAM)

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS) for acceptance a thesis entitled

Nonlinear System Identification and Control for Autonomous Robots

by

Laurens Bliek

in partial fulfillment of the requirements for the degree of

Master of Science Applied Mathematics

Dated: November 7, 2013

Chair: _____
Dr. J.W. van der Woude

Supervisor at university: _____
Dr. ing. D. Jeltsema, M.Sc.

Supervisor at company: _____
Ir. A.C. van Rossum

External expert: _____
Prof. dr. ir. M. Verhaegen

Abstract

Most robots receive information about their own dynamics or about their environment before starting to work on their tasks. This makes them less fit for operating in unknown environments without human supervision. In this thesis, the goal is to let a robot learn the relation between its actions and perceptions autonomously. This problem is divided in a prediction part, and a control part. The goal of the prediction process is to correctly predict the sensor data of the future, given the past sensor and actuator data. The goal of the controller is to provide exploratory behaviour, so that different situations can be learned by the predictor. Of the discussed techniques, one predictor and one controller are chosen to be used in an experiment: a mobile robot equipped with a camera that has to learn what happens when rotating inside a box with pictures. The chosen predictor is a new type of local linear model, the Shuffled Linear Model (SLM). In this model, the localities are chosen randomly and linear regression is used to determine the linear model parameters. It is proved that this model is a universal approximator, and an upper bound on the number of models is given. The chosen controller makes use of intrinsic rewards to aid the predictor by providing data that is both new and easy to predict.

Table of Contents

List of Figures

List of Tables

Preface

This master thesis is the final assignment for my degree of Master of Science in Applied Mathematics at the faculty of Electrical Engineering, Mathematics and Computer Science at Delft University of Technology. To get a balance between theoretical research and a practical assignment, I made the decision to do this project at a research company: Almende B.V. in Rotterdam. Although Almende's main focus is the concept of self-organisation in hybrid actor networks, this project involved a single-agent setting: letting an agent learn the relation between sensor and actuator data autonomously. The final result is an agent that can learn from its actions and sensor data to notice changes in the environment and distinguishing its own actions from external actions.

The thesis committee consists of ir. A.C. van Rossum as my daily supervisor at Almende B.V., dr. ing. D. Jeltsema, M.Sc. as my supervisor at Delft University of Technology, chair of the committee dr. J.W. van der Woude and external expert prof. dr. ir. M. Verhaegen.

Acknowledgements

First of all, I would like to thank my daily supervisor Anne van Rossum for his supervision during this research and for his idea for the research topic. Who knows what this thesis would be about if it were not for his ideas? I would also like to thank my other colleagues at Almende for providing such a great and inspiring work environment. Special thanks go to Dominik Egger for his work on the Romo robot. Most of the experiments in this thesis were possible mainly because of his work. Special thanks also go to Giovanni Pazienza for his advice on writing research papers, Remco Tukker for his advice and ideas on this research and Marc Hulscher for his 3D-printing skills.

Besides Almende B.V. I have had much help from several people at Delft University of Technology, especially my supervisor Dimitri Jeltsema. His supervision was valuable for my research, as well as the support of Jacob van der Woude and the advice of Michel Verhaegen, who are all part of my thesis committee. Of course I am also grateful for the university as a whole during the part of my study that led to this research.

I would also like to thank all my friends and relatives who supported me during this research and my whole academic career, both in the Netherlands and abroad. Unfortunately I cannot mention all of them by name, but I love all of them and appreciate their support. Last but not least, I would like to thank my loving God for His guidance and protection and for giving me the talents and opportunities that all led to this research.

Rotterdam — Laurens Bliek
November 7, 2013

Chapter 1

Introduction

Although our western society has been using machines on a large scale for almost two centuries, up until now most of them have been completely designed or programmed in advance to perform a specific task in a specific environment. This is one of the reasons we do not see many robots walking, riding or flying around in the streets: the real world, and human society in particular, is so complex that it is impossible to preprogram a robot that can handle every possible situation in that environment. However, we could benefit a lot from robots in everyday life, for example in performing dangerous tasks, repetitive tasks, tasks that require high precision, speed or computational power, or operate in situations that are unfit for humans. Many of these tasks are already being performed by robots, but only few of them occur in complex environments, and if they do, it is mostly with human supervision.

To interact with the physical world, any biological or artificial agent (like a robot) needs both sensors and actuators. If the agent also contains some cognitive processing abilities, there is potential for artificial intelligence. Although there is no clear definition of artificial intelligence, one important aspect of it is learning and reasoning about the world's structures. This type of knowledge is required to design a system with a desired behaviour, which is the main focus of *control theory*. Typically, this knowledge is represented in a mathematical model by using the physical knowledge scientists have acquired. But sometimes this knowledge is not available, and constructing a suitable mathematical model for the system becomes a subgoal in itself. The main area that focuses on constructing such a model is the area of *system identification*.

This thesis reverses the priorities of system identification and control in a certain sense. Usually, the main purpose of system identification is to control a system more accurately. Yet in this thesis, system identification is seen as the main goal, while control can be used to aid in this process. Another way to see this, is that the desired task of the agent to be considered is to identify the relation between actuator and sensor data, which will be called the system in this thesis. What is interesting about this approach is that this involves more than just extracting information from a stream of data: since the system is physical, future data will depend on past data, adding the aspect of time and moving the problem towards the field of

systems theory. Furthermore, the actuator data can be actively chosen by the agent itself, which adds a whole new dimension to the problem.

The relation between actuator and sensor data is the most basic structure available to the agent. Only when these structures are learned, there will be a potential to develop higher levels of cognition (e.g., emotion, consciousness, logical reasoning) and more complex skills (e.g., avoiding, using or tracking objects). These are beyond the scope of this thesis.

1-1 Research questions

The main goal of this thesis is to answer the following question:

Main research question
How can an autonomous physical agent learn the relation between its actuator and sensor data?

Although there are many possible routes to the answer of this question, the focus of this thesis will be within the framework of systems theory. Representing an agent as a system with inputs, outputs and hidden states opens up a box full of mathematical techniques that have proofs and descriptions of when they work, how they work, and of how well they work in terms of robustness, computation time, underlying assumptions, etc. An alternative would be to stay in the field of *machine learning*, where learning structures is actually the main goal. Although many algorithms and results from this field will appear in this thesis, they will be presented as much as possible within a system-theoretic notation, where objects are clearly defined and results are rigorously proved. Another alternative would be to take a *Bayesian* approach or use *information theory*. These approaches can have many benefits in the problem of this thesis, and there are several examples of where these are combined with systems theory, for example in stochastic control or chaos theory. However, this is beyond the scope of this thesis.

Using scientific results from various fields, one purpose of this thesis is to combine them in a system-theoretical framework. Therefore, the following subquestions will also be answered within this framework:

Subquestions

- How can the relation between actuator and sensor data be represented as a dynamical system?
- How can the right functions and parameters of such a system be found? Can this learning process also be described as a dynamical system?
- How can control be used to aid in the learning process?
- How can we test whether an agent has learnt the relation between actuator and sensor data?

These questions are given more explanation in Chapter 2.

1-2 Type of agents considered

Although the experiments in this this concern one specific agent, the theoretical results should apply to agents with the following properties:

1. The agents have sensors and actuators to interact with the physical world, and processing power to handle the data (not necessarily on the agent itself);
2. The agents are autonomous, meaning that they can operate without human supervision;
3. There is a relation between the actuators and the sensors;
4. The underlying dynamics of the agents are not necessarily linear and the agent is not necessarily holonomic[1];
5. The sensor data can be high-dimensional;
6. The agents cannot change anything about the environment, and the environment is assumed static and deterministic.

The first property binds the agent to the real world, where the laws of physics apply and specifically causality applies: the past does not depend on the future. Furthermore, the physical world, unlike some robot simulations for example, does not consist of only polygons and other 'nice' objects. This makes the system more complex, but also contains advantages. For example, two different locations in the world cannot be exactly equal.

The second property implies that the agent can only use internal information: the sensor and actuator data, and functions of this data (since the agent has access to processing power). There are no external signals like a goal or a target signal.

The third property implies that the control will have effect on the sensor data in some way. Maybe the effect is nonlinear, depends on an internal state or on past data, or the effect is delayed, but there is an effect.

The fourth property enforces the scope of this thesis to extend to nonlinear systems and prevents the use of linear models. Most parts of dynamical systems theory focus on linear systems. Nonholonomic agents pose difficulties in the control of an agent, but since this thesis does not focus on moving a system to a desired state, these problems are less relevant.

The fifth property gives rise to a new problem: the 'curse of dimensionality'. Many algorithms that appear through-out this thesis only work in practice for low-dimensional data, else the required computation time and number of data samples become too large. Solutions to this problem within the context of this thesis will be presented. Note that the high-dimensionality applies to the sensor data, not the actuator data.

Besides the problem of computation time, the high-dimensionality of the sensor data also causes the problem to differ from classical system-theoretic problems like observability and state estimation. Whereas observability is related to the problem of finding the hidden states of a system with only a limited system output, a system with high-dimensional data has the

[1] In a holonomic agent, the degrees of freedom and the number of actuators are equal.

problem that there is a lot of redundant information in it, making the problem more related to data mining.

The final property limits the problem to environments that do not change, making it easier to learn the relation between actuator and sensor data. If an agent performs the same actions in the exact same situation as before, the next situation will be exactly the same as before. The agent can move around in the world and observe the world, but it cannot change anything about it. Section 3-1 does shortly mention a possibility for dealing with non-static environments, but this will not be further discussed in this thesis. The deterministic part shifts the focus of this thesis away from probabilistic methods, to further narrow down the problem.

1-3 Outline of this thesis

This thesis is organised in the following way: in Chapter 2, the problem will be described and it will be explained how the main theory is divided into two parts, namely identification and control of the system. The thesis also contains a practical problem which is described in this chapter. Chapter 3 deals with techniques to describe the system and find its functions and parameters, when the control is assumed to be known (open-loop system). Chapter 4 contains the main contribution of this thesis: the proposed Shuffled Linear Model (SLM), used for the system identification part of this thesis. Approximation capabilities for this model are analysed and proven, and an upper bound on the required number of local models is given. Chapter 5 focuses on what control techniques to use, assuming the agent has some way to learn the relation between actuators and sensors. Note that these techniques are different from most traditional control techniques, where the goal is to move a system towards a desired state. Instead, the goal of the control in this case is to aid the system identification techniques as well as possible, without using any external signals, since only autonomous agents are considered. Chapter 6 contains the details and results of the practical part of this thesis. The thesis is concluded in Chapter 7, which discusses the main contribution of this thesis, the main results of the experiments, and suggestions for future work.

Chapter 2

Problem description

2-1 Theory

The problem in this thesis has a theoretical and a practical aspect. Both are concerned with the relation between actuator and sensor data. Learning the relation between actuator and sensor data can be a complex task, because it can usually not be described by a simple linear function. Dependence on the past makes it natural to represent the relation as a dynamical system. Unfortunately, concepts like nonlinearity, noise, and high-dimensionality make the problem too complex to be solved by classical system-theoretic techniques. Therefore, it is necessary to look at other fields, like machine learning and statistics. Still, the results in this thesis are represented within the framework of systems theory.

The first part of the problem is to describe the problem mathematically. In the next chapter, an optimisation problem is formulated, where most symbols are functions or variables in a dynamical system. This answers the first "subquestion" of the previous chapter.

The next part of the problem is to find those functions and variables that correctly describe the system. Techniques from the area of system identification are examined, and where needed, machine learning techniques are examined as well. This is also done in the next chapter.

The use of control in this thesis is different than in most system-theoretic literature. Controlling the system is not the goal here, rather control is seen as an aid for the system identification process. It is possible to look at the closed-loop system, where the control depends directly on the sensor data. Rather than diving into the theory of closed-loop systems, in this thesis the control and the system identification process are split, which causes the main theory of this thesis to be divided into two parts: system identification (Chapter 3), and control for system identification (Chapter 5).

In the first part, it is assumed that there is some controller that decides what the actuators should do. This actuator data is known, but is seen as an external signal: only the dependence from sensor data on actuator data is taken into account. This makes the system an open-loop system. In the second part of the theory of this thesis, it is assumed that there is a way to

learn the relation between actuator and sensor data, for a given control. The problem is then to choose a control that can aid in this learning process.

One reason for splitting the problem in identification and control is the possibility to use system identification techniques where the actuator data can be seen as an external signal. And the control techniques do not need to worry about unknown functions or parameters of the system and can work with an approximation of the real system. This makes sure there are no restrictions on the way the system is identified or controlled.

A second reason to split the problem is that many biological and mechanical systems are assumed to have separate systems for the prediction of state or sensor data, and the choice of motor commands. A major advantage of this is that the effect of the motor command can be simulated by sending a copy of the command to the predictor or forward model, which helps solving many problems like coping with delay in the sensorimotor loop, anticipating sensory effects, distinguishing between self-generated and non-self-generated effects, finding the error to facilitate motor learning, and simulation and planning [1].

2-2 Practice

The final part of the problem is the implementation of the theory in a real-life application. A mobile nonholonomic robot with visual sensors and differential drive dynamics communicates with a computer to send sensory signals and retrieve control signals. Visual sensors have the advantage that there is a direct effect of the robot's movement on the sensors, even though this effect can be very complex in our three-dimensional world. Another advantage is that it is possible to distinguish between different states of the robot using the visual sensors, since in general two different locations in the real world are not visually identical, provided the sensors are accurate enough.

The differential drive dynamics cause the robot to be nonholonomic. The dynamics can be realised by a state-space description that is nonlinear but affine in the control. These restrictions might cause linear system identification techniques to fail, but they do provide structure to the system, besides the structure of the environment: if the agent can really learn the relation between actuator and sensor data, it should be able to learn this control-affine relation, even with less actuators than the degrees of freedom (DoF) and with nonlinear dynamics.

Although in this thesis the dynamics and sensors of the robot are assumed unknown, and the robot has to find out the available structures by itself, this practical instance is kept in mind through-out this thesis: learning the relation between actuator and sensor data should be possible for a robot with differential drive dynamics and visual sensors.

The theories from this thesis are implemented on the computer, not on the robot itself. When it seems the relation between actuator and sensor data has been learnt, this is tested by means of some experiment that checks whether the difference between the robot's own actions and someone else's actions can be found. The answer to the last "subquestion" from the previous question should be answered before performing this experiment.

Chapter 3

System Identification

Finding structure and rules in nature and describing them is one of the main purposes of science. For some applications, including most robotic applications, these rules can be described by a dynamical system. When the input and output data of such a system is known, but the underlying rules of the system are not known, it becomes important to use techniques that can find these rules and structures. These techniques can be found in the area of system identification.

While the motivation for using system identification techniques can vary (typically either prediction or control of a system), the goal of the techniques themselves remain the same: construct a model for a dynamical system. For the problem in this thesis, if the relation between actuator and sensor data can be represented as a dynamical system, using system identification techniques could give the solution to the problem.

Although the area of system identification started to focus mainly on linear systems, over the years the attention started to shift to nonlinear systems. On the one hand, nonlinear models like neural networks [2] have been used to approximate dynamical systems, and can actually have good approximation capabilities for many dynamical systems [3]. However, analysis of these models can be hard. Other types of nonlinear models that are more related to linear systems are for example the bilinear [4] and block-oriented models [5]. Besides these state-space models, using input-output models is a more common approach to nonlinear system identification [6]. Although state-space and input-output models are related, they both have their own advantages and disadvantages.

Besides the type of model used, there have been two main approaches within the area of system identification to actually use these models: the Subspace Identification Method (SIM) [7] and the Prediction Error Method (PEM) [8, 9]. While these approaches have their own advantages and disadvantages, the PEM approach is used in this thesis, mainly because of the large number of models and techniques that fall under this category. The techniques used in this thesis should be as general as possible, so that they can be used with as little knowledge about the actuator and sensor data as possible. This is also termed *black-box modelling*.

An overview of techniques that can model nonlinear systems without any prior physical insight can be found in [6]. This is also the main basis for this chapter. An overview of

several communities and open problems within the general field of system identification is given in [10].

This chapter is organised as follows: first, the PEM is explained in Section 3-1. In Section 3-2, a distinction is made between possible choices for the regression vector, since it is not always necessary or practical to use all data. The topic of Section 3-3 is the predictor function: what are possible choices for the predictor function, and how can this function be parametrised? Section 3-4 discusses several methods for finding the optimal parameters of the predictor function. This chapter is concluded with a section about regularisation in Section 3-5.

3-1 Prediction error method

Suppose we receive input-output pairs (u_t, y_t) every timestep $t = 0, 1, 2, \ldots$, with actuator data $u \in \mathbb{R}^m$ and sensor data $y \in \mathbb{R}^n$. The input u is treated as an external signal, even though it could be the result of a feedback rule for example. We want to find a system that describes the relation between actuator and sensor data of an agent, without any prior knowledge. One way to do this is by trying to predict the corresponding output y_t at each timestep t. This can be done by using the information up to timestep $t-1$ to construct a predictor (or estimator, or observer) $\hat{y}_t = f(\varphi_{t-1})$. Here, $\varphi \in \mathbb{R}^l$ is called the regression vector, which can contain all input and output data up to timestep $t-1$, and f is called the predictor function. Examples for φ and f will appear in the following sections. The assumption in this thesis is that the past data in φ_{t-1} is enough to make an accurate model of the system, as long as t is large enough.

In general, $\hat{y}$ will not be a perfect approximation of y. The closer $\hat{y}$ is to y, the better the prediction. At each timestep, we can define the prediction error ϵ as $\epsilon_t = y_t - \hat{y}_t$. This gives the relation $y_t = f(\varphi_{t-1}) + \epsilon_t$. Once the predictor function f has been chosen, which will be the topic of the following sections, this answers the first subquestion of this thesis: **How can the relation between actuator and sensor data be represented as a dynamical system?** See figure 3-1 for an overview of the system.

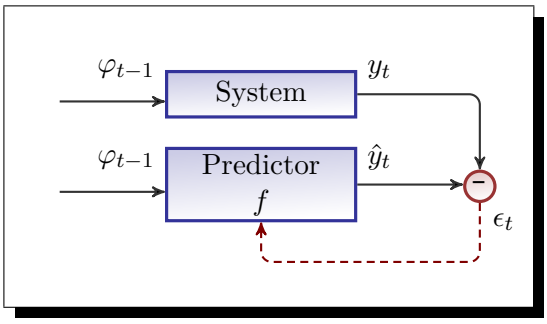


Figure 3-1: Using a predictor to model the system.

Since the prediction error can change at every timestep, it is not a good measure for the quality of the model. Information about the prediction error over all timesteps is needed to see if the model is accurate. For this, let the total error E be a function of the prediction error over all timesteps up to time t: $E_t = \sum_{i=1}^{t} V(\epsilon_i)$, for some function $V : \mathbb{R}^n \to \mathbb{R}$. Examples are the sum of squares with $V(\epsilon) = ||\epsilon||^2$, or the log-likelihood with $V(\epsilon) = -\log p(\epsilon)$ if the probability distribution p of the prediction error is known[1]. It is also possible to multiply V with a weight λ to make recent errors more important: $E_t = \sum_{i=1}^{t} \lambda^{t-i} V(\epsilon_i)$. This could be useful for non-static environments, but it will not be considered further in this thesis.

[1] In this thesis, $||\cdot||$ denotes the Euclidean norm, unless otherwise specified.

To get an accurate model, the total error E needs to be minimised. In the PEM, this is done by parametrising the function f and finding the parameters for which the total error is minimal, since the total error depends on these parameters. Depending on the functions f and V, different optimisation algorithms exist for this problem. Some examples are: gradient descent, linear least-squares, and the maximum likelihood method. Section 3-3 and Section 3-4 discuss some methods to parametrise the function f and to find the optimal parameters, respectively.

Using the parametrised function $f(\varphi_{t-1}, \theta)$, the goal of this chapter becomes to solve the following minimisation problem:

$$\min_{\theta \in \Theta} E_t(\theta), \quad E_t(\theta) = \sum_{i=1}^{t} V(y_i - f(\varphi_{i-1}, \theta)), \tag{3-1}$$

where θ is the vector of parameters of the function f, and Θ is some set of possible values for θ . Let θ^* be the solution to this problem, then the total system can be denoted as:

$$\begin{aligned} y_t &= f(\varphi_{t-1}, \theta_t) + \epsilon_t, \\ \theta_t &= \theta_{t-1} + \Delta\theta, \end{aligned} \tag{3-2}$$

where $\Delta\theta$ changes θ in the direction of the optimal parameters θ^*. Once f, θ and $\Delta\theta$ have been defined, which will be done in the following sections, the system above is expected to change parameters θ in such a way that the prediction error decreases. In this thesis, this is called the learning process. This answers the second part of the subquestion: **How can the right functions and parameters of this system be found? Can this learning process also be described as a dynamical system?**

Finding the right functions and parameters of the system will be the subject of the following sections. An overview of all the following methods, as well as a motivation for the choice of methods for the practical problem of this thesis, can be found in Section 6-3.

3-2 Choosing the regression vector

The predictor $\hat{y}_t = f(\varphi_{t-1})$ depends on the regression vector φ, which consists of past data. In theory, it is possible to use all past sensor and actuator data to make an accurate prediction. In this case, $\varphi_t = [y_{t-1}, y_{t-2}, \ldots, y_0, u_{t-1}, u_{t-2}, \ldots, u_0]$. In practice, however, this is not always desired: the number of parameters can become very high if there is no bound on how far back in the past the data will be used, especially for high-dimensional data. Therefore, the data used in the predictor needs to be restricted.

It is possible to make a distinction between input-output models and state-space models. Although for some systems it is possible to use both models, there are some major differences between the models. The most notable difference is the recursive structure of the state-space model. In the following subsections, first some examples of input-output models will be given, followed by some examples of state-space models. This section concludes with a comparison between input-output and state-space models.

3-2-1 Input-output models

In input-output models, the estimated outputs $\hat{y}_t$ depend on input and output data from the past, within a certain time window $[t-q,\ t]$, for example:

$$\hat{y}_t = f(\varphi_{t-1}) = f(y_{t-1}, \dots y_{t-q}, u_{t-1}, \dots u_{t-q}).$$

Other examples also use functions of past input and output data, like $\hat{y}$ or ϵ, in the regression vector φ. Or they can use only the input data or only the output data. Some examples of input-output models are given in this subsection, see also [6].

NFIR

In the Nonlinear Finite Impulse-Response model (NFIR), the output data depends on the input data only, within a finite time interval:

$$\hat{y}_t = f(u_{t-1}, \dots, u_{t-q}),$$

so the regression vector is $\varphi = [u_{t-1}^T, \dots, u_{t-q}^T]^T$. In the linear case (FIR), this is equal to:

$$\hat{y}_t = \sum_{i=1}^{q} B_i u_{t-i}.$$

Looking at the practical problem of this thesis, this model could be useful for agents for which the sensor data is determined solely by some of the past control commands (for example, a robot that can control its position, not just its velocity or acceleration).

NARX

In the Nonlinear AutoRegressive eXogenous model (NARX), the output data depends on both input and output data of the past, within a finite time interval:

$$\hat{y}_t = f(y_{t-1}, y_{t-2}, \dots y_{t-p}, u_{t-1}, u_{t-2}, \dots u_{t-q}),$$

so the regression vector is $\varphi = [y_{t-1}^T, y_{t-2}^T, \dots y_{t-p}^T, u_{t-1}^T, u_{t-2}^T, \dots, u_{t-q}^T]^T$. In the linear case (ARX), this is equal to:

$$\hat{y}_t = \sum_{i=1}^{p} A_i y_{t-i} + \sum_{i=1}^{q} B_i u_{t-i}$$

.

Many practical applications cannot be described by the NFIR model, for example if the control commands of the whole past are necessary to determine the output of an agent. Instead of using all past control commands, the NARX model could be a good alternative.

NARMAX

In the Nonlinear AutoRegressive Moving Average model with eXogenous input (NARMAX), the output data depends on past input and output data, and on the prediction error ϵ:

$$\hat{y}_t = f(y_{t-1}, y_{t-2}, \dots y_{t-p}, u_{t-1}, u_{t-2}, \dots u_{t-q}, \epsilon_{t-1}, \epsilon_{t-2}, \dots, \epsilon_{t-r}),$$

so the regression vector is $\varphi = [y_{t-1}^T, y_{t-2}^T, \dots y_{t-p}^T, u_{t-1}^T, u_{t-2}^T, \dots u_{t-q}^T, \epsilon_{t-1}^T, \epsilon_{t-2}^T, \dots, \epsilon_{t-r}^T]^T$. In the linear case (ARMAX), this is equal to:

$$\hat{y}_t = \sum_{i=1}^{p} A_i y_{t-i} + \sum_{i=1}^{q} B_i u_{t-i} + \sum_{i=1}^{r} C_i \epsilon_{t-i}$$

.

Including more information should cause the NARMAX model to be more accurate than the NARX model. However, because the error ϵ depends on past model outputs $\hat{y}$, not just past outputs y, this model is recurrent. Though not a state-space model, this recurrent model shares some of the drawbacks of state-space models due to this recursion, which are mentioned in Section 3-2-3.

3-2-2 State-space models

When using state-space models, the predicted outputs $\hat{y}_t$ do not depend on past inputs and outputs directly, but on the current state $x \in \mathbb{R}^k$ of a state-space model. The state depends on the previous input and state. In other words, the state x takes the role of a first-order NARX model ($x_t = g(x_{t-1}, u_{t-1})$), while the model output $\hat{y}$ is a function of the current state ($\hat{y}_t = h(x_t)$) or of the current state and the current input ($\hat{y}_t = h(x_t, u_t)$). This last case is not considered in this thesis[2].

The state-space models in this thesis assume the following form:

$$\begin{aligned} x_{t+1} &= g(x_t, u_t), \\ \hat{y}_t &= h(x_t), \end{aligned} \tag{3-3}$$

with state transition function g and output function h. Using the same regressor notation as earlier in this chapter, this can be written as $\hat{y}_t = f(\varphi_{t-1})$, with the regression vector $\varphi_{t-1} = [x_{t-1}^T, u_{t-1}^T]^T$ and state transition $x_{t+1} = g(x_t, u_t)$. The predictor function f follows directly from the state-space model; since $\hat{y}_t = h(g(x_{t-1}, u_{t-1})) = f(\varphi_{t-1})$, the predictor function f is simply the composition of g and h.

Note that there is a recursion within the regression vector, which is not the case for input-output models. In the input-output models, the regression vectors follow directly from the observed input and output data, while in the state-space model the regression vectors are updated according to the state transition function. This makes state-space models less restricted and can decrease the required number of parameters, but it makes it harder to find the optimal parameters, as will be explained in Section 3-4.

Throughout this section, the state-space notation will be used. Starting with the linear case, we consider several systems of this form.

[2]Letting the model output only depend on the state, not the state and input, makes sure the system is more similar to the basis function expansion used in Section 3-3.

LTI system

A linear time-invariant (LTI) system can be written in state-space form as follows:

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t, \\ \hat{y}_t &= Cx_t. \end{aligned} \tag{3-4}$$

In regressor notation this would be equivalent to $\hat{y}_t = C[A\ B]\varphi_{t-1}$, with $\varphi_{t-1} = [x_{t-1}^T, u_{t-1}^T]^T$.

For LTI systems, there is a direct link between the state-space form and input-output form (see Appendix A). Identification of LTI systems is thoroughly discussed in [8].

Many dynamical systems can be described by a LTI system. However, LTI systems have their limitations, and nonlinear systems are becoming increasingly popular. The focus in this thesis is mainly on nonlinear systems.

Bilinear system

A bilinear system does not only have a linear combination of the state and the control input but also of the Kronecker product of the state and the control input:

$$\begin{aligned} x_{t+1} &= Ax_t + F(u_t \otimes x_t) + Bu_t, \\ y_t &= Cx_t, \end{aligned} \tag{3-5}$$

where the Kronecker product of two vectors a, b is defined as

$$a \otimes b = [a_1b_1, \ldots, a_1b_n, a_2b_1, \ldots, a_2b_n, \ldots, a_nb_1, \ldots, a_nb_m]^T.$$

In regressor notation this system would be equivalent to

$$\hat{y}_t = f(\varphi_{t-1}) = C([A\ B]\varphi_{t-1} + M(\varphi_{t-1} \otimes \varphi_{t-1})),$$

with $\varphi_{t-1} = [x_{t-1}^T, u_{t-1}^T]^T$ and M such that $M(\varphi_{t-1} \otimes \varphi_{t-1}) = F(u_t \otimes x_t)$. Many sensor dynamics can be represented by a bilinear system [11]. See [12] for system identification techniques for bilinear systems.

Block-oriented models

In block-oriented models, the systems are composed of linear dynamic systems (denoted as rectangles in Figure 3-2) and nonlinear static transformations (denoted as circles). The motivation for this is that linear dynamical systems are well-understood, and combining them with nonlinearities can overcome the limitations of LTI systems. Physical insight can be used in the choice of the block structure.

The system of a Wiener model [13] can be written as:

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t, \\ y_t &= h(x_t). \end{aligned} \tag{3-6}$$

In regressor notation this would be equivalent to $\hat{y}_t = f(\varphi_{t-1}) = h([A\ B]\varphi_{t-1})$, with $\varphi_{t-1} = [x_{t-1}^T, u_{t-1}^T]^T$.

The system of a Hammerstein model [14] can be written as:

$$\begin{aligned} x_{t+1} &= Ax_t + Bp(u_t), \\ y_t &= Cx_t. \end{aligned} \tag{3-7}$$

In regressor notation this would be equivalent to $\hat{y}_t = f(\varphi_{t-1}) = C[A\ B]\varphi_{t-1}$, with $\varphi_{t-1} = [x_{t-1}^T, p(u_{t-1}^T)]^T$.

The system of a Hammerstein-Wiener model [15] can be written as:

$$\begin{aligned} x_{t+1} &= Ax_t + Bp(u_t), \\ y_t &= h(x_t). \end{aligned} \tag{3-8}$$

In regressor notation this would be equivalent to $\hat{y}_t = f(\varphi_{t-1}) = h([A\ B]\varphi_{t-1})$, with $\varphi_{t-1} = [x_{t-1}^T, p(u_{t-1}^T)]^T$.

A vast number of algorithms for the identification of these block-oriented models is available [15].

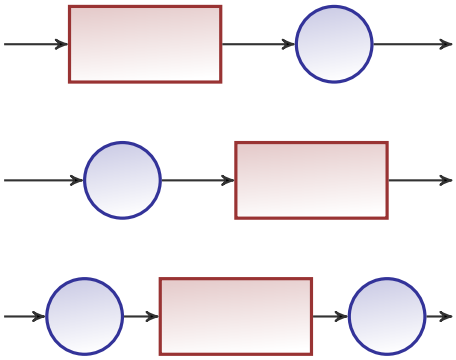


Figure 3-2: Structure of block-oriented models. From top to bottom: a Wiener model, a Hammerstein model, and a Hammerstein-Wiener model.

Recurrent Neural Network (RNN)

Unlike the block-oriented models shown above, a RNN contains a nonlinear dynamical system, followed by a static linearity. Where block-oriented models are used to decompose a complex system in simpler subsystems, a RNN is still a very complex system that is hard to analyse. The main advantage of a RNN is different: theory from neural networks in general and inspiration from biology can be used to find (efficient) learning algorithms.

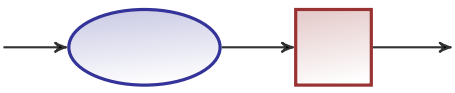


Figure 3-3: Structure of a recurrent neural network.

Using the block notation from above, a RNN could be represented by the block structure in Figure 3-3, where an ellipse represents a nonlinear dynamical system and a square a static linear transformation. The system of a RNN looks as follows:

$$\begin{aligned} x_{t+1} &= g(Ax_t + Bu_t), \\ y_t &= Cx_t. \end{aligned} \tag{3-9}$$

In regressor notation this would be equivalent to $\hat{y}_t = f(\varphi_{t-1}) = Cg([A\ B]\varphi_{t-1})$, with $\varphi_{t-1} = [x_{t-1}^T, u_{t-1}^T]^T$. Here, g is typically a sigmoid function like $g(x) = \tanh(x)$ or $g(x) = (1+e^{-x})^{-1}$, see also Section 3-3-1. Instead of sigmoid functions, one could also use binary functions, radial basis functions, fuzzy functions or wavelets, but the term $Ax_t + Bu_t$ should then be changed as well. Some overviews of algorithms for training RNNs can be found in [16, 17].

Unified notation

The above models can be used to simplify or approximate an unknown nonlinear system

$$\begin{aligned} x_{t+1} &= g(x_t, u_t), \\ \hat{y}_t &= h(x_t). \end{aligned} \tag{3-10}$$

Using the functions g, h and p, one unified notation can be used for all of the above mentioned state-space models:

$$\begin{aligned} \varphi_{t-1} &= [x_{t-1}^T, p(u_{t-1})^T]^T, \\ x_{t+1} &= g(x_t, p(u_t)), \\ f(\varphi_{t-1}) &= h(g(x_{t-1}, p(u_{t-1}))). \end{aligned} \tag{3-11}$$

Tabel 3-1 shows the models corresponding to different choices for g, h and p.

LTI	g, h linear, $p = I$
Bilinear	h linear, $p = I$
Wiener	g linear, $p = I$
Hammerstein	g, h linear
Hammerstein-Wiener	g linear
RNN	h linear, $p = I$

Table 3-1: Properties of the different state-space models in the unified notation. Here, I is the identity matrix, and the bilinear and RNN models differ by their choice of g.

3-2-3 Input-output versus state-space models

Input-output and state-space models are fundamentally different, since one contains a recursion within the regression vector while the other uses only observed data for the regression vector. Yet for some systems, it is possible to use both models. For example, if the output function h is invertible (i.e. there is a unique state for each sensor observation and vice versa), the system

$$\begin{aligned} x_{t+1} &= g(x_t, u_t), \\ \hat{y}_t &= h(x_t), \end{aligned} \tag{3-12}$$

can be converted to the first-order NARX model $\hat{y}_t = f(y_{t-1}, u_{t-1})$ by using $\hat{y}_t = h(g(h^{-1}(y_{t-1}), u_{t-1}))$. Conversely, an input-output model can in some cases also be realised by a state-space model. If this is the case, the order of the state-space system is lesser than or equal to the used number of delayed inputs and outputs [18]. In practice, this implies that input-output models often use more parameters than state-space models, especially for high-dimensional input or output data, making state-space models more attractive.

In robotic applications, the function g of equation (3-12) could represent the robot dynamics, while the function h could contain information about the environment. When using physical knowledge, splitting the predictor function in the functions f and g could give an advantage. However, in the case of sensors that correspond to direct observations of the environment (for example cameras or range-finders), the function h that approximates the sensor values for a given state might be very complex. In these cases, looking at the dynamics in the sensor space rather than the state space might be more beneficial. This can be done using input-output models.

The recursion in the state-space models allow for a long 'memory': prediction is not done for just one timestep, but the predicted output depends on the whole past. In input-output models it is also possible to get recursion. The predicted output can be used as the model input, giving a feedback loop, instead of using the observed sensor output each timestep. However, this might lead to instability, and it can be hard to analyse the stability conditions for these models, while state-space models have clear stability conditions even in the nonlinear case. When using recursion, whether in state-space or input-output models, system identification becomes more difficult. This is shown in Section 3-4.

3-3 Choosing the predictor function

Recall the predictor $\hat{y}_t = f(\varphi_{t-1})$, with predictor function $f : \mathbb{R}^l \to \mathbb{R}^n$. In this section the regression vector $\varphi \in \mathbb{R}^l$ is supposed to be known, and the time index will be omitted. After having chosen the regression vector, which is the input of the predictor function, the function f itself needs to be chosen. Since the relation between input and output data is unknown, f can be any function going from regressor space to output space. The desired function values of f are given by the output data y, making the problem in this chapter a *supervised learning* problem. There is a large variety of available techniques for this general problem, so only a few popular techniques are considered in this section.

For the nonlinear case, a common approach is to expand f into several *basis functions* f_k and take $f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k f_k(\varphi)$, with $\tilde{n}$ a chosen order of the model and coefficients $\alpha_k \in \mathbb{R}^n$. Most choices for f_k can be seen as a transformation of a *mother function* $\kappa : \mathbb{R}^l \to \mathbb{R}$. This gives

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k \kappa(\varphi, \beta_k, \gamma_k). \tag{3-13}$$

Here, β_k corresponds to a dilation and γ_k to a translation of the regression vector. With these parameters, each κ transforms the regression vector into a scalar, after which the parameters α_k are used to transform these values to the sensor space $\mathbb{R}^n$ in a linear fashion. Many blackbox modelling techniques use this form [6]. This form can be used for all input-output

models mentioned in Section 3-2-1, and for the state-space models from Section 3-2-2 for which the function h is linear[3]. Putting all parameters α_k, β_k and γ_k into one large parameter vector θ gives $\hat{y}_t = f(\varphi_{t-1}, \theta)$.

In the following subsections, three choices for parametrising κ are considered, namely: *ridge construction*, *radial construction* and *tensor products*. Within each subsection, some common choices for κ are discussed.

3-3-1 Ridge construction

In the ridge construction approach, the regression vector is multiplied with a vector $\beta_k \in \mathbb{R}^l$ and the parameter γ_k is subtracted, giving $\kappa(\varphi, \beta_k, \gamma_k) = \kappa(\beta_k^T \varphi - \gamma_k)$, with $\gamma_k \in \mathbb{R}$. Sometimes the subtraction comes before multiplication: $\kappa(\varphi, \beta_k, \gamma_k) = \kappa(\beta_k^T(\varphi - \gamma_k))$, in which case γ_k has the same dimension as the regression vector. The function f_k is now constant along a hyperplane $\{\beta_k^T \varphi \equiv \text{const.}\}$. Several popular approximation methods take this approach, for different choices of κ.

Fourier series

If φ is a scalar ($l = 1$), we can choose $\kappa(x) = \cos(x)$. In this case, the ridge construction approach is similar to the Fourier series expansion with the scalars β_k as the frequencies and scalars γ_k as the phases:

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k \cos(\beta_k(\varphi - \gamma_k)). \tag{3-14}$$

Continuous functions can be approximated arbitrarily well by Fourier series, as follows from the Riesz-Fischer theorem. The larger the value of $\tilde{n}$, the better the approximation. All techniques that are discussed in this section have this property.

The fourier series are given here as an example, and are not discussed further in this thesis.

Other one-dimensional examples

Some other examples of κ in case φ is a scalar, are the unit pulse $\kappa(x) = \mathbb{1}_{-0.5 \leq x < 0.5}$ and the unit step $\kappa(x) = \mathbb{1}_{x \geq 0}$, where

$$\mathbb{1}_S(x) = \begin{cases} 1, & x \in S, \\ 0, & x \notin S, \end{cases}$$

for any set S. Smooth versions of these are the Gaussian bell $\kappa(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ and the logistic function $\kappa(x) = \frac{1}{1+e^{-x}}$. Like the Fourier series example, these one-dimensional examples are not further discussed, since the regression vector φ of the problem in this thesis is in general not a scalar. However, the functions mentioned here are commonly used in techniques like neural or radial basis networks, which can handle regression vectors that are not necessarily scalars. These techniques will be discussed in the following subsections.

[3]For state-space models, function κ corresponds to function g, while parameters α_k correspond to the columns of the matrix of linear function h. This matrix was usually written as C in Section 3-2-2.

Sigmoid neural networks

The ridge construction approach is a popular approach used in *sigmoid neural networks* and *multilayer perceptrons*. Sigmoid functions are bounded differentiable functions with a positive derivative. Besides the logistic function, the hyperbolic tangent is a second popular choice for the sigmoid in neural networks. Using these neural networks, the predictor function f can be formulated as:

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k \sigma(\beta_k^T \varphi - \gamma_k), \tag{3-15}$$

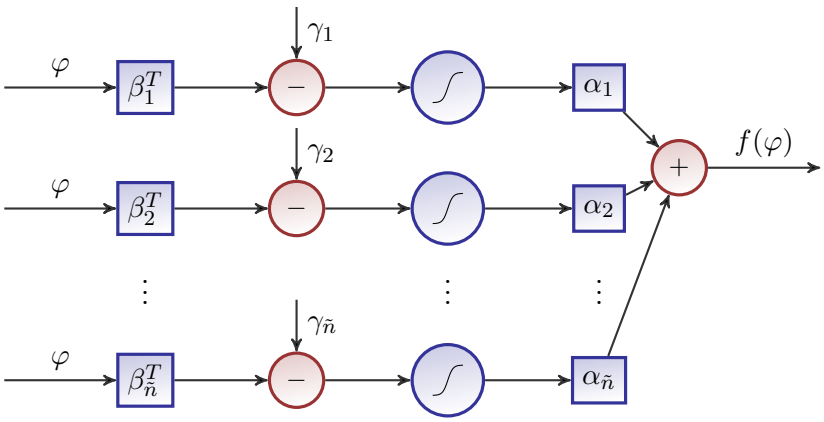


Figure 3-4: Structure of a feedforward neural network.

with σ a sigmoid function and $\kappa(x) = \sigma(x)$. The parameters β_k and α_k represent the input and output synaptic weights to and from hidden neuron k respectively, γ_k is called the *bias*, and $\tilde{n}$ corresponds to the number of *hidden neurons*. The structure of the network is shown in Figure 3-4. If the real output y depends continuously on the regression vector φ, the neural network 3-15 can approximate this relation up to arbitrary precision as long as $\tilde{n}$ is large enough [19].

Note that this notation can be used both for feedforward and for recurrent neural networks. The recurrent neural network (see Section 3-2-2) is a type of state-space model, while feedforward neural networks can be used for input-output models. For an input-output model, φ consists of past inputs and outputs of the real system but is called the input of the neural network. The term $\sigma(\beta_k^T \varphi - \gamma_k)$ represents the value of hidden neuron k, and $f(\varphi)$ becomes the output of the whole feedforward network. There is no recursion, so this network is called a feedforward model. For a state-space model, φ consists of the previous state and the previous input: $\varphi_{t-1} = [x_{t-1}^T \; u_{t-1}^T]^T$. A recurrent neural network has state transition $x_{t+1} = \sigma(\beta \begin{bmatrix} x_t \\ u_t \end{bmatrix} - \gamma)$, where $\beta = [\beta_1, \ldots \beta_{\tilde{n}}]^T$ and $\gamma = [\gamma_1, \ldots \gamma_{\tilde{n}}]^T$. This implies that some of the weights in β are recurrent: they connect x_t to x_{t+1}.

3-3-2 Radial construction

Instead of transforming the regression vector linearly by multiplication by β_k and translation by γ_k, one can use norms to transform the regression vector into a scalar. One possibility is taking the (Euclidean) norm of $\varphi - \gamma_k$ to get the distance between φ and the translation parameter γ_k. This value can then be scaled by $\beta_k \in \mathbb{R}$ to get $\kappa(\varphi, \beta_k, \gamma_k) = \kappa(\beta_k ||\varphi - \gamma_k||)$. The name of this approach comes from the fact that f is now constant along a sphere with γ_k as the center: $\{||\varphi - \gamma_k|| \equiv \text{const}\}$. Two techniques that make use of this approach are discussed below.

Radial basis networks

In radial basis networks, the function κ is usually a Gaussian: $\kappa(x) = e^{-x^2}$. These functions have the property that they approach zero when the distance between φ and γ is large. Using the scalar β_k multiplied by the distance between φ and γ_k, the predictor function f can be denoted as:

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k e^{-\beta_k ||\varphi - \gamma_k||^2}. \tag{3-16}$$

Here, γ_k represents the position of the center of the Gaussian, while β_k determines the width. The structure of the radial basis network is shown in Figure 3-5.

Like sigmoid neural networks, radial basis networks can approximate continuous functions up to arbitrary precision as long as $\tilde{n}$ is large enough [20].

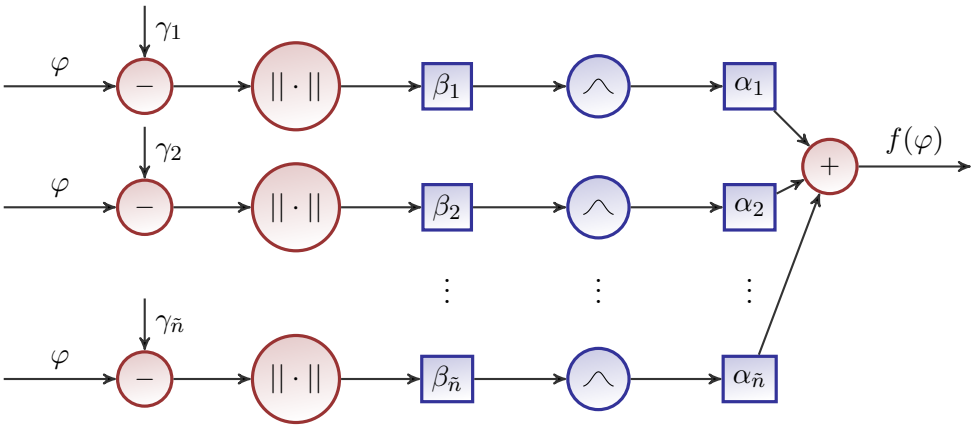


Figure 3-5: Structure of a radial basis network.

Local linear model

Radial basis networks can be generalised by using different α_k. In the approaches mentioned so far, α_k did not depend directly on the regression vector φ. It is, however, possible to see α_k as a function of φ. In the previous approaches, this would then be a constant function, but other

choices are possible. The most common example of this is in Local Linear models (LLMs), where α_k is a linear function of φ, for example $\alpha_k = A_k x_{t-1} + B_k u_{t-1} + O_k$ for the state-space case ($\varphi_{t-1} = [x_{t-1}^T, u_{t-1}^T]^T$). For a general regression vector φ this can be written as $\alpha_k(\varphi) = \mu_k \varphi + \nu_k$. Using this in combination with the radial basis network gives:

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \kappa(\beta_k ||\varphi - \gamma_k||)(\mu_k \varphi + \nu_k), \tag{3-17}$$

with a standard choice of $\kappa(x) = e^{-x^2}$. See [12] for an analysis and learning algorithms for local linear models.

One major advantage of the local linear model is its interpretability. Neural and radial basis networks do not have a clear interpretation when modelling a certain system, while the local linear model is just a weighted sum of linear models, and linear models are quite well understood. This implies that, when the optimal parameters $\alpha_k, \beta_k, \gamma_k$ have been found, these will also give some insight into the model.

Besides this, the LLM is also expected to have better approximation capabilities than, for example, a radial basis network. This is due to the higher number of free parameters: α_k of the previous models is an $n \times 1$ vector, as is ν_k, but μ_k is an $n \times l$-sized matrix. The downside of this is usually a higher computational cost for determining the optimal parameters. All of this is discussed in Chapter 4.

3-3-3 Tensor products

A third method to transform the input φ into a scalar, is by using the product of several functions of the input. Denote the regression vector as $\varphi = [\varphi_1, \ldots, \varphi_l]^T$, with l the dimension of the regression vector. Then, instead of using a function of a linear combination of the regressors (ridge construction) or a function of the distance between the regression vector and γ_k (radial construction), l functions of the regressors are multiplied: $\kappa(\varphi, \beta_k, \gamma_k) = \prod_{j=1}^{l} \kappa_j(\beta_{j,k}(\varphi_j - \gamma_{j,k}))$. This gives the following equation for the predictor function:

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \prod_{j=1}^{l} \kappa_j(\beta_{j,k}(\varphi_j - \gamma_{j,k})). \tag{3-18}$$

Fuzzy networks

In fuzzy logic and fuzzy set theory, mathematical statements like $s \in S$ can take a value between 0 and 1 rather than being true (1) or false (0). This branch of mathematics has also been applied to system identification [21]. One advantage of this approach is that vague linguistic statements like 'the speed is high' can be converted to a real value. This allows for so-called *fuzzy reasoning*, for example: 'if x is large, then y is small'. These statements can then have precise mathematical meanings using fuzzy logic.

A fuzzy set is defined as a function $A : U \to [0, 1]$, where U is any finite or infinite set. A statement like 'x is large' can then have a truth value of $A(x) \in [0, 1]$ for $x \in U$. The function A contains all information about how large each $x \in U$ is. In the following, a statement

will be denoted as $A(x)$. Using these continuous truth values, fuzzy logic extends the logical operators to continuous values.

One way to define the fuzzy logical **and** operator, is by using multiplication: $A(x) \wedge B(y) = A(x) \cdot B(y)$. Similarly, we can define fuzzy **or** and **not** operators by $A(x) \vee B(y) = A(x) + B(y) - A(x)B(y)$ and $\neg A(x) = 1 - A(x)$ respectively, although there are several alternatives. Most fuzzy networks consist of fuzzy rules of the form: **if** $A_1(x_1) \wedge A_2(x_2) \wedge \ldots \wedge A_l(x_l)$ **then** $B(z)$. If $B(z)$ is a non-fuzzy statement like $z = \alpha$, and multiplication is also used for implication, this rule can be denoted as $z = \alpha \prod_{j=1}^{l} A_j(x_j)$. Having $\tilde{n}$ of these rules with the regression vector as input, and letting each rule be defined by the same function transformed by dilation $\beta_{j,k}$ and translation $\gamma_{j,k}$, a similar notation as in previous sections can be used for the predictor function f:

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k \prod_{j=1}^{l} A\big(\beta_{j,k}(\varphi_j - \gamma_{j,k})\big). \tag{3-19}$$

It has been shown that under some conditions, this approach is equivalent to radial basis networks [22], which implies good approximation capabilities. Furthermore, this implies that fuzzy networks can be generalised to local linear models, just as radial basis functions can. A common type of fuzzy network that is equivalent to a local linear model is the Takagi-Sugeno fuzzy model [21]. This model is shortly analysed in Chapter 4.

3-4 Finding the optimal parameters

Once the regression vector φ has been chosen, as well as the parametrisation of the predictor function f, the parameters θ need to be fine-tuned so that the predictor $\hat{y} = f(\varphi, \theta)$ correctly predicts the output y, for several regression vectors φ. This can be achieved by minimising the total error, as described in Section 3-1:

$$\min_{\theta \in \Theta} E_t(\theta), \quad E_t(\theta) = \sum_{i=1}^{t} V\big(y_i - f(\varphi_{i-1}, \theta)\big). \tag{3-20}$$

The most common choice for V is the sum of squares $V(\epsilon) = ||\epsilon||^2$, with $||\cdot||$ the Euclidean norm. This causes equation (3-20) to become a least-squares problem. This is the approach that will be used in this section, although the maximum likelihood method, with $V(\epsilon) = -\log p(\epsilon)$, is a popular alternative. These two approaches are equivalent under the assumption of white Gaussian noise for the prediction errors[4].

Let θ^* be the (locally) optimal solution to equation (3-20). The problem of finding the optimal parameters θ^* is called the *learning process*. As denoted in Section 3-3, the problem

[4]In the maximum likelihood method, the prediction errors $\epsilon_t = y_t - \hat{y}_t$ are assumed to have a certain probability density function p. The function V is chosen to be $V(\epsilon) = -\log p(\epsilon)$. For white Gaussian noise, the errors ϵ_t are assumed to be i.i.d. with variance σ^2 and probability density function $p(\epsilon) = (2\pi)^{-n/2}|\sigma^2 I|^{\frac{1}{2}} e^{-\frac{1}{2\sigma^2}||\epsilon||^2}$. Since only the error ϵ depends on θ, the other terms and constants can be left out in the minimisation of the total error, giving: $\min E_t(\theta) = \sum_{i=1}^{t} -\log(e^{-||\epsilon||^2}) = \sum_{i=1}^{t} ||\epsilon||^2$. This is equivalent to using the sum of squares for V.

is a supervised learning problem, since the desired values of f are available after one timestep. It is also possible to distinguish between *offline* and *online* learning. In offline learning (also called batch learning), the parameters θ^* for input-output data up to timestep t need to be found without knowing the parameters θ of previous timesteps. In online learning, the chosen parameters θ are given for input-output data up until timestep $t-1$. Given new data of timestep t, these parameters then need to be adjusted towards the optimal θ^*. Most online algorithms have an efficient use of memory and computation time, but they need to be executed every timestep while an offline algorithm is executed only once.

To summarise, it is assumed that the following is known:

- Current timestep t.
- The regression vector φ_{t-1} as a function of input-output pairs (u_i, y_i) up to timestep $t-1$.
- The desired output y_t.
- A parametrisation of the predictor function f of the form: $f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k \kappa(\varphi, \beta_k, \gamma_k)$, with κ and $\tilde{n}$ known, and parameters $(\alpha_k, \beta_k, \gamma_k)$ only known in the case of online learning.

With this information, the goal in this section is to find the following optimal parameters:

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{i=1}^{t} ||y_i - f(\varphi_{i-1}, \theta)||^2. \tag{3-21}$$

A large number of techniques deals with this problem. The names of these techniques may differ, depending on the chosen regression vector and on the parametrisation of f (and on the scientific field). However, most techniques can be classified within one of two groups: linear or nonlinear optimisation techniques. In linear optimisation, the parameters β_k and γ_k are chosen by some method and stay fixed. After choosing β_k and γ_k, only the parameters α_k are optimised. This reduces the problem above to a linear least squares problem. In nonlinear optimisation, the most common approach is to perform a gradient descent on the error for all parameters $(\alpha_k, \beta_k, \gamma_k)$ to reach a local optimum.

The following subsections deal with methods for finding values of α_k, β_k, and γ_k. First, nonlinear least squares techniques are discussed, which give optimal values for these parameters. Then, the linear least squares problem for fixed β_k and γ_k is presented, giving a method for finding the optimal values of α_k only. This last method only works in combination with techniques for finding values for β_k and γ_k, so these techniques are presented next. Popular methods from several scientific fields are discussed.

3-4-1 Nonlinear least squares

The minimisation problem of equation (3-21) is a nonlinear least squares problem, for the techniques mentioned in this chapter. In general, these problems do not have an analytical solution, so iterative search techniques are the most common approach to find a local minimum for the error. In these approaches, the parameters θ are adapted each iteration in such a way that the error decreases. This can be done by using the gradient $G(\theta) = E'(\theta)^T = \frac{\partial E}{\partial \theta}(\theta)^T$, which is a vector of the same dimension as the number of parameters. Following the gradient can be seen as moving in parameter space in a 'downhill' fashion, for each iteration. This is illustrated in Figure 3-6.

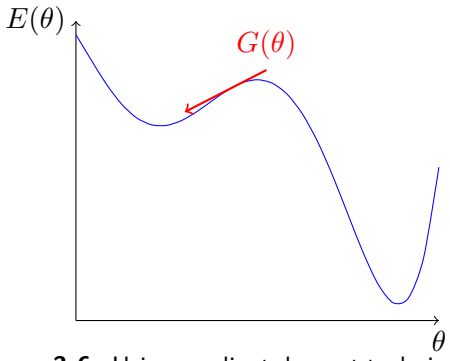


Figure 3-6: Using gradient descent techniques to find a local minimum of the error.

In a local optimum of E, the gradient is zero: $G(\theta) = 0$. This can be solved by Newton's method:

$$\begin{aligned} G(\theta + \Delta\theta) &\approx G(\theta) + G'(\theta)\Delta\theta = 0, \\ \implies \Delta\theta &\approx -G'(\theta)^{-1}G(\theta). \end{aligned} \tag{3-22}$$

This gives the iterative procedure $\theta_{i+1} = \theta_i + \mu\ \Delta\theta$ with

$$\Delta\theta = -\mu\ G'(\theta)^{-1}G(\theta) = -\mu\ E''(\theta)^{-1}E'(\theta)^T.$$

Here, μ is a small step size, and the second derivative E'' is a square matrix of dimension equal to the number of parameters squared. Since the number of parameters can be quite large, this term is usually substituted by a term R that is easier to compute, resulting in different techniques of the form $\Delta\theta = -\mu\ R(\theta)^{-1}E(\theta)^T$. Newton's method then takes this same form with $R = E''$.

Using Equation (3-20), the first and second derivative of E can be computed as follows:

$$E'(\theta) = \frac{\partial}{\partial\theta}\sum_{i=1}^{t}||y_i - f(\varphi_{i-1},\theta)||^2 = \sum_{i=1}^{t}2(y_i - f(\varphi_{i-1},\theta))^T\frac{\partial}{\partial\theta}f(\varphi_{i-1},\theta), \tag{3-23}$$

$$E''(\theta) = \sum_{i=1}^{t}2\frac{\partial}{\partial\theta}f(\varphi_{i-1},\theta)^T\frac{\partial}{\partial\theta}f(\varphi_{i-1},\theta) + 2(y_i - f(\varphi_{i-1},\theta))^T\frac{\partial^2}{\partial\theta^2}f(\varphi_{i-1},\theta). \tag{3-24}$$

To prevent the computation and inversion of this big matrix, several approximations of E'' can be used. These are discussed in the following techniques.

Gradient descent

Using $R = I$, with I the identity matrix, no matrix inversion is required. The parameters literally follow the gradient by the same amount for each iteration. This direction is the direction in which the error decreases fastest. However, this does not mean that the error converges in the least number of iterations: usually gradient descent is outperformed by other methods.

In neural network theory, the popular backpropagation algorithm uses an efficient implementation of gradient descent.

Gauss-Newton

In the Gauss-Newton method, we have $R(\theta) = H(\theta) = \sum_{i=1}^{t} f(\varphi_{i-1}, \theta)^T f(\varphi_{i-1}, \theta)$, ignoring the second term of the matrix E'' from Newton's method. From Equation (3-24) it can be seen that this is a good approximation if the error is small. Because H actually approximates E'', the Gauss-Newton method usually outperforms the gradient descent approach, at the cost of more computation time per iteration since this matrix needs to be inverted. Compared to Newton's method, one advantage is that H is always positive definite, making sure the error actually decreases, though convergence cannot be guaranteed.

Levenberg-Marquardt

The Levenberg-Marquardt method could be seen as a mix between Gradient descent and Gauss-Newton: $R(\theta) = H(\theta) + \mu I$. Larger δ makes the method more similar to gradient descent, and smaller δ makes it more similar to Gauss-Newton. It can also be seen as a trust-region approach of the Gauss-Newton method [23], making it more robust at the cost of more computation time per iteration.

Conjugate gradient method

The conjugate gradient method uses gradients from past iterations together with line optimisation to compute a new optimal search direction. Like in the other methods above, an approximation of E'' is used, making it more accurate than gradient descent. However, an advantage compared with the other methods is that no matrix inversion is necessary, just like in the gradient descent method.

3-4-2 Problems with recursion

In the nonlinear least squares methods mentioned above, the derivative of the error $E'(\theta) = \sum_{i=1}^{t} 2(y_i - f(\varphi_{i-1}, \theta))^T \frac{\partial}{\partial\theta} f(\varphi_{i-1}, \theta)$ was used. Since most functions that appeared in Section 3-3 are differentiable, and their derivatives are known, this should not be a problem. However, even for differentiable functions, a problem arises when the regression vector φ contains a recursion: the derivative $\frac{\partial}{\partial\theta} f(\varphi_{i-1}, \theta)$ then depends on all past data.

This can be seen by looking at the predictor function for the state-space models in unified notation:

$$\begin{aligned} \varphi_{t-1} &= [x_{t-1}^T, p(u_{t-1})^T]^T, \\ x_{t+1} &= g(x_t, p(u_t)), \\ f(\varphi_{t-1}) &= h(g(x_{t-1}, p(u_{t-1}))), \end{aligned} \tag{3-25}$$

where the unknown function h depends on α, and g and p depend on both β and γ. Even in case h is linear and p is the identity function, we have:

$$\begin{aligned} \frac{\partial}{\partial\theta} f(\varphi_{i-1}, \theta) &= \alpha\left(\frac{\partial}{\partial\theta} g(x_{t-1}, u_{t-1}) + \frac{\partial}{\partial x_{t-1}} g(x_{t-1}, u_{t-1}) \frac{\partial}{\partial\theta} x_{t-1}\right), \\ \frac{\partial}{\partial\theta} x_{t-1} &= \frac{\partial}{\partial\theta} g(x_{t-2}, u_{t-2}) + \frac{\partial}{\partial x_{t-2}} g(x_{t-2}, u_{t-2}) \frac{\partial}{\partial\theta} x_{t-2}. \end{aligned} \tag{3-26}$$

This implies that the gradient has to be calculated for all past timesteps. The NFIR and NARX models do not suffer from this problem, since the regression vector only depends on the input and output data for these models, which do not depend on the parameters θ. However, the NARMAX model does have a similar problem, because the prediction error appears inside the regression vector, which does depend on the parameters θ.

There exist many solutions to this problem. One of them is considered in the next subsection.

3-4-3 Linear least squares

Instead of optimising over all parameters θ, it turns out that optimising only the parameters α_k is an easier problem. Suppose that the parameters β_k and γ_k are given, for example by one of the methods that will be presented in the remainder of this chapter. The function f is then linear in the parameters α_k and the error is given by:

$$\begin{aligned} E_t(\alpha) &= \sum_{i=1}^{t} ||y_i - f(\varphi_{i-1}, \theta)||^2 \\ &= \sum_{i=1}^{t} ||y_i - \sum_{k=1}^{\tilde{n}} \alpha_k \kappa(\varphi_{i-1}, \beta_k, \gamma_k)||^2 \\ &= \sum_{i=1}^{t} ||y_i - \alpha\kappa(\varphi_{i-1}, \beta, \gamma)||^2, \end{aligned} \tag{3-27}$$

with $\alpha = [\alpha_1, \ldots, \alpha_{\tilde{n}}]$ and $\kappa(\varphi, \beta, \gamma) = \begin{bmatrix} \kappa(\varphi, \beta_1, \gamma_1) \\ \vdots \\ \kappa(\varphi, \beta_{\tilde{n}}, \gamma_{\tilde{n}}) \end{bmatrix}$. If the values of y_i and $\kappa(\varphi_{i-1}, \beta, \gamma)$ are put in columns of desired output matrix Y_t and construction matrix K_t respectively, this can be denoted in matrix notation as

$$E_t(\alpha) = ||Y_t - \alpha K_t||_F^2, \tag{3-28}$$

where $||\cdot||_F$ denotes the Frobenius norm. The iterative procedures from the previous subsection can be used for this problem. However, since this is a linear least squares problem, direct methods exist, circumventing the problem of calculating the gradient of the error. These linear regression techniques are illustrated in Figure 3-7.

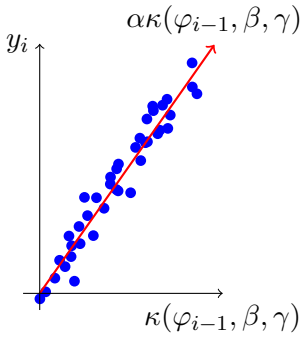


Figure 3-7: Using linear regression for parameter estimation.

An efficient direct method to find the parameters α that minimise this error, involves the use of the Moore-Penrose pseudo-inverse $K_t^\dagger$. The solution $\alpha = Y_t K_t^\dagger$ is the smallest-norm solution to equation (3-28) [24]. This implies that $||\alpha||_F$ is minimised as well. Minimising the norm of the parameters, besides the error, can have positive effects on the regularisation capabilities of the model (see Section 3-5).

There exist online algorithms that can efficiently calculate $K_t^\dagger$ when $K_{t-1}^\dagger$ is known [25]. Some online algorithms that do not make use of the Moore-Penrose pseudo-inverse, can also take weighting into account, for example when the prediction errors in the past should not count as heavily as recent prediction errors:

$$E_t(\alpha) = \sum_{i=1}^{t} \lambda^{t-i} ||y_i - \alpha\kappa(\varphi_{i-1}, \beta, \gamma)||^2.$$

Here, $\lambda \in (0, 1]$ is a forgetting factor. The recursive least-squares algorithm is one of the most popular techniques for this problem. Using this technique with $\lambda = 1$ is equivalent to minimising the error in equation (3-28).

Note that the dimension on the x-axis of Figure 3-7 is equal to $\tilde{n}$. Since it is easier to fit a linear model with the same number of datapoints in higher dimensions, it is expected that for larger $\tilde{n}$, a smaller error can be achieved.

Note also that the linear least-squares method gives a *global* solution to the minimisation problem $\min E_t(\alpha)$, with β and γ fixed. This does not mean it is a global solution for θ, so β and γ have to be chosen carefully. Some possibilities for choosing β and γ are discussed below.

Random Projection methods

An obvious first attempt for choosing β and γ, is by assigning random values to these parameters. Especially in applications where no prior physical knowledge is available, as is the case in this thesis, it is useful to analyse this method. Although simple in its essence, this approach has gained a lot of popularity recently, especially in the area of neural networks. The most popular examples are the Extreme Learning Machine (ELM) [26] for feedforward neural networks, and Reservoir Computing (RC) techniques [17] for recurrent neural networks. Although conceptually simple, these techniques tend to perform extremely well in many applications. Here, the essence of this technique will be described not just for neural networks but for the prediction error method in general, and the technique will be called Random Projection Method (RPM) in line with [27, 28].

The Random Projection Method for the Prediction Error Method can be defined as follows:

- Given timestep t, the regression vector φ_{t-1} and the predictor function $f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k \kappa(\varphi, \beta_k, \gamma_k)$, take the parameters β, γ from a continuous probability distribution and keep them fixed for every timestep t.
- Calculate the construction matrix $K_t = \begin{bmatrix} \kappa(\varphi_1, \beta_1, \gamma_1) & \dots & \kappa(\varphi_t, \beta_1, \gamma_1) \\ \vdots & & \vdots \\ \kappa(\varphi_1, \beta_{\tilde{n}}, \gamma_{\tilde{n}}) & \dots & \kappa(\varphi_t, \beta_{\tilde{n}}, \gamma_{\tilde{n}}) \end{bmatrix}$.
- Minimise the error $E_t(\alpha) = ||Y_t - \alpha K_t||_F^2$ for α, with Y_t the desired output matrix, using any linear least squares method.

For sigmoid neural networks and radial basis networks, it has been shown that these networks can still act as universal approximators when using the method above [29]. Chapter 4 of this thesis shows similar results for the local linear model.

Constructive methods

Instead of choosing random values for β and γ, information of the input and output values can be used if they are available. In radial basis networks, for example, it is common to choose some of the input samples and use them as the centers of the Gaussian functions. As an alternative, one could look at the range of possible input values and distribute the parameters in such a way that the input space is fully covered, creating a lattice. The input data can also be used to further optimise the parameters with clustering algorithms or Hebbian learning. Finally, one could try several parameters and see which one works best, possibly using heuristics like genetic algorithms.

3-5 Regularisation

Recall the total error from Equation (3-20): $E_t(\theta) = \sum_{i=1}^{t} V(y_i - f(\varphi_{i-1}, \theta))$. A model with a very small total error is not always a good model. The reason for this is that only the given input and output data are used for fitting the model, while future or unencountered data is not taken into account. When a model has a very low error for the given training data but might have a high error when encountering new data, this is called overfitting. One method to prevent overfitting is called regularisation.

It turns out that both $\tilde{n}$ and the size of the parameters play a key role in overfitting. Since reducing $\tilde{n}$ usually leads to a larger error, a common approach is to focus on minimising the size of the parameters, besides minimising the total error. This can be done by changing the objective function to:

$$\min_{\theta \in \Theta} W_t(\theta),$$

$$W_t(\theta) = E_t(\theta) + \delta||\theta|| = \sum_{i=1}^{t} V(y_i - f(\varphi_{i-1}, \theta)) + \delta||\theta||^2, \tag{3-29}$$

with δ the regularisation parameter. For the linear least squares method from Section 3-4-3, where optimisation is done over parameters α while parameters β and γ are fixed, this corresponds to Tikhonov regularisation or ridge regression:

$$\min_{\alpha \in \mathbb{R}^n} W_t(\alpha), \quad W_t(\alpha) = ||Y_t - \alpha K_t||_F^2 + \delta||\alpha||^2. \tag{3-30}$$

The method of using the Moore-Penrose pseudo-inverse can be seen as solving the Tikhonov regularisation with $\lim_{\delta \downarrow 0}$. For the nonlinear least squares methods, the Levenberg-Marquardt algorithm also uses a regularisation term, while in gradient descent or backpropagation algorithms it is more common to reduce the number of iterations rather than the size of the parameters.

Besides these regularisation methods, it is common to prevent overfitting by choosing the data carefully. For example, one could use only a selected subset of the data to fit the model, which is very common in the area of machine learning. Here, the data that is used for fitting the model is usually called the training set, while the rest of the data is called the validation

set. In this thesis, a different approach is used: the control has a large effect on the data that will be encountered. Choosing the control in such a way that the data is representative for all the data that the model might encounter in the future, could result in a better model. This is the topic of Chapter 5.

Chapter 4

Shuffled Linear Model

The main contribution of the previous chapter of this thesis is an overview of some popular techniques for fitting a prediction model for sensor and actuator data. For the practical application of this research, a choice has to be made for the regression vector φ, predictor function f and optimisation procedure. For the predictor function, the Local Linear model (LLM) from Section 3-3-2 has been chosen. The chosen optimisation procedure is the Random Projection Method (RPM) from Section 3-4-3, together with linear least squares as explained in Section 3-4-3. A choice for the regression vector has also been made. This choice, as well as the motivation for all of these choices, can be found in Chapter 6.

In this chapter, the LLM combined with the RPM and linear regression is analysed[1]. To the author's knowledge, this has not been done before. The proposal and analysis of this technique are therefore the main contribution of this thesis. It will be shown that under certain conditions, the combination of this model and choice of parameters can result in an accurate model for the relation between input and output data. The accuracy is also analysed, and it is proved that the model can act as a universal approximator.

Several interpretations exist for this model. Since the theorems in this chapter use similar proofs as the theorems in the literature of Extreme Learning Machines (ELMs), the main viewpoint of the model in this chapter will be an extension to ELMs.

This chapter starts with an introduction to ELMs and LLMs in Sections 4-1 and 4-2. Section 4-3 shows the proposed model, followed by an analysis of the approximation capabilities of the model in Section 4-4. Section 4-5 gives several interpretations of the model. This chapter is concluded with suggestions for further research in Section 4-6.

4-1 Introduction

The approximation of nonlinear functions or systems is a problem that appears in many disciplines [30] and that can be tackled by various approaches. One of these approaches is

[1] The content of this chapter has also been submitted in scientific paper format to arXiv: see http://arxiv.org/abs/1308.6498 for more information on this topic.

to decompose the problem into smaller problems and use a combination of simple solutions that only work for a small part of the problem. The combination of these local models should then approximate the global function or system. The LLM is a key example of this strategy, where a dynamical system is partitioned into multiple operating regimes, and for each regime a simple linear model is used to model the system.

If the goal is to approximate a function, not to model a system, the use of artificial neural networks has been a popular approach. One of the most popular algorithms for approximating a function with neural networks is the backpropagation algorithm (see also Section 3-4-1). The ELM algorithm as proposed by Huang et al. [26], is an alternative algorithm which outperforms the backpropagation algorithm in many aspects. Instead of tuning the parameters of the hidden layer and output layer, the hidden layer parameters are initialised randomly and kept fixed while the output weights are trained by a linear least-squares method. This was explained in Sections 3-4-3 and 3-4-3. Since very efficient algorithms are available for solving linear least-squares problems, the ELM algorithm is an efficient method for training a feedforward neural network.

Although ELMs can use nodes with many functions that do not at all resemble the activation functions observed in biological neurons [29], which makes ELMs a very general approach, to the author's knowledge they have not yet been extended in such a way that they include LLMs. The activation functions can be very general, but the hidden network nodes typically take one of two forms: either additive nodes, or Radial Basis Function (RBF) nodes are used, while the output is a linear combination of these. This corresponds to the ridge and radial constructions from Section 3-3. Unfortunately, with this architecture, this type of network can not be made equivalent to a LLM[2]. To prove that the same ELM algorithm does work for LLMs, an extension has to be made.

The ELM algorithm is extended in this chapter to include the use of LLMs, and the approximation capabilities are analysed and proved mathematically. Since ELMs use randomly initialised hidden layer parameters, the linear models used in the LLM will be accurate near a randomly chosen local point. This can be seen as shuffling a deck of cards and handing a card to each player, where the players are linear models and the cards represent the local points. The proposed model in this chapter is therefore called Shuffled Linear Model (SLM). Note that the linear models themselves are not initialised randomly, but follow from linear regression.

Extending the ELM algorithm to include LLMs gives several advantages, like a better physical interpretation and the potential to identify nonlinear systems, making the algorithm more fit for system identification. Besides this, the extension could be used in the area of fuzzy modelling. Fuzzy models are a popular approach to model nonlinear systems, and can be used to combine both rigid mathematical concepts and vaguer linguistic concepts, see Section 3-3-3. One of the most popular fuzzy models is the Takagi-Sugeno Model (TSM) [21]. TSMs use linear consequences in their fuzzy rules, making them equivalent to LLMs under some circumstances. This is also shortly analysed in this chapter, but the main point of this chapter is the extension of the ELM algorithm.

[2] The reason that the neural network architecture described in Section 3-3 is not fit for the LLM, is the fact that the output weights or basis function coefficients α_k do not depend on the regression vector. In the LLM, α_k are not parameters but rather (linear) functions of the regression vector φ. This can in fact be written in basis function format, as will be done in this chapter, but the result of this is a network that differs from the ridge or radial construction approach.

4-2 ELM summary

The ELM architecture is similar to a feedforward neural network with one hidden layer. The output neurons have a linear activation function, while the hidden neurons have a nonlinear activation function. The main difference with traditional neural networks is in the tuning of the parameters: the hidden layer parameters are initialised randomly and remain fixed, while only the output weights are tuned. This leads to a linear least-squares problem.

Although the original ELM used neural alike hidden nodes, ELMs have been generalised to include many activation functions that are not neuron alike [31]. The most common example is the RBF $\kappa(\varphi, \beta_k, \gamma_k) = e^{-\beta_k ||\varphi - \gamma_k||^2}$, with parameters $\gamma_k \in \mathbb{R}^l$, $\beta_k \in \mathbb{R}_{>0}$, and regression vector $\varphi \in \mathbb{R}^l$. Parameters γ_k and β_k are drawn from continuous probability distributions, according to the RPM. In this chapter, only ELMs with RBF nodes are considered.

For a network with $\tilde{n}$ hidden nodes, the output $\hat{y} \in \mathbb{R}^n$ of the ELM is:

$$\hat{y} = \sum_{k=1}^{\tilde{n}} \alpha_k \kappa(\varphi, \beta_k, \gamma_k) = \sum_{k=1}^{\tilde{n}} \alpha_k \kappa_k(\varphi), \tag{4-1}$$

with $\alpha_k \in \mathbb{R}^n$ the output weight vector connecting the k-th hidden node with the n output nodes. For t inputs φ_i, this can be written in matrix notation as

$$\hat{Y} = K\alpha, \tag{4-2}$$

where

$$K = \begin{bmatrix} \kappa_1(\varphi_1) & \dots & \kappa_{\tilde{n}}(\varphi_1) \\ \vdots & & \vdots \\ \kappa_1(\varphi_t) & \dots & \kappa_{\tilde{n}}(\varphi_t) \end{bmatrix}_{t \times \tilde{n}} \tag{4-3}$$

and

$$\hat{Y} = \begin{bmatrix} \hat{y}_1^T \\ \vdots \\ \hat{y}_t^T \end{bmatrix}_{t \times n}, \quad \alpha = \begin{bmatrix} \alpha_1^T \\ \vdots \\ \alpha_{\tilde{n}}^T \end{bmatrix}_{\tilde{n} \times n}. \tag{4-4}$$

If a similar $t \times n$-matrix Y consisting of target output values y is constructed, the goal is to minimise the sum of square errors:

$$E_t(\alpha) = \sum_{i=1}^{t} ||\hat{y}_i - y_i||^2 = ||\hat{Y} - Y||_F^2 = ||K\alpha - Y||_F^2, \tag{4-5}$$

where $||\cdot||_F$ denotes the Frobenius norm. Note that the matrices differ from those of Sections 3-4-3 and 3-4-3; some matrices are transposed to stay in line with the ELM literature.

The output weights α that minimise E, can be found by using the Moore-Penrose pseudo-inverse [24], here denoted as $\dagger$. This also makes sure that the norm of the output weights are minimised, for regularisation. Then the optimal solution is:

$$\alpha^* = K^\dagger Y. \tag{4-6}$$

4-3 Proposed model

The predictor function for the LLM was given in Section 3-3-2:

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \kappa(\beta_k ||\varphi - \gamma_k||) \cdot (\mu_k \varphi + \nu_k), \tag{4-7}$$

with $\kappa(x) = e^{-x^2}$, $\mu_k \in \mathbb{R}^{n \times l}$, and $\nu_k \in \mathbb{R}^n$. In the proposed model, the parameters β_k and γ_k are drawn from continuous probability distributions, just like in the ELM. The 'parameters' $\alpha_k(\varphi) = \mu_k \varphi + \nu_k$ are to be determined by linear regression. Note that these are not really parameters anymore, since they are a (linear) function of the regression vector. But μ_k and ν_k can be determined by linear regression, as will be shown in this chapter.

This model will be termed Shuffled Linear Model (SLM) in this thesis, since the Gaussians determine where a linear model is most valid, and these localities are chosen randomly by drawing parameters β_k and γ_k from continuous probability distributions. The parameters of the linear models themselves are not random, but optimised by linear regression. The output of the above equation will be denoted as SLM(φ).

4-4 Universal approximation with SLM

In this section, it will be proved that a SLM, as an extension of an ELM, can act as a universal approximator, and an upper bound on the number of models will be given. First, it will be proved that the SLM can act as a universal approximator. After that, several lemmas will be proved that lead to the main theorem of this paper, where Lemma 2 is used in the proof of Lemma 3, and Lemmas 1 and 3 are used in the proof of the main theorem. This section concludes with a learning algorithm for the SLM.

To prove that the SLM can act as a universal approximator, note that the SLM is equivalent to a radial basis network (see Section 3-3-2), in case $\mu_k = 0$:

$$\text{SLM}(\varphi)|_{\mu_k=0} = \sum_{k=1}^{\tilde{n}} \nu_k \kappa(\beta_k ||\varphi - \gamma_k||). \tag{4-8}$$

See also Equation (3-16). Now, choosing the parameters β_k and γ_k of a radial basis network randomly, and determining the other parameters from linear regression (in other words, using the RPM method), results in the ELM model [26]. This implies that the theory of ELMs can also be used for the SLM. The first theorem follows directly from the theory of ELMs, while the second one requires some lemmas, where the proof is similar to the proofs found in ELM theory.

Theorem 1. *SLMs are universal approximators. That is, given a continuous target function $f^* : \mathbb{R}^l \to \mathbb{R}^n$ and a SLM with free parameters μ_k and ν_k, $k = 1 \dots \tilde{n}$, then with probability one the parameters μ_k and ν_k can be chosen in such a way that*

$$\lim_{\tilde{n} \to \infty} \int_{\mathbb{R}^n} ||SLM(\varphi) - f^*(\varphi)||^2 dx = 0.$$

Proof. Choosing $\mu_k = 0$, $k = 1 \dots \tilde{n}$, the SLM is equivalent to an ELM. For an ELM, the parameters ν_k can be determined using linear regression to guarantee universal approximation [31, Thm. II.1]. □

The interesting part is, of course, whether the extension from ELM to SLM gives any improvements. It turns out that a SLM can also be trained using linear regression with all free parameters, just like an ELM. But for the same approximation accuracy, it needs less local models than the number of hidden nodes in the ELM. This is simply due to the fact that a SLM has more free parameters. The remainder of this section is devoted to the proof of these results, after which an algorithm for training the SLM will be presented.

4-4-1 Necessary lemmas

Before presenting the next theorem, some lemmas will be presented. Most proofs are similar to those of the ELM theorems [26, 31].

Lemma 1. *Given a SLM with inputs φ_i and desired outputs y_i, $i = 1 \dots t$, the total error $E_t = \sum_{i=1}^{t} ||SLM(\varphi_i) - y_i||^2$ can be minimised using linear regression.*

Proof. Let $\lambda_k = [\mu_k \ \nu_k]$ be the $n \times (l+1)$-matrix of the free parameters for model k, and let $\lambda = [\lambda_1 \ \dots \ \lambda_{\tilde{n}}]^T$ be the $(l+1)\tilde{n} \times n$-matrix of all free parameters. Let also $z_t = \begin{bmatrix} \varphi_t \\ 1 \end{bmatrix}$ and $\kappa_k(\varphi) = \kappa(\alpha_k, \beta_k, \varphi) = e^{-\beta_k ||\varphi - \alpha_k||^2}$, then the output of the SLM can be written as:

$$\mathrm{SLM}(\varphi) = \sum_{k=1}^{\tilde{n}} \kappa_k(\varphi) \lambda_k z. \tag{4-9}$$

Now the error E_t can be rewritten as follows:

$$\begin{aligned} E_t(\lambda) &= \sum_{i=1}^{t} ||\mathrm{SLM}(\varphi_i) - y_i||^2 \\ &= \sum_{i=1}^{t} || \sum_{k=1}^{\tilde{n}} \kappa_k(\varphi_i) \lambda_k z_i - y_i||^2 \\ &= \sum_{i=1}^{t} ||\lambda^T \begin{bmatrix} \kappa_1(\varphi_i) z_i \\ \vdots \\ \kappa_{\tilde{n}}(\varphi_i) z_i \end{bmatrix} - y_i||^2 \\ &= ||\lambda^T K^T - Y^T||_F^2, \\ &= ||K\lambda - Y||_F^2, \end{aligned} \tag{4-10}$$

with

$$Y = \begin{bmatrix} {y_1}^T \\ \vdots \\ {y_t}^T \end{bmatrix}_{t \times n}, \tag{4-11}$$

and

$$K = \begin{bmatrix} \kappa_1(\varphi_1)z_1^T & \dots & \kappa_{\tilde{n}}(\varphi_1)z_1^T \\ \vdots & & \vdots \\ \kappa_1(\varphi_t)z_t^T & \dots & \kappa_{\tilde{n}}(\varphi_t)z_t^T \end{bmatrix}_{t\times(l+1)\tilde{n}}. \tag{4-12}$$

This sum of squared errors can be minimised using linear regression, for example by using the minimum norm least-square solution

$$\lambda^* = K^\dagger Y. \tag{4-13}$$

This completes the proof. □

Lemma 2. *Given t distinct samples $(\varphi_i, y_i) \in \mathbb{R}^l \times \mathbb{R}^n$, if $\gamma_k \in \mathbb{R}^l$ is chosen randomly from a continuous probability distribution, then with probability one we have $||\varphi_i - \gamma_k|| \neq ||\varphi_j - \gamma_k||$, for $i \neq j$, $i, j = 1 \dots t$.*

Proof. The proof is an adaptation to one of the proofs of Huang and Babri [32]. Consider the set $V(\varphi_i, \varphi_j) = \{\gamma_k \in \mathbb{R}^l : ||\varphi_i - \gamma_k|| = ||\varphi_j - \gamma_k||\}$. This set is a hyperplane in $\mathbb{R}^l$. Since this is an $(l-1)$-dimensional surface in $\mathbb{R}^l$, and there are only finite i, j, the union $U = \bigcup_{i\neq j} V(\varphi_i, \varphi_j)$ is a finite union of $(l-1)$-dimensional surfaces for $i \neq j$, $i, j = 1 \dots t$. So, for any probability density function f, we have $P(\gamma_k \in U) = \int_U f(\gamma_k)d\gamma_k = 0$.

Therefore, the probability that $||\varphi_i - \gamma_k|| = ||\varphi_j - \gamma_k||$ for $i \neq j$, is 0 for randomly chosen γ_k, and the result of the lemma follows. □

Lemma 3. *Matrix K from Equation (4-12) has full rank with probability one if the matrix $Z = [z_1 \ \dots \ z_t]^T$ has full rank.*

Proof. For $k = 1 \dots \tilde{n}$, $j = 1 \dots l+1$ and $l = k \cdot j$, the l-th column of K is given by $[\kappa_k(\varphi_1)z_{1_j} \ \dots \ \kappa_k(\varphi_t)z_{t_j}]^T$, where $\kappa_k(\varphi) = \kappa(\varphi, \beta_k, \gamma_k) = e^{-\beta_k||\varphi-\gamma_k||^2}$. Let $I = (u, v)$ be any interval from $\mathbb{R}$, with $u < v$. Let $p : I \to \mathbb{R}^t$ be the curve defined as

$$p(\beta_k) = [\kappa_k(\varphi_1) \ \dots \ \kappa_k(\varphi_t)]^T, \tag{4-14}$$

seen as a function of β_k, for $\beta_k \in I$. Using a similar proof as that of Tamura and Tateishi [33], it can be proved by contradiction that $p(\beta_k)$ does not belong to a subspace with dimension less than t.

Suppose it does, then there exists a vector $w \neq 0$ orthogonal to this subspace:

$$w^T(p(\beta_k) - p(u)) = \sum_{i=1}^{t} w_i\kappa(\alpha_k, \beta_k, \varphi_i) - w^T p(u) = 0. \tag{4-15}$$

Without loss of generality, assume $w_t \neq 0$, then

$$\kappa(\varphi_t, \beta_k, \gamma_k) = \frac{1}{w_t}w^T p(u) - \sum_{i=1}^{t-1} \frac{w_i}{w_t}\kappa(\varphi_i, \beta_k, \gamma_k). \tag{4-16}$$

On both sides of this equation are functions of β_k. Taking the s-th partial derivative to β_k on both sides gives the following equation:

$$\frac{\partial^s}{\partial \beta_k^s}\kappa(\varphi_t, \beta_k, \gamma_k) = -\sum_{i=1}^{t-1} \frac{w_i}{w_t} \frac{\partial^s}{\partial \beta_k^s}\kappa(\varphi_i, \beta_k, \gamma_k). \tag{4-17}$$

Now, κ is infinitely differentiable w.r.t. β_k, with

$$\frac{\partial^s}{\partial \beta_k^s}\kappa(\varphi_i, \beta_k, \gamma_k) = (-||\varphi_i - \gamma_k||^2)^s e^{-\beta_k||\varphi_i - \gamma_k||^2}.$$

Using this fact and Lemma 2, it follows that with probability one, Equation (4-17) actually contains an infinite number of equations that are linear in parameters w_i, for $s = 1, 2, \ldots$. The number of free parameters w_i, however, is t. This gives a contradiction. Hence, $p(\beta_k)$ does not belong to a subspace with dimension less than t.

Since the above contradiction holds for β_k from any interval $I \subseteq \mathbb{R}$, it is possible to choose $\tilde{n}$ values $\beta_1, \beta_2, \ldots, \beta_{\tilde{n}}$ from any continuous probability distribution over $\mathbb{R}$, such that the following matrix K has full rank:

$$K = [p(\beta_1)\ p(\beta_2)\ \ldots\ p(\beta_{\tilde{n}})]. \tag{4-18}$$

This is the same $t \times \tilde{n}$ matrix as the hidden layer output matrix from equation 4-3. The matrix $Z = [z_1\ \ldots\ z_t]^T$ is a $t \times (l+1)$-matrix. Let $M = K \otimes Z$ be the Kronecker product of these matrices. Then M is a $t^2 \times (l+1)\tilde{n}$-matrix. Since both K and Z have full rank, and $\text{rank}(A \otimes B) = \text{rank}(A)\text{rank}(B)$ for any matrices A and B, the matrix M has rank $r = \min(\tilde{n}, t)\min(l+1, t)$.

Now, matrix K actually appears inside matrix M. For matrix M, we have:

$$M = K \otimes Z = \begin{bmatrix} \kappa_1(\varphi_1)z_1^T & \ldots & \kappa_{\tilde{n}}(\varphi_1)z_1^T \\ \vdots & & \vdots \\ \kappa_1(\varphi_1)z_N^T & \ldots & \kappa_{\tilde{n}}(\varphi_1)z_N^T \\ \vdots & & \vdots \\ \kappa_1(\varphi_N)z_1^T & \ldots & \kappa_{\tilde{n}}(\varphi_N)z_1^T \\ \vdots & & \vdots \\ \kappa_1(\varphi_N)z_N^T & \ldots & \kappa_{\tilde{n}}(\varphi_N)z_N^T \end{bmatrix}. \tag{4-19}$$

The $t \times (l+1)\tilde{n}$-matrix K from equation (4-12) follows from this matrix by removing rows until only the rows with indices $(i-1)(t+1)+1$ are left, for $i = 1, \ldots, t$. It follows that the rank of matrix K is equal to $\min(t, r) = \min(t, (l+1)\tilde{n})$, so matrix K has full rank. □

4-4-2 Improvement of SLM as an extension of ELM

Using the Lemmas that appeared in this section, the main theorem of this paper, which is similar to that of Huang [26, Thm. 2.2], can be presented. From the proofs of the Lemmas, the following definitions are needed:

$$\begin{aligned} z_i &= \begin{bmatrix} \varphi_i \\ 1 \end{bmatrix}, \\ Z &= [z_1\ \ldots\ z_t]^T. \end{aligned} \tag{4-20}$$

Theorem 2. *Given any $\epsilon > 0$, and given t distinct samples $(\varphi_i, y_i) \in \mathbb{R}^l \times \mathbb{R}^n$, if matrix Z has full rank, then there exists $\tilde{n} \leq t/(l+1)$ such that a SLM with $\tilde{n}$ local models can be trained using linear regression to get $E_t = \sum_{i=1}^{t} ||SLM(\varphi_i) - y_i||^2 < \epsilon$ with probability one.*

Proof. The error E_t can actually become zero by choosing $\tilde{n} = t/(l+1)$. Since Z has full rank, from Lemma 3 it follows that matrix K also has full rank, with probability one. Then K is also invertible, since the size of K is $t \times (l+1)\tilde{n} = t \times t$. The solution showed at the end of the proof of Lemma 1 is now actually equal to $\lambda^* = K^{-1}Y$, and the error $E_t = ||K\lambda^* - Y||_F^2$ is zero.

Since zero error can be achieved by choosing $\tilde{n} = t/(l+1)$, there always exists an $\tilde{n} \leq t/(l+1)$ to let the error be as small as desired. □

Note that the condition that Z has full rank can be easily satisfied, for example by adding noise to the training samples. This would in fact also be beneficial for regularisation.

Theorem 2 shows that the SLM needs less local models than the number of hidden nodes in an ELM. This does not necessarily imply a decrease in computation time, since the matrix of which a pseudo-inverse needs to be computed, has size $t \times (l+1)\tilde{n}$, not size $t \times \tilde{n}$. An ELM with $\bar{n}$ hidden nodes and a SLM with $\tilde{n} = \bar{n}/(l+1)$ local models are therefore comparable in both approximation capabilities and in computational efficiency. The construction of the matrix for which a pseudo-inverse needs to be computed, however, can be done more efficiently due to the block structure of K, so there is some gain in computational efficiency for a large number of models or hidden neurons. There might also exist a more efficient way to calculate the pseudo-inverse of this block matrix.

It is questionable whether it is desired to use less nodes or models in an approach that uses random nodes and models, since the model performance should not depend highly on parameters that are chosen randomly.

Still, there are several advantages of the SLM, compared to an ELM. As mentioned above, the matrix K can be constructed efficiently because it contains a block structure, and maybe in the future an efficient method to calculate the pseudo-inverse of this matrix could be found. Besides this, the SLM is not as "black-box" as an ELM, because the output matrix λ actually shows the direct connections between input and output near several operating points of the input space. In contrast, the output weights of an ELM only show relations between the hidden layer and the outputs, not between input and output. Finally, the SLM allows for a clear physical interpretation due to its similarities with local linear models from the area of system identification, and its similarities with fuzzy inference models (see Section 4-5-3).

4-4-3 Proposed algorithm

The proposed learning algorithm for the SLM is shown in Figure 4-1. After running the algorithm, the output of the SLM for input φ is given by $\text{SLM}(\varphi) = \lambda^T \begin{bmatrix} \kappa_1(\varphi)z \\ \vdots \\ \kappa_{\tilde{n}}(\varphi)z \end{bmatrix}$, for $z = \begin{bmatrix} \varphi \\ 1 \end{bmatrix}$.

SLM learning algorithm

1. Given t input-output pairs $(\varphi_i, y_i) \in \mathbb{R}^l \times \mathbb{R}^n$, randomly generate parameters $\alpha_k \in \mathbb{R}^l$ and $\beta_k \in \mathbb{R}_{>0}$ from a continuous probability distribution independent of the input.
2. Choose the number of models $\tilde{n}$ and let $\kappa_k(\varphi) = e^{-\beta_k||\varphi-\alpha_k||^2}$ for $k = 1, \ldots, \tilde{n}$.
3. For $z_i = \begin{bmatrix} \varphi_i \\ 1 \end{bmatrix}$, calculate the $t \times (l+1)\tilde{n}$-matrix
$$K = \begin{bmatrix} \kappa_1(\varphi_1)z_1^T & \ldots & \kappa_{\tilde{n}}(\varphi_1)z_1^T \\ \vdots & & \vdots \\ \kappa_1(\varphi_t)z_t^T & \ldots & \kappa_{\tilde{n}}(\varphi_t)z_t^T \end{bmatrix}.$$
4. Put the desired outputs y_i in matrix $Y = [y_1 \ \ldots \ y_t]^T$ and calculate $\lambda = K^\dagger Y$ by using the Moore-Penrose pseudo-inverse.

Figure 4-1: Proposed SLM algorithm.

4-5 Interpretation of the model

Several interpretations of the SLM are possible. These interpretations will be described in the following subsections.

4-5-1 ELM interpretation

The main interpretation of the SLM in this chapter is the extension of an ELM with RBF nodes. The output weights α_k of an ELM can be seen as constant functions, since they do not depend directly on the input. After training an ELM, the output weights are the same for every possible input of the network. In the SLM extension, the output weights are not constants, but functions of the input. Instead of multiplying the outputs of the hidden layer $\kappa_k(\varphi)$ with constant output weights α_k, they are multiplied with a linear function of the input: $\mu_k\varphi + \nu_k$. This also changes the interpretation of output weight to linear model, and from hidden layer output to model weight.

Since only the parameters in the hidden layer are fixed, a SLM has more free parameters than an ELM. The vector ν_k is of size n and the matrix μ_k is of size $n \times l$. If $\tilde{n}$ is the number of local models in a SLM and $\bar{n}$ the number of hidden neurons in an ELM, this implies that the SLM has $n \times (l+1)\tilde{n}$ free parameters, while an ELM has only $n \times \bar{n}$ free parameters. A logical conclusion is that an ELM needs $l+1$ times as many neurons as the number of local models in a SLM, for similar approximation capabilities. This is true under the conditions shown in Theorem 2.

4-5-2 RBF interpretation

A SLM can be interpreted as a RBF network, but there are some important differences. In the first place, it is a generalisation of the RBF network from Section 3-3-2. Several generalisations of RBF networks exist [34, 35], but the one used in this chapter is similar to the one proposed by Hunt, Haas and Murray-Smith [36]. Using the notation from this chapter, a standard RBF network can be denoted as $\hat{y} = \sum_{k=1}^{\tilde{n}} \alpha_k \kappa_k(\varphi)$, while a generalised RBF network is denoted as $\hat{y} = \sum_{k=1}^{\tilde{n}} \alpha_k(\varphi)\kappa_k(\varphi)$, letting the output weights depend on the input[3]. Using $\alpha_k(\varphi) = \mu_k\varphi + \nu_k$, where the output weights are a linear function of the input, this is equivalent to a SLM.

In the second place, a SLM differs from a RBF network in the way the hidden layer parameters are chosen. It is customary in RBF networks to let the location of the centers of the Gaussians γ_k be the same as some of the training samples φ_i, or else to determine them by some sort of clustering algorithm. In a SLM, both the centers γ_k as well as the widths β_k of the Gaussians are taken from a continuous probability distribution, and are therefore independent of the training samples[4]. This has the advantage that no prior information about the training samples is necessary, and that it takes practically no time to determine the parameters.

Figure 4-2 shows the structure of a local linear model in a similar fashion as the structure of the RBF network showed in Figure 3-5.

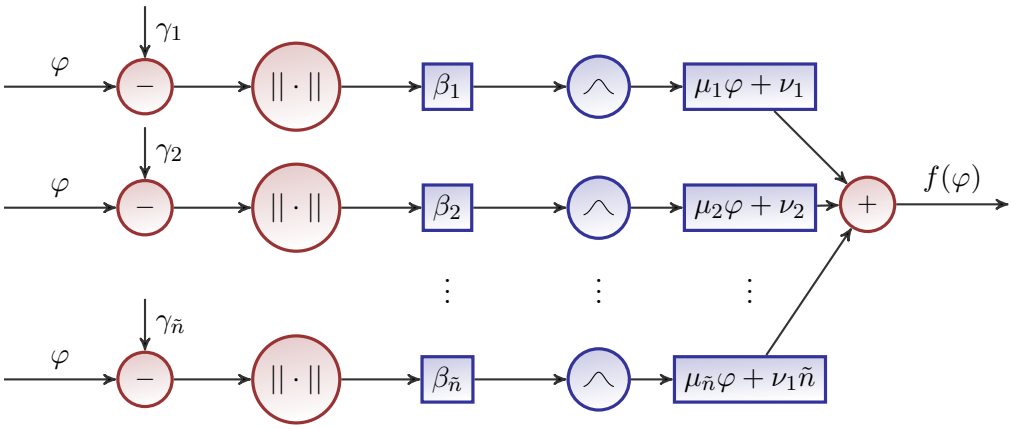


Figure 4-2: Structure of the Shuffled Linear Model. Parameters β_k and γ_k are fixed randomly, while parameters μ_k and ν_k follow from linear regression.

[3] The generalised RBF as defined by Hunt, Haas and Murray-Smith, differs from a standard RBF network in three aspects. Only one of these aspects is considered here, namely that the output weights can depend on the input.

[4] The probability distribution of the parameters γ_k and β_k can be any continuous probability distribution, for example the normal or the uniform distribution. However, the distribution of β_k should be such that $P(\beta_k \leq 0) = 0$, since $\beta_k \in \mathbb{R}_{>0}$. In theory this is the only constraint, but in practice there might be more constraints due to rounding errors in the numerical evaluations of the Gaussians.

4-5-3 Fuzzy interpretation

The SLM can also be compared to the fuzzy Takagi-Sugeno Model (TSM). Comparisons between the TSM and a generalised radial basis network have already been made [36]. Following the same line of thought, it is not difficult to show the equivalence between the TSM and SLM models. The TSM consists of a number of fuzzy if-then rules with a fuzzy premise and a linear consequence, e.g.

$$\begin{aligned} R_k \quad : \quad & \text{if } \varphi_1 \text{ is } A_1 \ \wedge \ \varphi_2 \text{ is } A_2 \ \wedge \ \cdots \ \wedge \ \varphi_l \text{ is } A_l \\ & \text{then } \alpha_k(\varphi) = \mu_k \varphi + \nu_k \end{aligned} \tag{4-21}$$

The TSM is equivalent to the generalised RBF network, and therefore also to the SLM, if 1) the membership functions within each rule are all Gaussians, and 2) the operator used for the fuzzy $\wedge$ is multiplication. Using multiplication for the if-then part as well, and letting $\kappa_{jk}(\varphi)$ denote the fuzzy membership function of the fuzzy set A_j of fuzzy rule R_k, this result comes as follows:

$$\begin{aligned} TSM(\varphi) &= \sum_{k=1}^{\tilde{n}} \prod_{j=1}^{l} \kappa_{jk}(\varphi^{(j)})(\mu_k \varphi + \nu_k) \\ &= \sum_{k=1}^{\tilde{n}} \prod_{j=1}^{l} e^{-\beta_k (\varphi^{(j)} - \gamma_k^{(j)})^2} (\mu_k \varphi + \nu_k) \\ &= \sum_{k=1}^{\tilde{n}} e^{-\beta_k \sum_{j=1}^{l} (\varphi^{(j)} - \gamma_k^{(j)})^2} (\mu_k \varphi + \nu_k) \\ &= \sum_{k=1}^{\tilde{n}} e^{-\beta_k ||\varphi - \gamma_k||^2} (\mu_k \varphi + \nu_k) \\ &= \text{SLM}(\varphi). \end{aligned} \tag{4-22}$$

A SLM can therefore be seen as a TSM with randomly generated membership functions for the premises of the fuzzy rules. Using this result and Theorem 2, the following holds:

Corollary 1. *Theorem 2 also holds for the Takagi-Sugeno fuzzy model with $\tilde{n}$ fuzzy rules, if the membership functions for each rule are Gaussians with parameters chosen randomly from continuous probability distributions, and the operator used for the fuzzy $\wedge$ is multiplication.*

4-5-4 LLM interpretation

LLMs usually appear within the context of nonlinear dynamical systems, where the nonlinear system is approximated by taking a weighted sum of simpler (linear) models. These linear models can be seen as linearisations of the nonlinear system around an operating point, and the weights are used to describe the transitions between the operating points. The problem of using a LLM to approximate a nonlinear dynamical system, belongs to the area of system identification. Several overviews of identification techniques for local models exist [37, 12].

The input of such a *dynamical* LLM consists of past inputs and outputs or states of the system, while the output of the dynamical LLM should give the next output or state of the system, e.g.

$$s_{t+1} = f(s_t, u_t), \tag{4-23}$$

where s is the output or state, and u the control input of the dynamical system. If the function f is differentiable, it can be linearised around an operating point $(s^{(k)}, u^{(k)})$, to get:

$$\begin{aligned} f(s_t, u_t) &\approx f(s^{(k)}, u^{(k)}) + \frac{\partial f}{\partial s_t}(s^{(k)}, u^{(k)})(s_t - s^{(k)}) \\ &\quad + \frac{\partial f}{\partial u_t}(s^{(k)}, u^{(k)})(u_t - u^{(k)}) \\ &= A_k s_t + B_k u_t + O_k, \end{aligned} \tag{4-24}$$

where A_k and B_k are the partial derivatives of f w.r.t. s and u respectively, and $O_k = f(s^{(k)}, u^{(k)}) - A_k s^{(k)} - B_k u^{(k)}$. The local linear model is then a weighted sum of these linearisations:

$$LLM(s_t, u_t) = \sum_{k=1}^{\tilde{n}} \kappa_k(s_t, u_t)(A_k s_t + B_k u_t + O_k), \tag{4-25}$$

where κ_k is a scalar-valued function that represents the validity of the local model. The relationship between these kinds of dynamical LLMs and TSMs has also been investigated [38].

Replacing (s_t, u_t) by a general φ_t, the model can be used for both dynamical systems and functions in general.

Now, for a SLM, the weights are Gaussian functions of the input, but the parameters of the Gaussians (i.e. their centers and widths, or the operating points and regimes) are chosen randomly. This implies that the linear models are supposed to be good approximations of the global system around randomly chosen operating points, while the traditional approach is to use a gradient-descent type algorithm to determine the optimal operating points and regimes. Theorem 2 shows that even with randomly chosen operating points and regimes, it is possible to approximate any function with any desired accuracy, as long as the number of local models $\tilde{n}$ is high enough. This can be done with a direct algorithm, as an efficient alternative to iterative methods.

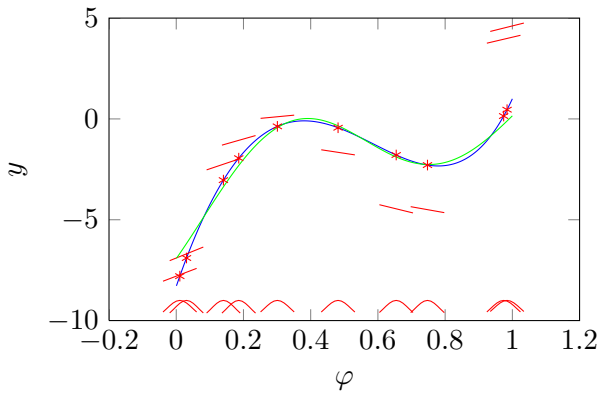


Figure 4-3: Example of a SLM. At each local point γ_k (denoted by an asterisk), which are all chosen randomly, a linear model (line segment) is multiplied with a Gaussian (curve near the horizontal axis) to approximate the original function locally. The sum of these local models gives the output of the SLM.

Figure 4-3 shows an example of using an SLM to approximate a one-dimensional function. Operating points are chosen randomly and are denoted with an asterisk. Gaussian widths are chosen randomly as well, and are showed at the bottom of the graph (in this case, the

widths do not differ much). Linear models are found by linear regression, and are showed as line segments. When the linear models are multiplied with their respective Gaussians and added together, this results in an approximation of the function.

4-6 Further research

The SLM as discussed in this chapter gives rise to many questions. A few questions for further research could be:

- For which supervised learning problems is the SLM the best option?
- Can nonlinear local models be used in the same way as the linear models in the SLM?
- Can the SLM also be used for other (un)supervised learning problems?
- What would be the effect of normalisation in the RBF nodes?
- Which probability distributions should be used for the randomly chosen parameters?

For the first question, the paper on arXiv[5] shows one example where the SLM works, though it is not necessarily the best model to use, so the question remains unanswered. Since this thesis already has one practical application as a goal, this question is not further discussed.

As for the second question, Appendix B shows an example with a local bilinear model that can be used in the same way as the SLM. In fact, any model for which the model output depends linearly on the model parameters could be used in a similar fashion as the linear models in the SLM. This is due to the fact that the linearity of the relation between local model output $\alpha_k(\varphi)$ and regression vector φ is not used; only the linearity of the relation between local model output $\alpha_k(\varphi)$ and its parameters is used. This result greatly increases the number of models that can be used in a similar way as the linear models in the SLM approach.

The third question can actually be answered by looking at the link between neural networks and kernel methods. It turns out that the SLM actually gives rise to a specific kernel: a Gaussian kernel multiplied with a linear kernel: $K(x, y) = e^{-\beta||x-y||^2} x^T y$. This is proved in Appendix B. Many supervised and unsupervised learning problems, like classification, clustering and dimension reduction, can be solved using kernel methods [39].

The last two questions are not answered in this thesis and remain open for further research. However, the last question should be answered somewhere in this thesis if the SLM model is to be used. This is done in the next chapter. The choice of the probability distribution for the parameters in the next chapter is mostly based on insight and trial and error, so research in this direction could be beneficial.

[5] http://arxiv.org/abs/1308.6498

Chapter 5

Control for System Identification

In the previous chapter, the prediction error method was used to construct a predictor f to approximate the relation between input and output data: $y_t = f(\varphi_{t-1}) + \epsilon_t$, where φ_{t-1} is the regressor consisting of past inputs and outputs, and ϵ is the prediction error. Control was not considered in the previous chapter, but in this chapter the focus will be on using the control in such a way that it aids the prediction process of the agent. The prediction process is defined in this thesis as fitting a model that can correctly predict the next sensor data, given past sensor and actuator data.

Since the last chapter was also dedicated to this same goal, the role of control in this process must be clearly understood. Control can be used to determine which data the model will encounter in the future. For example, a robot that stands still the whole time will probably also encounter the same data each timestep. This is easy to learn, but not the desired outcome: the model should accurately be able to predict the next sensor data given past sensor and actuator data, for various situations and various choices of the control. Besides the regularisation element discussed in Section 3-5, there should be some kind of exploration in the agent to encounter various situations. The control can play a key part in this exploration, and should be used to get this kind of behaviour in the agent.

In this chapter, the concept of reinforcement learning and intrinsic motivation is explained in Section 5-1, and the role of control is also further explained. The other sections examine various techniques using intrinsic motivation to get the desired behaviour in an agent.

5-1 Intrinsic motivation

A common technique to let an agent perform the desired behaviour, is reinforcement learning [40]. When the agent performs the desired behaviour, it is rewarded, and when it does not, it might receive a penalty. The control is then chosen in such a way that the future reward is maximised. An example is to let an agent receive a reward proportional to its distance to some target location. In this example, it is expected that the reward-maximising agent will eventually reach the target location.

In this thesis, the desired behaviour of the agent is to let it explore various situations while predicting the next sensor data correctly. This is a big difference with most reinforcement learning problems, where external information is used to determine for example the distance to a target location. In the problem of this thesis, no external information is available. The only 'target' is some vague term like 'exploratory behaviour', and the only information available is the past sensor and actuator data, and the information of the predictor.

It turns out that the information of the predictor, especially the prediction error, can be very useful in influencing the agent's behaviour. Inspired by psychological concepts, using this internal information in combination with reinforcement learning, has been termed 'intrinsic motivation' [41]. Several methods fall under this category.

5-2 Exploration versus exploitation trade-off

Before considering the various techniques that use intrinsic motivation, the problem should be defined more precisely. The research question of this chapter is the following: **How can control be used to aid in the learning process?** The goal can be defined more precisely as follows:

> **Given at timestep t a predictor f as proposed in Chapter 3 and past input-output data φ_{t-1}, choose input data u_t in such a way that the future total error $E_T = \sum_{i=1}^{T} ||y_i - f(\varphi_{i-1})||^2$ is minimised for as many different u_i and y_i as possible.**

Here, $T > t$ is some future timestep, possibly at infinity, that indicates when to stop doing what we are doing in this thesis: learning the relation between sensor and actuator data. At this timestep, it is desired not only that the prediction errors have been minimised, but also that this has been done for as many different situations as possible.

As will be made clear in this chapter, these goals are contradictory. For many models, dealing with new data, also called validation or generalisation, is a difficult task, and usually causes large errors. The two subgoals can be seen as an exploitation vs. exploration trade-off. While keeping the error low, the knowledge of the predictor is exploited. At the same time, the controller is used to explore new situations to make sure that the u_i and y_i are different. Several methods to achieve both of these subgoals are investigated in the following sections.

5-3 Methods using intrinsic motivation

5-3-1 Random movement

The first method discussed here does not really make use of any information available to the robot. At the first timestep, there is no past input or output data so the choices for the control are limited. Since no external knowledge is available to the autonomous robot, it makes sense to just choose a random control value in this case. A probability distribution F has to be defined over all possible control values, so this is knowledge that should be available to the agent. Then the control takes the form

$$u_t \sim F. \tag{5-1}$$

Using random movement for several timesteps, it is expected that exploratory behaviour does occur: different u_t will be chosen due to the randomness. However, this does not imply a diversity in the sensor data y_t. For example, if an agent can move in different directions, and the control for each direction has the same probability of being chosen, it is expected that random movement will cause the agent to stay near its initial position, while choosing the same direction for several timesteps might put the agent in a new situation.

There are several other disadvantages to choosing random movement. The probability that an agent encounters an old situation (one it has already encountered), can be very high, increasing the required number of timesteps for learning different situations. There is also no guarantee that the predictor will be able to succesfully predict the result of a random movement.

Nevertheless, interesting results have been obtained by combining this approach with e.g. chaotic search [42, 43] or confidence [44].

5-3-2 Minimise prediction error

For the autonomous agent equipped with a predictor, more information than just input and output data is available. Since the predictor produces a predicted output $\tilde{y}_t = f(\varphi_{t-1})$ each timestep, the error $\epsilon_t = y_t - \tilde{y}_t$ can also be used. As discussed in Section 3-4, the square of these errors is used in determining the quality of the model. This can also be used for the controller: if a negative reward is used proportional to the prediction error, it is expected that a controller that maximises this reward, will minimise the future total error. The controller then cooperates with the predictor by using the optimal solution u_t^* to the problem:

$$\min_U = \sum_{i=1}^{T} ||\epsilon_i(u_t)||^2, \tag{5-2}$$

with $U \subseteq \mathbb{R}^m$ the set of possible control values u. The error is written as a function of the control here, since this chapter focuses on the choice of control only, while the choice of parameters was discussed in the previous chapter.

Now, the main problem is that the choice of u_t can only influence the errors of future timesteps, not of previous timesteps: $\epsilon_i(u_t) = \epsilon_i$, for $i \leq t$. But the only known errors are those of previous timesteps: the errors of future timesteps are unknown at timestep t. Several options for solving this problem will be examined in Section 5-4. For now, let us assume there are methods to deal with these future errors, then the objective function can be simplified by leaving out those timesteps that are not influenced by the control of timestep t:

$$\min_U \sum_{i=t+1}^{T} ||\epsilon_i(u_t)||^2. \tag{5-3}$$

The expected behaviour of an agent with a predictor and a controller that minimises the prediction error, is trivial. It is expected that only those actions are chosen that give the

lowest prediction error, which could be the actions that move the agent towards situations that have already been learned, or trivial situations like standing still or staring at a black screen. One goal of the main problem in this thesis would be solved, namely how to predict the next sensor data given the current sensor and actuator data. This is described in Section 5-2 as the *exploitation* part. The *exploration* part, however, would not be solved: the controller of the agent is not 'motivated' to choose actions that lead to new situations for the agent. This shows that both of these aspects are important, and the controller should aid the predictor by providing exploratory behaviour rather than just aiding the predictor by choosing actions that minimise the prediction error.

5-3-3 Maximise prediction error

One way to get exploratory behaviour is to reward the agent for going to situations where the prediction error is high. Since the predictor itself still minimises the prediction error, it is expected that a situation with a high prediction error is actually a new situation. The controller then uses the optimal solution u_t^* to the following problem:

$$\max_{U} \sum_{i=t+1}^{T} ||\epsilon_i(u_t)||^2. \tag{5-4}$$

This is exactly the opposite of the result from the previous section. The expected resulting behaviour is changed: instead of choosing the actions that are easiest for the predictor, the most unpredictable actions will be chosen. For example, if the predictor has learned what will happen when a robot moves to the left, the controller that maximises the prediction error might then choose to move to the right. The controller and the predictor are then adversaries, while the previous approach showed a cooperating controller and predictor.

It is expected that maximising the prediction error leads to exploratory behaviour. Nevertheless, several researchers [45, 46, 47] pointed out that there are some drawbacks: since the controller is actually an adversary for the predictor, it might find ways to prevent the predictor from learning. For example, it might provide such instable behaviour that the sensor data becomes very noisy. Or it might move the system towards a part of the environment that is noisy. If it is possible to move the system towards a situation where no learning is possible, this approach can cause practical problems. The controller and predictor should not be adversaries, but complement each other. There should be a balance between exploratory behaviour and predictability.

5-3-4 Homeokinesis

Recently [48], the problem of retaining a balance between exploratory behaviour and predictability has been tackled by backpropagating the prediction error through the sensorimotor loop to get the time-loop error TLE $_t$ of the loop (also called reconstruction error), and let the controller minimise this error instead. This process has been termed homeokinesis. If an inverse of the predictor f is defined as $g(\tilde{y}_{t+1}) = g(f(\varphi_{t-1}) = \varphi_{t-1}$[1] , the reconstruction error

[1]It is by no means clear that the predictor is invertible. However, an approximation of the inverse of the predictor could be constructed in various ways, for example by using a Taylor expansion or a similar model as the predictor itself, but with the inputs and outputs reversed.

could be seen as the difference between the reconstructed regressor and the real regressor: $\text{TLE}_t = \varphi_{t-1} - g(y_{t+1})$.

The homeokinetic controller then uses the optimal solution u_t^* to the following problem:

$$\min_U \sum_{i=t+1}^{T} ||\text{TLE}_i(u_t)||^2. \tag{5-5}$$

It is expected that an agent with this type of controller maintains a balance between exploratory behaviour and predictability. The reason for this is not trivial, but explaining it is beyond the scope of this thesis. And even with this balance, there are some drawbacks to using homeokinesis. The main drawback is the choice of the function g: inverting the prediction process is not at all a trivial process, especially with the complex predictor functions discussed in the previous chapter. Injectivity and surjectivity can not always be guaranteed, and if Taylor expansions are used, it is assumed that the predictor is differentiable. Even if it is, matrix invertibility can become an issue. An approach that maintains the same balance between exploratory behaviour and predictability, but without using an inverse predictor, would be highly beneficial for solving the problem of this thesis.

5-3-5 Maximise learning progress

Various researchers [47, 46] recognised the drawbacks of maximising the prediction error, and a new reward was defined: the learning progress LP . Intuitively, an agent should not just be driven towards situations where the prediction error is high, because there is no guarantee that the predictor can adapt to those situations. Rather, an agent should be driven towards situations where the predictor actually learns the relation between sensor and actuator data. These are the situations where the prediction error is decreasing. The learning progress is therefore defined as the decrease in prediction error. Still, several definitions exist [41], for example $\text{LP}_t = \epsilon_{t-1} - \epsilon_t$ or $\text{LP}_t = \epsilon_t - \epsilon'_t = f(\varphi_{t-1}, \theta_{t-1}) - f(\varphi_{t-1}, \theta_t)$, where ϵ' is the difference between real sensor data y_t and predicted sensor data *after* the parameters of the predictor function have been updated.

Within this approach, the controller uses the optimal solution u_t^* to the following problem:

$$\max_U \sum_{i=t+1}^{T} ||\text{LP}_i(u_t)||^2. \tag{5-6}$$

There are two situations in which the reward for this controller is low. One is a situation where the predictor can already achieve a low prediction error, and no further improvement is possible. The second is a situation where the predictor might give a large error, but also cannot adapt in such a way that the error becomes lower. Preventing both learned and unlearnable situations, it is expected that the controller moves the agent towards situations where the predictor can be improved. Like in the approach of homeokinesis, this results in a balance between exploratory and predictable behaviour, but without the need of computing or approximating the inverse of the predictor.

Although there are quite some advantages to this technique, one drawback follows from the definition of the learning progress: alternating between unlearnable situations and situations

that have already been learned, can still result in a large learning progress. This might prevent the controller from moving towards situations where the predictor can learn. A solution to this drawback exists, and will be explained in the next section.

5-3-6 Intelligent Adaptive Curiosity

To prevent an agent that maximises the learning progress from alternating between complex unlearnable and simple learned situations, a solution has been suggested by Kaplan and Oudeyer [49]. This method has been called Intelligent Adaptive Curiosity (IAC). The idea is to divide the predictor into several experts, which are all specialised in predicting the future sensor data given the current sensor and actuator data *for a specific situation*. If the learning progress is then defined as the decrease in errors for a specific situation, it is expected that maximising the learning progress will not result in alternating between two different situations, since this does not necessarily give a large learning progress.

The learning progress for situation s is defined as $\mathrm{LP}^s = \epsilon^s_{t-1} - \epsilon^s_t$, where ϵ^s_t is the error at timestep t of expert s, and ϵ^s_{t-1} is the error of expert s at the previous timestep where the situation was specific to expert s.

The manner in which the predictor is decomposed into several experts, will not be described in this section. But the way the control is chosen, is as follows: for each timestep i, choosing control u_i would result in a specific situation $s(u_i)$, for which there is only one expert. In this situation, there will also be a learning progress. The control that would result in the highest learning progress, is chosen. This gives the following problem:

$$\max_U \sum_{i=t+1}^{T} ||\mathrm{LP}^{s(u_i)}(u_t)||^2. \tag{5-7}$$

As soon as one expert learned to predict correctly what happens in a specific situation, the prediction error will stay low in that situation. This implies that the learning progress of this situation is also low, even when alternating with an unpredictable situation, and the control that leads to this situation will not be chosen anymore. Similarly, unpredictable situations, where the prediction error can not be decreased by the predictor, will give a low learning progress, even when alternating with predictable situations. The expected result is that the agent will move towards those situations where the error can decrease, instead of moving towards learned or unlearnable situations, or alternating between the two.

5-4 Optimising with future data

In this chapter, several control types that influence the behaviour of the agent have been proposed. As is usual in the reinforcement learning framework, these controllers were maximising the future reward, giving a problem of the form

$$\max_U \sum_{i=t+1}^{T} R_i(u_t), \tag{5-8}$$

where $R_i(u_t)$ is the reward at timestep i if u_t would be chosen at timestep t. For example, in the approach of minimising the prediction error, we have $R_i = -\epsilon_i$. As stated in Section 5-3-2, the problem is that future rewards are unknown, even when choosing a u_t, so it is impossible to know which u_t would maximise the sum of rewards. Besides this, if T is infinite, the sum might not be bounded. For infinite T, this can be solved by including a discount factor γ, giving the problem:

$$\max_{U} \sum_{i=t+1}^{T} \gamma^{i-t} R_i(u_t). \tag{5-9}$$

Several algorithms deal with this kind of objective function [40]. However, most of these algorithms are not quite fit for this thesis. Many are only fit for discrete environments, discrete action spaces, require external knowledge not available in this thesis, or require too much computation time. Therefore, the approach in this thesis is to take a simple approximation of the future reward, by using previous rewards. The upper bound T is chosen to be equal to $t+1$, until some final timestep $\hat{T}$, making sure that only the reward of the next timestep needs to be computed. Then an approximation of the next reward can just be chosen as $R_{t+1}(u_t) = R_{t'}(u_t)$ or $R_{t+1}(u_t) = R_{t'}(u_t) + (R_{t'}(u_t) - R_{(t-1)'})$. Here, $R_{t'}(u_t)$ is the reward of the last timestep where u_t was chosen, and $R_{(t-1)'}(u_t)$ is the reward of the last timestep before t' where u_t was chosen. Alternatively, higher-order approximations can be used, like polynomial extrapolation, or another predictor could be used just for the rewards.

The above approximation of the future reward requires a drive to try new control values, since it looks only at past control values. This can be done using random control values with a certain probability, or in the case of continuous control values, by adding some random value to the chosen control value.

The use of approximations for the future rewards is justified by the goal we are trying to achieve. The goal is not to find the control that is an exact optimal solution of the optimisation problem (5-8), but rather to find a control that will cause a behaviour of the agent that aids the prediction process, both in accurately predicting the next sensor data and by making sure different situations will occur. The predictions should be exact, but the control will mainly determine the data that will be encountered by the predictor. As long as the general behaviour is as desired, like preventing situations where the predictor has already learned to predict correctly, and preventing unlearnable situations, the controller has achieved its main goal. It is expected that even when using approximations of the future rewards, the general behaviour that emerges, is similar to the expected behaviours described in Sections 5-3-2 to 5-3-6.

Chapter 6

Experiments

6-1 State-space dynamics

Combining the results of chapter 3 about system identification and chapter 5 about control for system identification, some techniques from these chapters are combined and tested on a mobile robot. As stated in section 2-2, the robot has visual sensors and differential drive dynamics. Let u be the control input of the robot, x the state of the robot ($[x_1, x_2]$ is the position and x_3 the orientation), and y the visual sensor output. Then the robot dynamics can be described as a state-space model:

$$\begin{aligned} \dot{x} &= \begin{bmatrix} \frac{1}{2}r\cos(x_3) & \frac{1}{2}r\cos(x_3) \\ \frac{1}{2}r\sin(x_3) & \frac{1}{2}r\sin(x_3) \\ \frac{r}{L} & -\frac{r}{L} \end{bmatrix} u, \\ y &= h(x), \end{aligned} \tag{6-1}$$

where r is the wheel radius of the robot, L the distance between wheels, and the function h represents the relation between the state of the robot and the visual sensor data. Using system identification techniques from chapter 3 to find the unknown parameters r and L, and unknown function h, with the aid of control techniques from chapter 5, this could lead to having a description of the relation between actuator and sensor data of the agent. This would answer the main research question of this thesis. However, one major problem is the function h: the relation between the position and orientation of the robot and the visual sensor data is very complex. This function should contain all information about the environment, about the way the visual sensors are attached to the robot, about what happens to the visual data when the robot performs a specific movement, 3D-information about objects, etc. This task is too complicated for all of the algorithms discussed in this thesis.

Besides this problem, the state is not known to the robot. The state would have to be estimated from the sensor data, which leads to similar problems as learning the function h. For these reasons, the state-space model (6-1) is not used in the application of this thesis.

6-2 Sensor space dynamics

Instead of looking at the dynamics of the position and orientation of the robot, the state-space dynamics, it is possible to look at the dynamics in the sensor space. Instead of using a state-space description like

$$\begin{aligned} \dot{x} &= g(x,u), \\ y &= h(x), \end{aligned} \tag{6-2}$$

the following equation can be used:

$$\dot{y} = h(y,u). \tag{6-3}$$

This could both be seen as a state-space model with the sensor data as state and no output, or as a first-order input-output (NARX) model.

As an example, suppose the robot has a 3 by 3 pixel camera with brightness values a to i. Suppose also that the robot moves to the left until all pixels are shifted one pixel to the right (not considering the depth of the image), and three new pixels appear:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \longrightarrow \begin{bmatrix} j & a & b \\ k & d & e \\ l & g & h \end{bmatrix}.$$

If the sensor data is written in vector form, this gives the following relation:

$$[a\ b\ c\ d\ e\ f\ g\ h\ i]^T \longrightarrow [j\ a\ b\ k\ d\ e\ l\ g\ h]^T.$$

This transformation can be realised by a matrix multiplication and vector addition:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{bmatrix} + \begin{bmatrix} j \\ 0 \\ 0 \\ k \\ 0 \\ 0 \\ l \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} j \\ a \\ b \\ k \\ d \\ e \\ l \\ g \\ h \end{bmatrix}. \tag{6-4}$$

This shows that for a specific situation, the dynamics in sensor space should not be too hard to learn, since they are even linear in the above example. No relation between position and sensor data is required.

Of course, the function h is a complex nonlinear function that cannot be described by a single linear transformation. For different control values u of the robot, the mapping will be different. But at different locations in the environment the mapping might also be different due to the object depth, noise, shadow and lighting, higher-order dynamics like inertia and slipping, etc. But around a local point $(\hat{y}, \hat{u})$, the function can be approximated by a linear transformation.

6-3 Choice of predictor

The above observation leads to the insight to use the Local Linear model (LLM)from Section 3-3-2, where the global model is a weighted sum of local linear models, and each local model is valid only in a specific regime of the input space. Using this model, it is expected that a local model that is more valid in a regime of the input space where the control moves the robot to the left, will have larger values in the diagonals below the main diagonal. Similarly, local models that are more valid in a regime of the input space where the control moves the robot to the right, should have larger values in the diagonals above the main diagonal. The goal is to let the predictor learn these models from the given data, and the goal of the controller is to make sure several of these situations actually occur so that the predictor gets the opportunity to learn them.

Table 6-1 shows the properties of the predictor. The chosen predictor function is the LLM:

$$f(\varphi) = \sum_{k=1}^{\tilde{n}} \kappa(\beta_k||\varphi - \gamma_k||)(\mu_k\varphi + \nu_k).$$

The choice of regression vector and method of parameter estimation will be motivated in the remainder of this section.

Table 6-1: Chosen properties of the predictor.

Regression vector	$\varphi_{t-1} = (u_{t-1}, y_{t-1})$
Predictor function	LLM
Method of parameter estimation	RPM + Linear least squares

6-3-1 Choice of regression vector

Section 3-2 of this thesis focuses on the choice of the regression vector. For the application in this chapter, the choice has been made to use the sensor and actuator data of the last timestep to predict the sensor data of the following timestep. This corresponds to an input-output model of the NARX type. This chapter already explained why a state-space model for the robot dynamics is not used, and why the sensor dynamics are considered instead.

It would be possible to use data of previous timesteps, however. For example we could choose $\varphi_{t-1} = (y_{t-1}, y_{t-2}, \dots y_{t-p}, u_t, u_{t-1}, \dots u_{t-q})$ for some p, q. However, the sensor data consists of images, which is high-dimensional data. Using many images can give computational problems. Since the actuator data for the mobile robot is low-dimensional, it is actually possible to use the actuator data of several timesteps without increasing the computation time too much. Still, the choice has been made to use only the actuator data of the last timestep, since the experiments are simple enough to not take higher-order dynamics like inertia into account.

This is also related to the Markov property from probability theory, where future states of a process only depend on the present state, not the whole past. In the application of this thesis, the assumption is that the sensor data of the next timestep depends only on the current sensor and actuator data. No higher-order dynamics is taken into account, and it is assumed

that the robot cannot change the environment for example by moving objects. For the simple experiments conducted in this thesis, these assumptions should be fairly accurate.

The motivation for the choice of regression vector, as well as an overview of alternatives, is shown in Figure 6-1.

6-3-2 Choice of predictor function

Section 3-3 showed several possibilities for the choice of the predictor function. In this chapter, the motivation for choosing the LLM was explained. Around a specific point in the input space, the sensor dynamics can be approximated by a linear model. This increases the number of parameters compared to the other models described in this thesis. In all other models of Section 3-3, we have the vectors $\alpha_k \in \mathbb{R}^n$, for $k = 1, \dots \tilde{n}$. But for the LLM, parameters $\alpha_k \in \mathbb{R}^{n \times l}$ are matrices. With more parameters, it is expected that this model will have better accuracy, at the cost of more computation time. Therefore, the model order $\tilde{n}$ can be lower than when using other predictor functions. This was discussed in Chapter 4.

The motivation for the choice of predictor function, as well as an overview of alternatives, is shown in Figure 6-2.

6-3-3 Choice of parameters

To prevent having to use an iterative nonlinear least squares algorithm for optimising over all parameters θ, the parameters β_k, γ_k will be chosen randomly from a continuous probability distribution, and stay fixed. If the optimisation is then performed over parameters α_k only, the problem is reduced to a linear least squares problem as described in Section 3-4-3. Using the Moore-Penrose pseudo-inverse, the parameters α_k will not only be optimal in the least-squares sense, but will also have minimum norm. Section 3-4-3 further describes how this is combined with the random choice of the other parameters.

Fixing the parameters β_k, γ_k has the consequence that optimising the parameters becomes a least-squares problem. But choosing the parameters randomly β_k, γ_k has the advantage that no prior knowledge is necessary: as soon as the robot 'wakes up' or starts working, it will not have knowledge of the environment or its actuator or sensor data, so choosing the parameters randomly is the only option there is. Tuning these parameters afterwards is possible, but then the parameters α_k would have to be tuned over and over again every time the other parameters are adapted. This is a possibility, but not the one chosen in this thesis.

In Chapter 4, it was shown that the LLM can indeed be trained by optimising over parameters α_k only, letting parameters β_k, γ_k be fixed randomly. This approach was called Shuffled Linear Model (SLM), and parameter estimation of the model is reduced to a linear least squares problem. The accuracy of the model was also analysed theoretically, with proofs about the approximation capabilities of the model and an upper bound on the model order $\tilde{n}$.

The motivation for the choice of parameters, as well as an overview of alternatives, is shown in Figure 6-3.

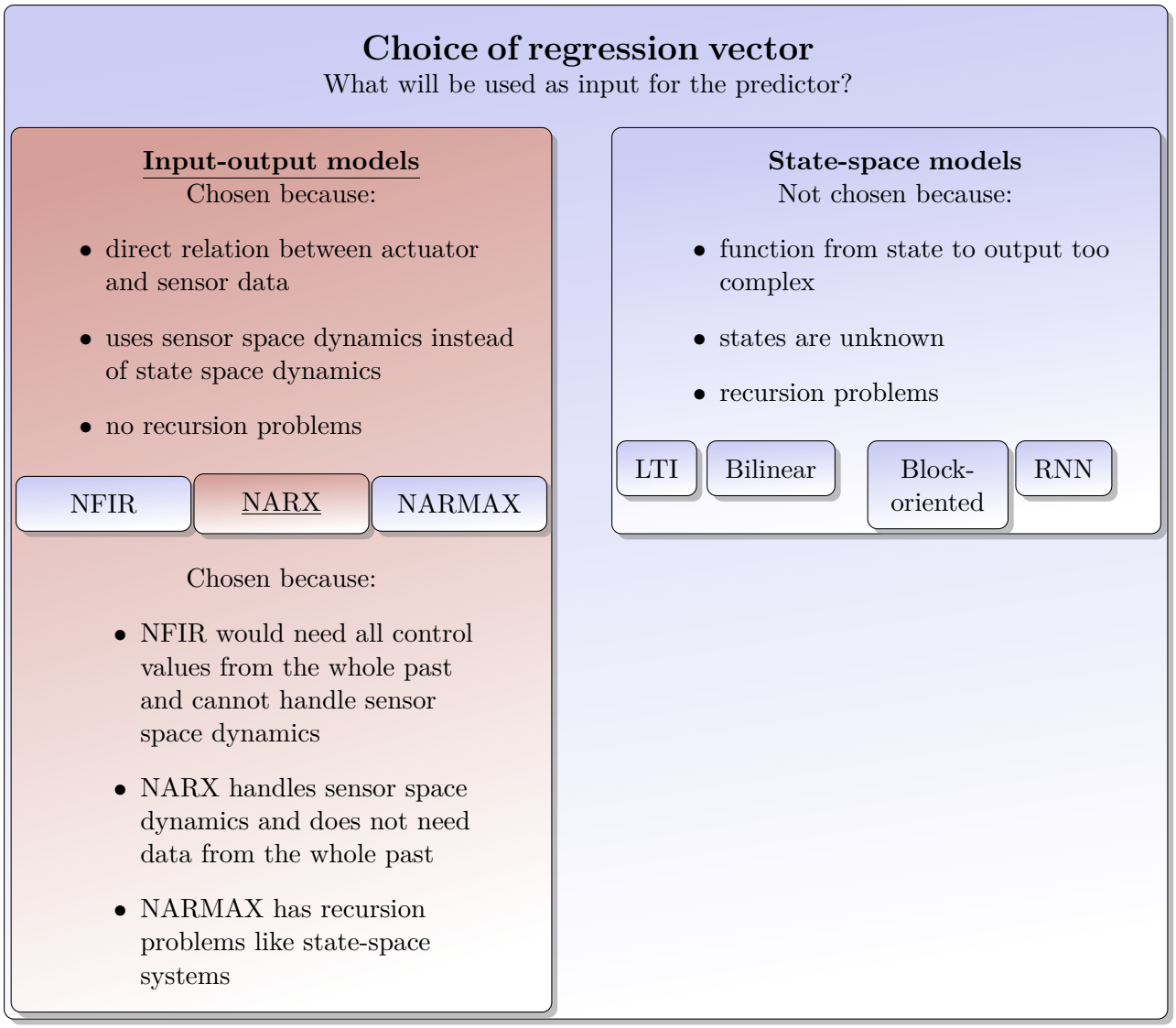


Figure 6-1: Motivation for the choice of regression vector, and alternatives that were discussed in this thesis.

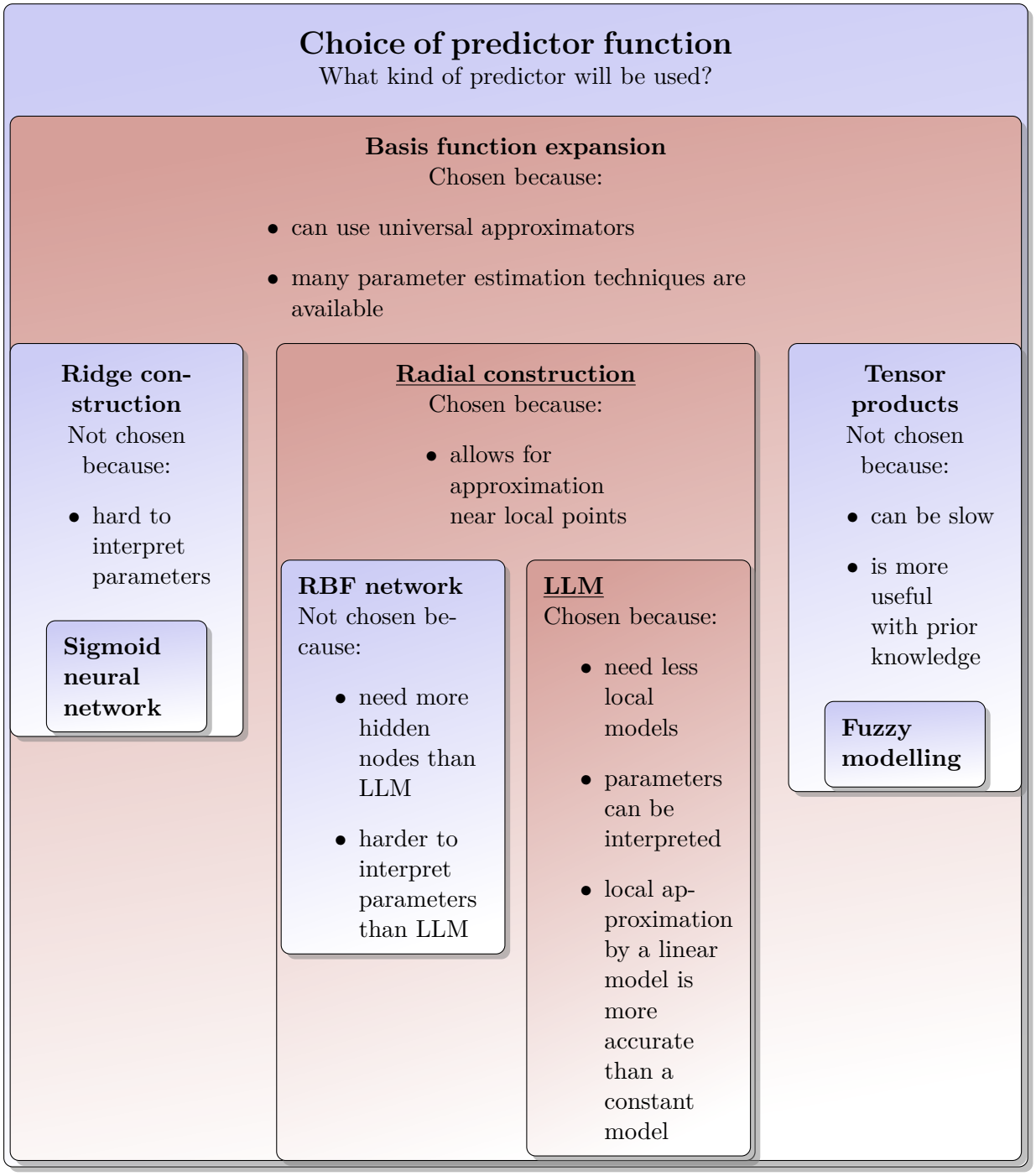


Figure 6-2: Motivation for the choice of predictor function, and alternatives that were discussed in this thesis.

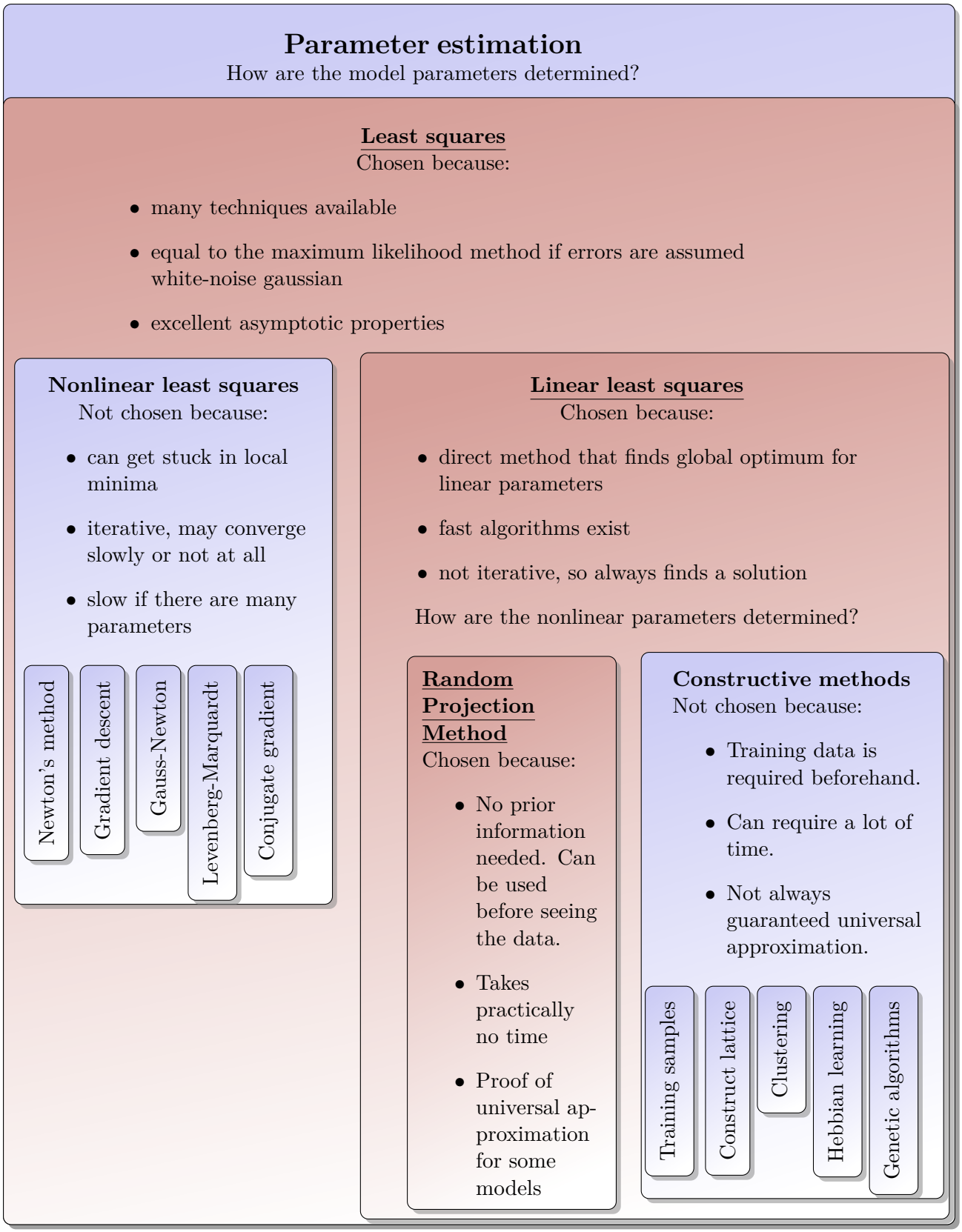


Figure 6-3: Motivation for the choice of parameter estimation, and alternatives that were discussed in this thesis.

6-4 Choice of control

In Chapter 5, various types of controllers that maximised a certain type of internal reward are discussed. The expected behaviour of an agent equipped with these controllers, was different for each controller, and each had their merits and drawbacks. For example, minimising the prediction error with both the predictor and controller, normally leads to homeostatic behaviour like standing still or performing the same movement over and over again. The desired behaviour is exploratory behaviour: the controller has to ensure that as much new sensor and actuator data is provided that the predictor can handle.

The approach of maximising the learning progress from Section 5-3-5, was the simplest approach analysed in this thesis to maintain a balance between exploratory and predictable behaviour. It turns out that, because of the simplicity of the experiments, the drawbacks of this approach are not a big problem. It is not possible for the robot to move back and forth between complex unpredictable and simple predicted situations, so it is not necessary to implement the approaches of homeokinesis and IAC. Still, it would be very interesting to investigate these approaches in further research, especially since they could combine very well with the SLM[1]. Besides the method of maximising the learning progress, the even simpler approaches of maximising and minimising the prediction error were investigated for comparison.

In all chosen control techniques, there is a probability of choosing a random control action. This prevents the robot from choosing the same action over and over again, while a different action might actually give a higher reward. Furthermore, when a control action has been chosen, it is repeated for 5 timesteps, to prevent the robot from switching between speeds too fast. The controller looks 50 timesteps in the past to determine the control of the current timestep, by choosing the control of the timestep that resulted in maximum reward, together with some small white Gaussian noise to further stimulate exploration of new control values.

6-5 Experimental set-up and specifications

The mobile robot in the experiment is a Romo 1.0 equipped with an Android smartphone. The robot communicates with a laptop with an 2.8 GHz processor via Wifi and a server. Camera images are sent from smartphone to laptop, and the control commands are sent from laptop to smartphone, all via a server and using the message system ØMQ[2]. The smartphone then sends the control commands to the robot via an audio signal. This has been developed by DO bots[3].

Although the robot can send 640×480 JPEG images to the computer, these images are scaled down to 16×16 pixels to decrease computation time and memory usage. The robot can move

[1] The approaches of homeokinesis and IAC require inverse properties of the predictor or a partition of the predictor in several experts. Since the SLM uses linear models, inverse properties could be analysed more easily than for other black-box models. Furthermore, the combination of the linear models is done in such a way that each model can be seen as an expert for a specific range of situations that is determined by the Gaussian function, which is what is needed for the IAC approach. Further research is necessary to see how the SLM combines with the approaches of homeokinesis or IAC.

[2] See www.zeromq.org.

[3] See www.dobots.nl.

forward and backward and can take turns, but the control is limited to one dimension to make the experiment simpler: the robot is limited to rotating left or right at different speeds, with speed 0 equal to standing still. Images and controls are received and sent to the robot at a frequency of 2 Hz.

The robot is put in a box with images on the inside, and the relation between actuator and sensor data is to be learned for the situation that the robot rotates at different speeds and directions inside the box. To test the chosen techniques for more than one situation, a fish-eye lens is equipped to the smartphone camera to provide a 360° view. The experiment is conducted both with and without a fish-eye lens. The set-up for both cases is shown in Figures 6-4 and 6-5.

Figure 6-4: Experimental set-up. The robot is equipped with a smartphone and has to learn the relation between actuator and sensor data by rotating and looking at pictures inside a box.

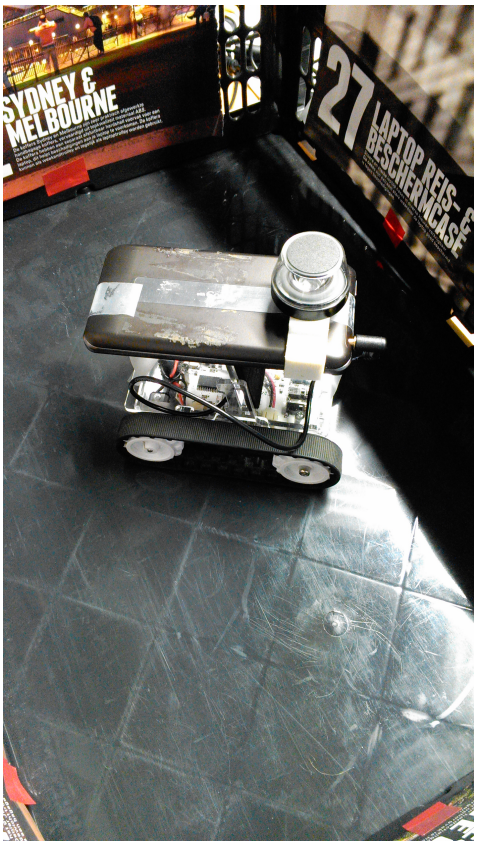


Figure 6-5: Using a fish-eye lens to give the robot a 360° view.

6-6 Model specifications

The SLM model is used (see previous section and Chapter 4) with a number of $\tilde{n} = 50$ local models. Though a larger number of local models would give a more accurate model, it would also require more computational power, so this number was chosen to give decent accuracy

while not requiring too much computational power[4]. The parameters β_k are drawn from a uniform distribution over $(0.45, 0.55)$, while parameters γ_k are drawn from a uniform distibution with different ranges depending on the dimension. For the dimensions corresponding with the previous image y_{t-1}, the elements of γ_k were drawn from a $(0, 1)$ uniform distribution, and for the dimension corresponding with the previous control u_{t-1}, this element of γ_k was drawn from a $(-1, 1)$ uniform distribution. The reason for this is that the images consisted of pixel brightness values scaled to $(0, 1)$, and the control was a value scaled to $(-1, 1)$, where $u = -1$ corresponded with rotating left at full speed, and $u = 1$ with rotating right at full speed, and $u = 0$ with standing still. So there actually is some prior knowledge available to the robot: the range of its input values.

In the calculation of the construction matrix K from Section 3-4-3 and from Equation (4-12), that is, when calculating the outputs of the radial basis functions, one problem has to be solved. Since the dimension n of the image data y_{t-1} is much higher than the dimension m of the previous control u_{t-1}, the Gaussians could be dominated by the images, while the control would have no significant effect on the model. To prevent this, the image data has been scaled by a factor c that controls the importance of image data compared to the control. This has been done only when calculating the output of the Gaussian functions, not in the local models themselves. Since the Euclidean norm $||\varphi - \gamma_k||$ is used as an input to the Gaussian functions, the value of c for which image and control data are equally important is $c = 2\sqrt{m/n}$, where the factor 2 comes from the different input ranges for image and control data.

Ideally, learning should be done online. But this would require the computation of a pseudo-inverse at every timestep, which is not possible at the required speed and with the dimensions used. But offline learning is also not possible, since the controller requires a prediction error so that it can maximise its reward. Therefore, learning has been done in *phases*: after several timesteps, the robot is stopped and learning takes place using all past data, then the robot starts moving again without learning. The number of timesteps between learning phases is 25, and the final learning phase determines the final parameters of the learned model.

6-7 Testing

Only one of the subquestions of this thesis has not been answered yet: **How can we test whether an agent has learnt the relation between actuator and sensor data?** This chapter described which techniques will be used to predict the next sensor data given the current sensor and actuator data. It was also described which techniques will be used for the controller to aid in this process. Assuming the techniques work correctly, this will result in a model that can predict the next sensor data correctly. What is left is some means to test the quality of the model, and of the techniques that resulted in the model. These will be tested on various fronts:

- **Pre-learning total error**: the total error was defined in Section 3-1. Using the model parameters from each timestep, the error is computed at each timestep.

[4] In fact, this thesis provided an upper bound for the number of models necessary to give a good approximation. Using this upper bound would require too much computational power, so this theoretical result is not used in the experiment.

- **Post-learning total error**: after the last learning phase, the optimal parameters have been computed. Using these parameters, the total error is recomputed for each timestep to see the performance with the new parameters if the robot would be in the exact same situations it has already encountered. This can be compared with the pre-learning total error to see how much the agent has learned during the experiment.
- **Change detection**: after the last learning phase, a part of the environment will be changed. It is checked whether the error of the robot will be higher in this situation.
- **External influence detection**: after the last learning phase, the robot control will be changed in such a way that the real movement of the robot does not correspond with the control values it receives. It is checked whether the error of the robot in this situation will be higher.
- **Interpretability**: most models discussed in this thesis are black-box models, that require no prior knowledge but also are hard to interpret. However, the SLM that is used, does have a physical interpretation. It will be checked whether the learned model has a logical physical explanation.

Together with the results from the next section, this should answer the last subquestion of this thesis.

6-8 Results

6-8-1 Error and chosen control actions

Table 6-2: Results of one run of the three tested controllers for one simulation: minimise prediction error, maximise prediction error, and maximise learning progress.

Chosen controller	Pre-learning error	Post-learning error
Minimise prediction error	20.0759	1.4611
Maximise prediction error	24.1857	4.6256
Maximise learning progress	19.6401	3.5219

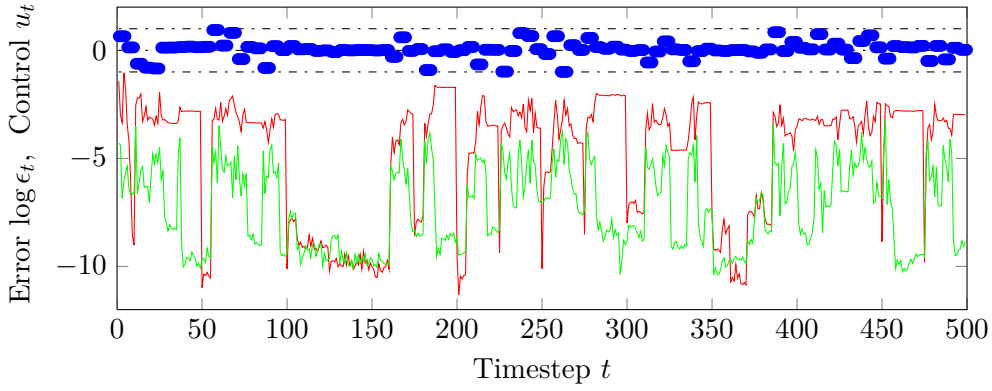
Table 6-2 shows the total error of one run for each of the three tested controllers. The errors of each timestep have been divided by the input dimension before they were summed over each timestep. A large difference can be seen between the total error before learning, and the total error after the last learning phase. This means that, if the robot would encounter the exact same situation after learning, it would predict its sensor data better than when it first encountered that situation.

Since these results are only from one run, no precise conclusions can be drawn. Still, the different controllers can be compared by looking at Figure 6-6. In this figure, both the chosen control actions and the prediction error have been displayed. As explained in Chapter 5, the method of minimising the prediction error resulted mostly in the trivial action of standing still, with some other control actions chosen randomly with a certain probability, and this

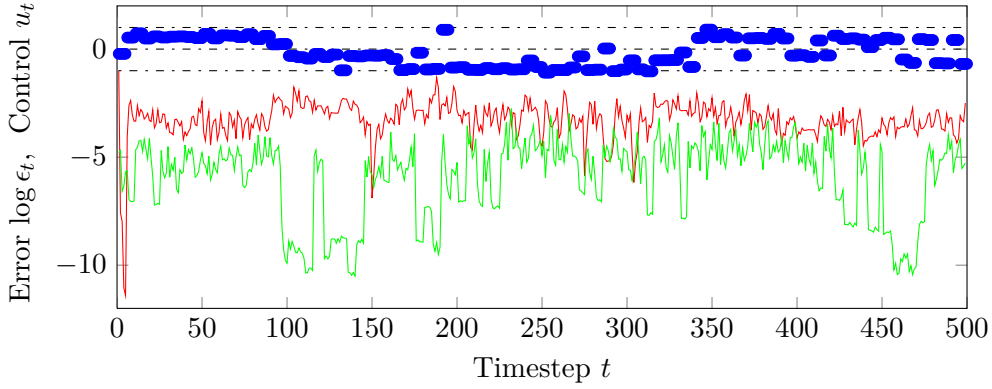
happened for most of the simulations. It can also be seen that the error was often lower than for the other controllers.

The controllers where the prediction error or the learning progress was maximised, showed exploratory behaviour, as explained in Chapter 5. Although the controllers could get 'stuck' on one chosen control action, because this action gave the highest reward, there is still a diversity in the chosen control actions. This diversity occurred in most of the simulations, though not in all of them. This disadvantage was already explained in Chapter 5 and could be solved by choosing a more complex type of controller like the IAC method explained in Section 5-3-6.

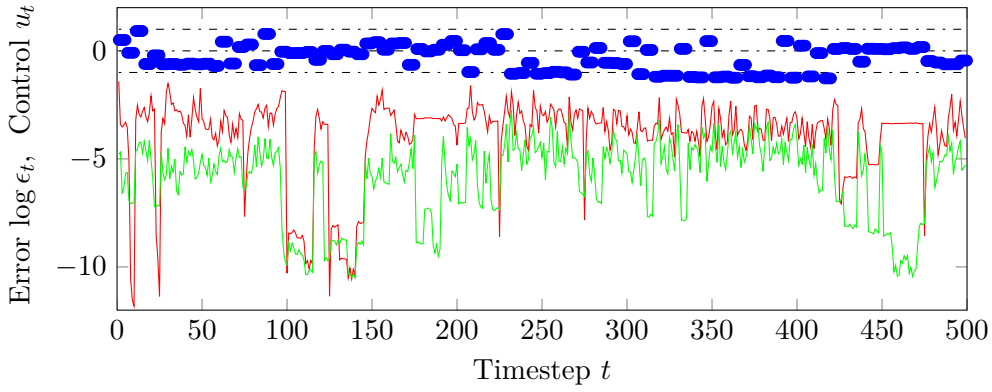
Figure 6-6 shows that the model could give a low error for some situations with all three controllers, but not for every situation. Probably more training data would be required, since 500 timesteps is not really that much, and the training data was also not that 'clean' (robot moving a bit from its place or against walls, communication problems, changes in lighting, moving objects, etc.). Still, the accuracy of the model with only a small amount of noisy data is remarkable.



(a)



(b)



(c)

Figure 6-6: Results of one run of the three tested control strategies: **(a)** minimise prediction error, **(b)** maximise prediction error, and **(c)** maximise learning progress. The blue dots represent the chosen control at timestep t, with the black lines showing the intervals $[-1, 0]$ and $[0, 1]$. The red graph shows the logarithm of the prediction error at timestep t before the learning algorithm included that timestep, while the green graph shows the logarithm of the prediction error at timestep t using the learned parameters from the last timestep.

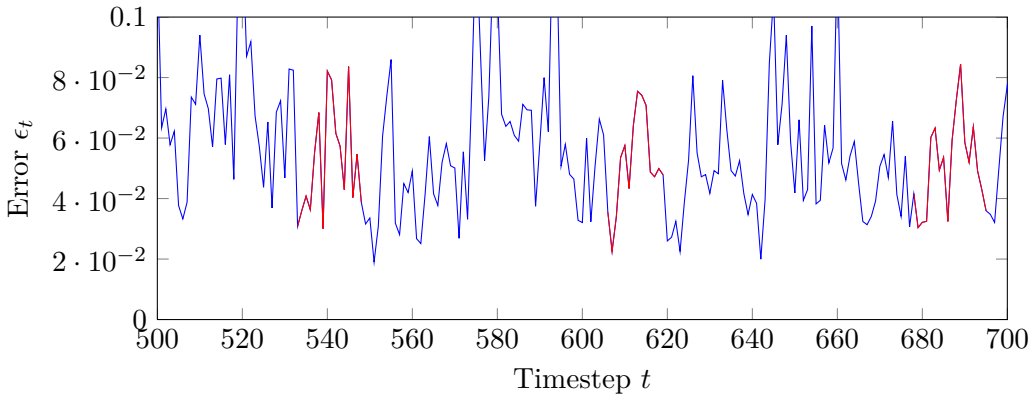
6-8-2 Change detection

To further show the accuracy of the learned model, and to test the practical use of it, another experiment has been done using the model learned during the experiment that was explained in this chapter. With the final parameters following from the last learning phase, it was checked whether a change in the robot's environment could be detected. One of the pictures in the environment of the robot was removed to introduce a different picture. Then the robot was put in the box, and the control actions were chosen such that the robot rotated in one direction at a constant speed, while still predicting the next sensor data but without learning. After the change in the environment was seen three times, this experiment was terminated. Figure 6-7 shows the prediction error during the experiment, with the timesteps where the robot was looking at a changed part of the environment highlighted. These timesteps were found manually.

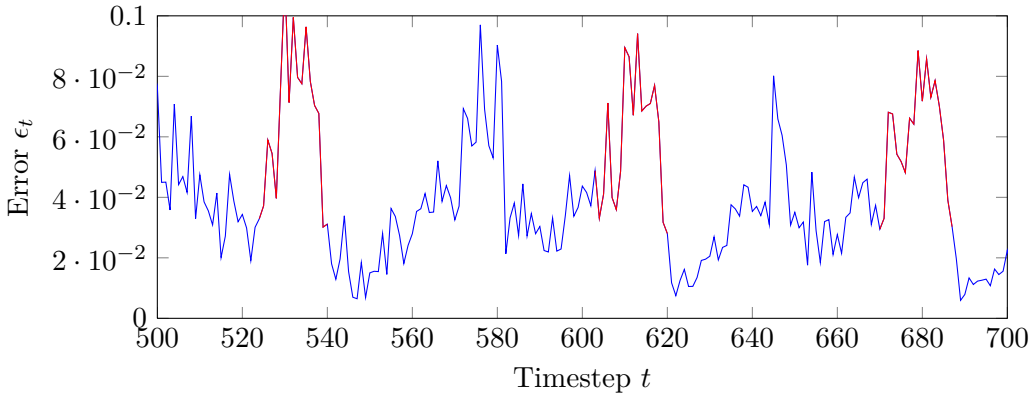
Note that the fish-eye lens was not used in this experiment, since this would make the robot see the changed part of the environment at every timestep, making it impossible to distinguish between the prediction errors for unchanged and for changed parts of the environment.

The results of the approach where the prediction error is minimised, shown in Figure 6-7a, is not very convincing. Other parts of the environment resulted in a higher error than the part where the environment was changed. This is probably due to the fact that when learning, the robot was standing still most of the time, so it has not really learned what happens in most parts of the environment. The other two approaches, where the prediction error and the learning progress were maximised, show more promising results (see Figures 6-7b and 6-7c). Although some parts of the environment were apparently not fully learned due to the small number of training samples when learning, the model still manages to give a distinguishing higher error when the robot looks at a part of the environment that is different from what it has learned.

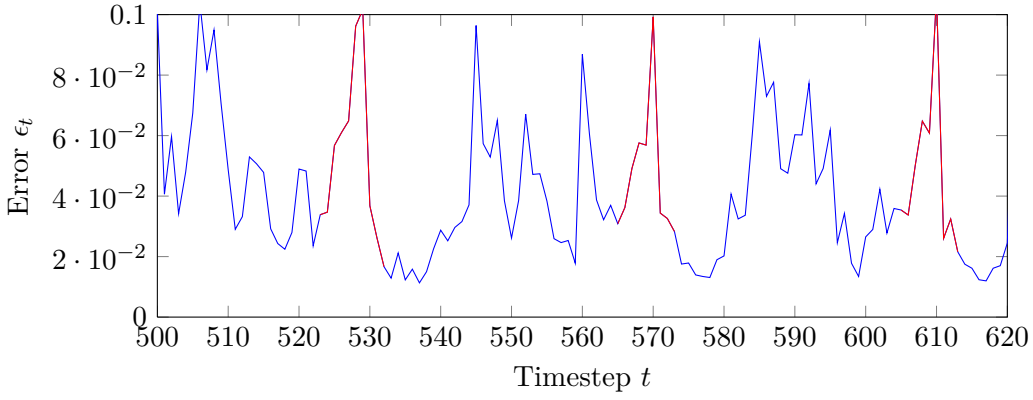
The results of the experiment show that the robot and the chosen techniques can not really be used directly in a practical application. The results are not satisfying enough for that. But they do show that there is a lot of potential for further development. Detecting a change in the environment, while the environment has been learned autonomously, could very much be of practical use in, for example, robotic surveillance. This experiment showed that it is possible to do so with the chosen techniques, at least for a simple experiment, but possibly for more complex cases as well.



(a)



(b)



(c)

Figure 6-7: Results of the change detection experiment for: **(a)** minimise prediction error, **(b)** maximise prediction error, and **(c)** maximise learning progress. The graph shows the prediction error during the whole experiment, with the red parts of the graph highlighting those timesteps where the robot looked at a part of the environment that was different from the first experiment where learning took place. In the change detection experiment, no learning took place.

6-8-3 External influence detection

The goal of the third and last experiment was to see if the robot can detect a change in its own dynamics, not in the environment. A similar goal is to distinguish between the robot's own actions and the actions of a different entity. The first idea was to let the robot rotate and then hold it still manually at some timesteps. This would result in a difference between the expected sensor data (different from last timestep due to expected moving) and the real sensor data (same as last timestep due to standing still). Then it would be checked whether the prediction error would be higher at those timesteps where the robot was meant to rotate but could not because it was prevented to do so. Unfortunately, this experiment did not lead to satisfying results. The reason for this is probably that the robot could not perfectly predict changes in sensor data, and in these cases standing still lead to a result that is even easier to predict than the sensor data resulting from the chosen control value.

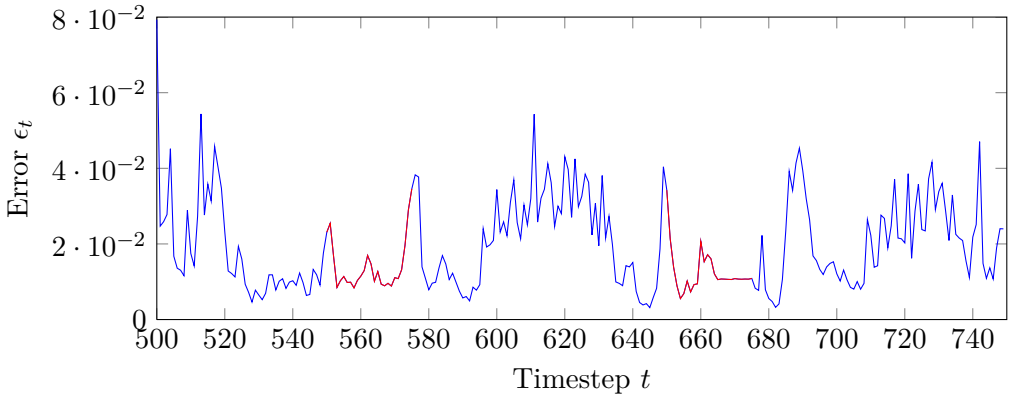
After these negative results, the nature of the experiment was changed. Instead of holding the robot manually, a different 'virtual' control value was given to externally influence the robot not by hand, but by making use of the control mechanism. The robot was put in the box and set to rotate at a constant speed. At a certain point, the control value used to control the robot was reversed, making the robot rotate in the opposite direction. But at the same time, the control value given to the predictor was *not* changed, causing a difference between the expected movement and the real movement. The predictor would expect the sensor data to be changed in the same way as before, while the sensor data actually changed in a different way because of the change in the control value.

Figure 6-8 shows the results of this experiment for the normal lens. The results for the fish-eye lens are shown in Figure 6-9. In both figures, the prediction error is plotted for each timestep, with highlighted parts for those timesteps where the control is changed. As in the change detection experiment, the case where the prediction error is minimised (Figure 6-9a) does not show good results. It is expected that the prediction error is higher for the highlighted parts, where the control value used by the model is different from the movement of the robot. This is not the case, probably because not every situation has been learned by the robot in this case, because the robot mainly stood still when learning. What is interesting is that the results improve when the robot is equipped with a fish-eye lens, probably because the robot could see its whole environment during learning, even if it mostly stood still.

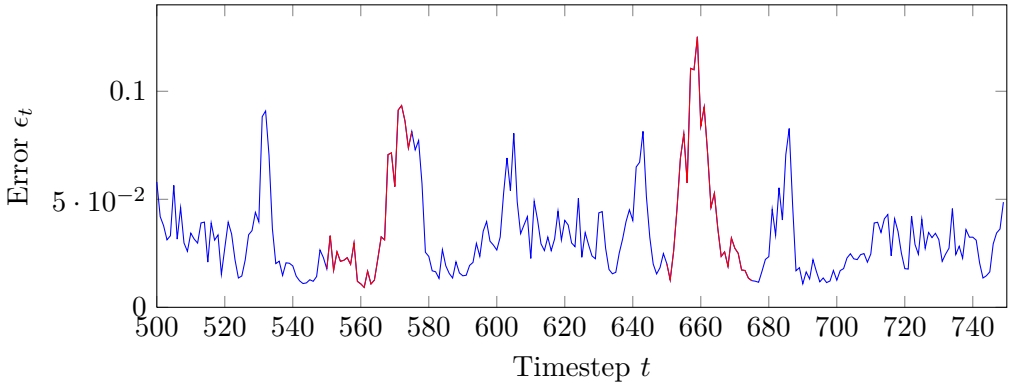
The other two approaches, where the prediction error or the learning progress were maximised, show a peak in the prediction error for those timesteps where the control values used by the model are different from the movement of the robot. This is as expected, and even though there are other situations where the error shows a peak, the results indicate that there is potential to let the robot distinguish between its own actions and someone else's. A potential application for this would, again, be the setting of robotic surveillance. After learning its own dynamics, the robot might be able to detect actions that were performed by other entities, for example an intruder that opens a door. Automatically detecting such events can be of great practical use.

Note that the results improve a lot when using the fish-eye lens. Where the results for the standard lens also show peaks in the prediction error for other situations than the highlighted parts of the graphs, the peaks for the fish-eye lens are more distinctive. One reason for this is that the robot could see the whole environment all the time when using the fish-eye lens,

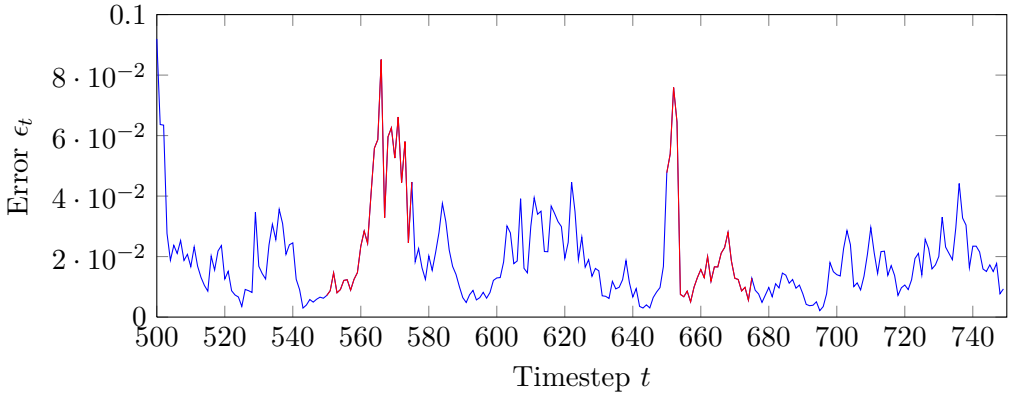
making all situations more similar than when the standard lens was used. Another reason is that the fish-eye lens gives a circular image with the whole environment in it. With only 16×16 pixels, this means that no details of the environment can be observed, and the part of the image outside the circle is black. Sensor data contains less detailed information, so changes in the dynamics have a large effect on the sensor data, whereas the sensor data for the normal lens is more sensitive to changes in the environment. This shows that both lenses have their own strengths and weaknesses, but more research is required to give more information about the differences.



(a)

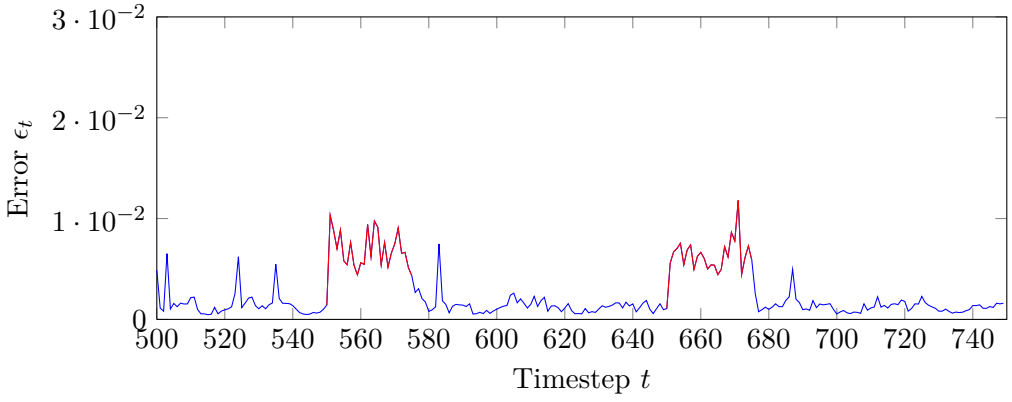


(b)

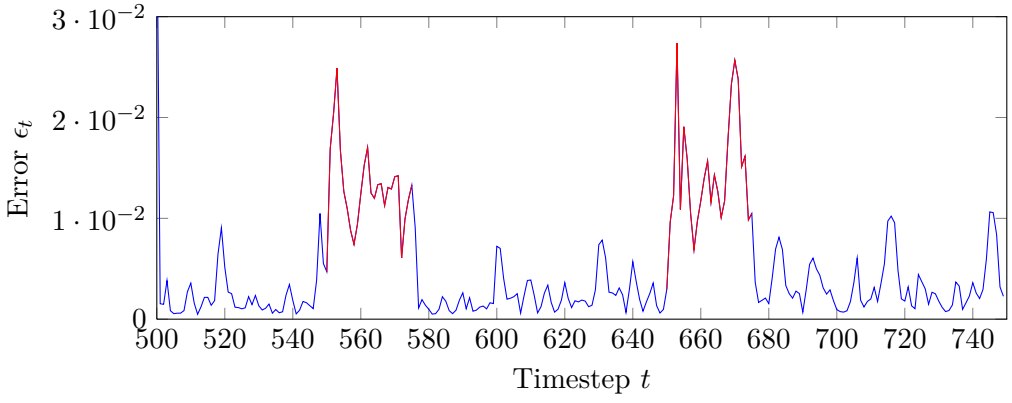


(c)

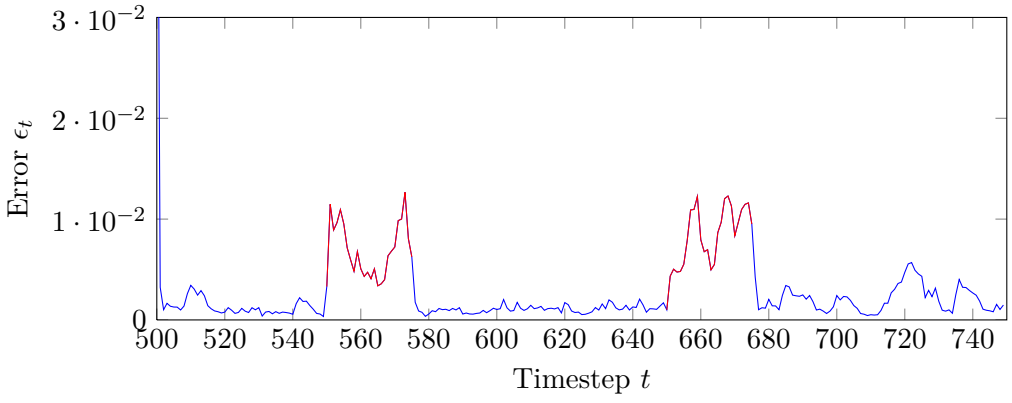
Figure 6-8: Results of the external influence detection experiment for: **(a)** minimise prediction error, **(b)** maximise prediction error, and **(c)** maximise learning progress. The graph shows the prediction error during the whole experiment, with the red parts of the graph highlighting those timesteps where the real movement of the robot was different from the control values given to the model. In this experiment, no learning took place.



(a)



(b)



(c)

Figure 6-9: Results for the same experiment as in the previous figure, except that the robot was equipped with a fish-eye lens here.

6-8-4 Interpretability

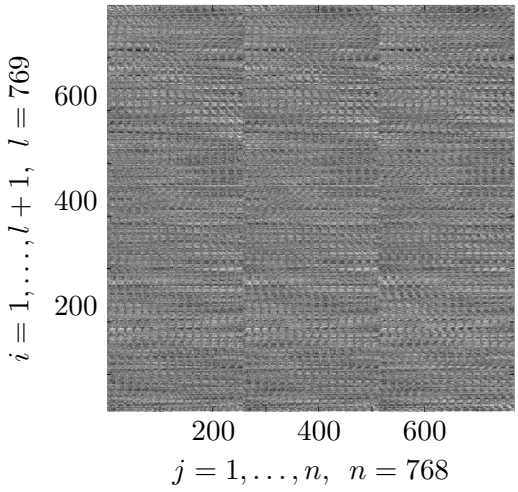
One advantage of the used model, as explained in Section 3-3-2, is the interpretability: whereas neural networks and other blackbox models do not give any insight into the model, but only produce the desired output from a given input, the SLM that was used in these experiments does have a physical interpretation. This is due to the parameters α_k being a function of the input, rather than a value independent of the input. Looking at the predictor function $f(\varphi) = \sum_{k=1}^{\tilde{n}} \alpha_k \kappa(\varphi, \beta_k, \gamma_k)$ from Section 3-3, the function values $\kappa(\varphi, \beta_k, \gamma_k)$ represent the validity of linear model k for input φ, while parameters of the linear model appear inside α_k.

In Chapter 4 we saw that for each linear model k, the parameters α_k can be represented by an $n \times (l+1)$ matrix λ_k. Since this matrix represents the parameters of a linear model, it is expected that it contains information that is physically interpretable. For example, a linear model that is most valid when the robot stands still, should have a matrix of parameters that is similar to the identity matrix. The models that are most valid when the robot is rotating, should have matrices similar to the one described in Equation (6-4).

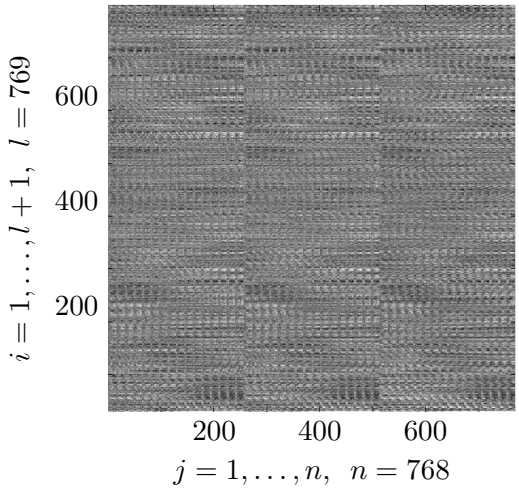
Figure 6-10 shows the parameters of the linear models for three local models: one that is most active when the robot stands still, one that is most active when the robot rotates left, and one that is most active when the robot rotates right. We also zoomed in on the last image. Lighter parts of the images represent higher values of the matrix elements. These kind of matrices appeared most of the time for all tested controllers, although sometimes the matrices were too vague to see the desired structure. Like in Equation (6-4), the matrices consist of dashed diagonals, corresponding to the transformation that is done on the images. The reason that the matrix corresponding to rotating right has diagonals below the main diagonal, while in Equation (6-4) this was the case for rotating left, is the different matrix multiplication used in the model (the matrix is used in a right multiplication, not a left multiplication).

When using the fish-eye lens described in Section 6-5, the resulting parameter matrices had a different structure. One example, corresponding to the robot rotating left, can be found in Figure 6-11. This shows that different types of transformations can be learned with the chosen model.

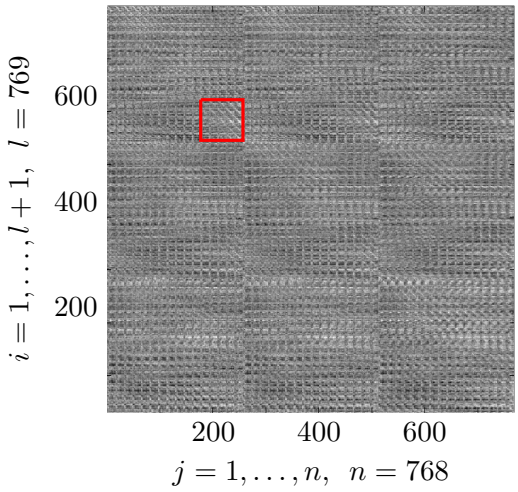
The shown matrices correspond with some kind of knowledge of the relation between actuator and sensor data. Depending on the actuator data and the sensor data, the relation can locally be described by a linear model, resulting in a global model that is nonlinear. This gives an answer to the main research question of this thesis: **How can an autonomous physical agent learn the relation between its actuator and sensor data?**



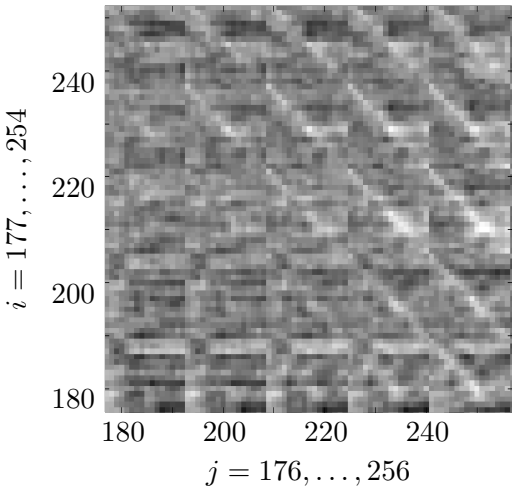
(a) Parameters of $\lambda_k, \ k = 22$



(b) Parameters of $\lambda_k, \ k = 6$



(c) Parameters of $\lambda_k, \ k = 41$



(d) Zoomed part of (c).

Figure 6-10: Learned parameters of linear models $\alpha_k(\varphi) = \lambda[\varphi^T \ 1]^T$ corresponding to three different situations: **(a)** stand still (22th local model), **(b)** rotate left (6th local model), and **(c)** rotate right (41st local model). In **(d)**, we zoomed in on the highlighted part of (c) so that the matrix structure can be seen more clearly. The parameters are put in a matrix, where each row corresponds to the input dimension $+1$ (the bottom row corresponds to the bias, the one above to the one-dimensional motor space, and all the other rows correspond to one of the $16 \times 16 \times 3$ pixels). Each column corresponds to the output dimension, which has dimension $16 \times 16 \times 3 = 768$. The element of the matrix on position i, j corresponds to the linear relation between input and output data for input dimension i and output dimension j. There were $\tilde{n}$ of these matrices, one for each linear model.

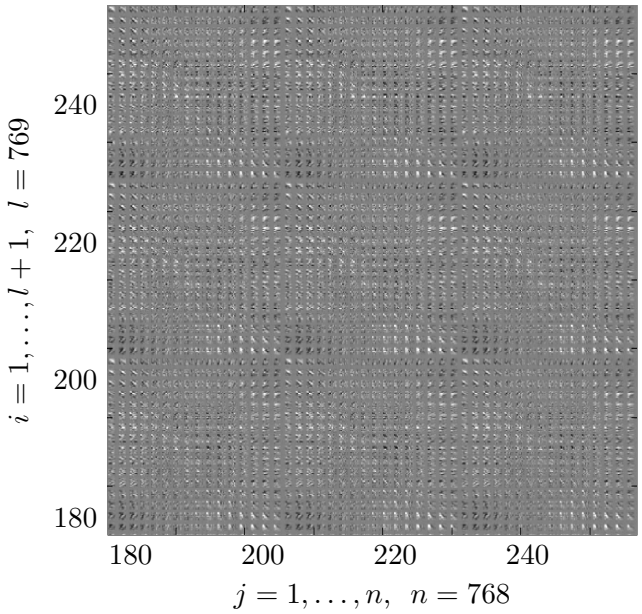


Figure 6-11: Learned parameters of λ_{17}, which constitute a linear model for the situation that the robot rotates left. Here, the robot was equipped with a fish-eye lens.

Chapter 7

Conclusion

There exist several techniques that can help an agent to learn the relation between its actuator and sensor data autonomously. Several of these techniques were discussed. The chosen technique is the Prediction Error Method, with a Local Linear Model as a predictor function. The model uses some randomly initialised parameters, while the other parameters follow from a linear least squares problem. The resulting model is a novel model that has been given the name Shuffled Linear Model (SLM).

The SLM has many benefits. First of all, since efficient algorithms exist for linear regressoin, parameter estimation can be done in a computationally efficient manner. Although some parameters were chosen randomly, it was proved that the SLM can still act as a universal approximator. The required number of models was shown to be less than the required number of hidden neurons for similar algorithms. An interpretation of the parameters was also given, something that can be quite hard for other so-called 'black-box' models.

Besides the theoretical results about the SLM, the model has been tested on a mobile robot, equipped with a smartphone. The goal was to let the robot learn the relation between actuator and sensor data. The SLM was used as a predictor, with the current sensor and actuator data as input, and the sensor data of the next timestep as desired output. The robot was put in a box with images and could rotate, in order to learn its own dynamics (in sensor space) and the environment. When given enough training samples, the robot indeed learned a mapping between input and output correctly, as could be seen from the parameters of the SLM and from the result of the different tests.

Two practical tests were used to evaluate the use of the model. One was the test of change detection: after fitting the model, one image of the box was replaced by a new image. It turns out that the robot could 'notice' the change by giving a higher prediction error when looking at the new image. The other test was to check external influences: the movement of the robot was changed while keeping the same control value. This caused a difference between the movement of the robot and the control value of the predictor, increasing the prediction error in these situations. A result is that the robot could potentially learn the difference between the effects of its own actions, and the effect of someone else's actions.

Besides proposing a new type of model and using it as a predictor, one contribution of this thesis was to combine this model with a controller that uses only internal information as a reward. The controller that maximised intrinsic reward made sure that the agent would autonomously explore those parts of its environment and sensorimotor space where it had not yet learned correctly how to predict its sensor data. The combination of a powerful prediction model and a controller that could provide exploratory agent behaviour, lead to an agent learning its sensor dynamics and its environment autonomously. For robots with visual sensors, this combination has not been analysed before.

Many suggestions remain for future research. The suggested model has been analysed in some aspects, but many questions remain. For example, what is the role of normalisation in the radial basis functions, and what is the relation to linearised models of nonlinear systems? The control part of this thesis also has room for improvement: the used controller has both drawbacks and solutions that were already mentioned in this thesis, and the approach of Intelligent Adaptive Curiosity or the approach of Homeokinesis have not yet been combined with the model proposed in this thesis. Both of these approaches could be well combined with the SLM, since this model is a combination of several models and might also have properties about the inverse relation between sensor and actuator data.

The practical part of this research also has a lot of room for improvement. The experiments that were presented, were only simple ones, and the chosen techniques were not being used at their full potential. To make a real practical application, there is a lot to be improved in, for example, the data provided to the model and the implementation of the controller techniques. And with different assumptions, for example one where the environment is dynamic, the model would have to be changed.

The model predicts only one timestep in the future. This is limiting for practical applications, and multi-step prediction is another suggestion for future research. One could also look at the results of combining the chosen techniques not just into one, but in multiple agents. Distributed control or distributed sensing could then have a new topic for research. Finally, combining the techniques used in this thesis, with different intrinsic rewards (like empowerment), could lead to a robot that could truly learn to navigate in the real world.

Appendix A

Link between linear state-space and input-output models

A linear time-invariant (LTI) system can be written in state-space form as:

$$\Sigma \left\{ \begin{array}{lcl} x_{t+1} & = & Ax_t + Bu_t \\ \hat{y}_t & = & Cx_t \end{array} \right. . \tag{A-1}$$

Analysis of discrete-time LTI systems is typically done by looking at the impulse response or by looking at the system in the Z-transform domain. The impulse response h_t of a system is defined as the zero-state response of the system to input

$$u_t = \delta_t = \left\{ \begin{array}{ll} u_t = 1, & t = 0 \\ u_t = 0, & t \neq 0 \end{array} \right. .$$

For the LTI system, this gives:

$$\begin{array}{ll} x_0 = 0, & \hat{y}_0 = 0 \\ x_1 = Ax_0 + B\delta_0 = B, & \hat{y}_1 = Cx_1 = CB \\ x_2 = Ax_1 + B\delta_1 = AB, & \hat{y}_2 = Cx_2 = CAB \\ x_3 = Ax_2 + B\delta_2 = A^2B, & \hat{y}_3 = Cx_3 = CA^2B \\ \vdots & \vdots \\ x_t = A^{t-1}B, & h_t = \hat{y}_t = CA^{t-1}B. \end{array}$$

From the linearity and time-invariance of Σ, it follows that the impulse response $h_t = CA^{t-1}B$ is a complete characterization of the system: $\hat{y}_t = \sum_{k=1}^{\infty} h_k u_{t-k}$.

A common notation is the shift operator $qu_t = u_{t+1}$ [8]. Using this notation, the system can be written as:

$$\begin{aligned} \hat{y}_t &= \sum_{k=1}^{\infty} h_k u_{t-k} \\ &= \sum_{k=1}^{\infty} h_k (q^{-k} u_t) \\ &= [\sum_{k=1}^{\infty} h_k q^{-k}] u_t \\ &= G(q) u_t, \end{aligned} \tag{A-2}$$

with $G(q) = \sum_{k=1}^{\infty} h_k q^{-k}$ the transfer operator. Note that this description is an input-output model, since there is no dependence of $\hat{y}$ on the state, but only on the past inputs. Using this notation, the system $\hat{y}_t = G(q)u_t$ can be generalised to systems of the following form:

$$A(q)y_t = \frac{B(q)}{F(q)} u_t + \frac{C(q)}{D(q)} \epsilon_t, \tag{A-3}$$

with $\epsilon = \hat{y} - y$ the prediction error. This form includes all of the linear models of Section 3-2-1 (FIR, ARX, and ARMAX) and the Box-Jenkins model (BJ) and Output-Error model (OE) models, closing the gap between input-output models and state-space models.

The Z-transform is defined as: $\mathcal{Z}\{x_t\} = \sum_{t=-\infty}^{\infty} x_t z^{-t} = X(z)$. Using the linearity ($\mathcal{Z}\{ax_{1_t} + bx_{2t}\} = aX_1(z) + bX_2(z)$) and time-shift ($\mathcal{Z}\{x_{t+k}\} = z^k X(z)$) properties of the Z-transform, the system Σ above can be transformed as follows:

$$zX(z) = AX(z) + BU(z) \tag{A-4}$$

$$\Rightarrow X(z) = (zI - A)^{-1} BU(z), \tag{A-5}$$

$$\hat{Y}(z) = CX(z) \tag{A-6}$$

$$= C(zI - A)^{-1} BU(z), \tag{A-7}$$

where X, $\hat{Y}$, and U are Z-transforms of x, $\hat{y}$ and u respectively. This is also similar to the input-output models. Besides showing the connection between input-output and state-space models, the impulse response and the Z-domain are not considered further in this thesis.

Appendix B

SLM questions

At the end of Chapter 4, some further research questions were posed. Two of them will be answered in this appendix.

B-1 Shuffled Nonlinear Models

One question that arises from Chapter 4, is about the use of the linear model. Even around a local operating point, it can be the case that a linear model does not correctly represent the relation between inputs and outputs, so it might be better to look at local nonlinear models. Can other models be used in the same way as the linear models in the SLM approach?

An answer to this question can be found when looking at the proof of the lemmas and theorems of Section 4-4. Although it was proved that the model could be trained using linear regression, nowhere in the proof was the linear relation between the local model outputs $\alpha_k(\varphi)$ and the regression vector φ used. What was used, was the linear relation between local model outputs $\alpha_k(\varphi)$ and its parameters.

For local model $\alpha_k(\varphi) = \mu_k(\varphi) + \nu_k$, let $\lambda_k = [\mu_k \ \nu_k]$ and $z(\varphi) = [\varphi^T \ 1]^T$. Then we can write $\alpha_k(\varphi, \lambda_k) = \lambda_k z(\varphi)$. In Equation (4-10), the linear relation between α_k and λ_k was used to show that the model can be trained using linear regression. But since the linear dependence on φ was not used anywhere, we have the following conjecture:

Corollary 2. *Theorem 2 holds for any function $z(\varphi) : \mathbb{R}^l \to \mathbb{R}^c$, for some constant $c \in \mathbb{N}$, if the factor $l + 1$ in the theorem is changed to $\max(c, t)$.*

Indeed, if any function $z(\varphi) : \mathbb{R}^l \to \mathbb{R}^c$ is chosen, the factor $l + 1$ should be changed to c in all the proofs of Section 4-4. The proofs should then still be valid, unless $c \geq t$ is chosen, because there has to be at least 1 model.

An example of a different local model that could be used is the bilinear model from Section 3-2-2. In this case, $z(\varphi) = [(\varphi \otimes \varphi)^T \ \varphi^T \ 1]^T$, and $c = l^2 + l + 1$. The non-fixed parameters λ

consist of matrices for the constant, linear and bilinear part, which have 1, l, and l^2 columns respectively. The matrix K from the SLM algorithm is then a $t \times (l^2 + l + 1)\tilde{n}$-matrix. This should further decrease the required number of models. This does not necessarily mean that local models with higher order are always better, since the condition for Theorem 2 should still hold. This condition states that the matrix $Z = [z_1 \dots z_t]^T$ should have full rank.

However, it is reasonable to assume that less models are needed if the local models are more complex. Indeed, if $c = t$, then one local model should suffice for achieving zero error with all t datapoints.

One interesting result is that, since SLM models themselves are also linear in their parameters, they can also be used as a local model, adding a recursive aspect to the theory. This *recursive SLM* could have some kind of hierarchical structure, making it more related to deep learning algorithms [50]. A recursive SLM could be defined as follows:

$$SLM(\varphi) = \sum_{k=1}^{\tilde{n}} \kappa_k(\varphi)(\mu_k \varphi + \nu_k) = \sum_{k=1}^{\tilde{n}} \kappa_k(\varphi)\lambda_k z(\varphi), \tag{B-1}$$

$$recSLM(\varphi) = \sum_{k=1}^{\tilde{n}} \kappa_k(\varphi) SLM(\varphi). \tag{B-2}$$

If this recursion is repeated Q times, the result is a model of the form

$$\begin{aligned} rec^Q SLM(\varphi) &= \sum_{k=1}^{\tilde{n}} \kappa_k(\varphi) \sum_{k^{(2)}=1}^{\tilde{n}^{(2)}} \kappa_{k^{(2)}}(\varphi) \cdots \sum_{k^{(Q)}=1}^{\tilde{n}^{(Q)}} \kappa_{k^{(2)}}(\varphi)\lambda_{k,k^{(2)},\dots,k^{(Q)}}(\mu_k \varphi + \nu_k) \\ &= \sum_{k=1}^{\tilde{n}} \kappa_k(\varphi)\tilde{\lambda}_k z(\varphi), \end{aligned} \tag{B-3}$$

with

$$\tilde{\lambda}_k = [\lambda_{k,1,1,\dots,1} \dots \lambda_{k,\tilde{n}^{(2)},\tilde{n}^{(3)},\dots,\tilde{n}^{(Q)}}], \tag{B-4}$$

$$z(\varphi) = [\kappa_1(\varphi) \dots \kappa_1(\varphi)[\varphi^T \ 1]^T \dots \kappa_{\tilde{n}^{(2)}}(\varphi) \dots \kappa_{\tilde{n}^{(Q)}}(\varphi)[\varphi^T \ 1]^T]. \tag{B-5}$$

In this case, $\tilde{\lambda}_k$ is an $n \times (\prod_{i=2}^{Q} \tilde{n}^{(i)})$-matrix, giving a total of $n \times \tilde{n} \prod_{i=2}^{Q} \tilde{n}^{(i)}$ parameters. Linear regression can still be applied, since the model has the same form as the SLM. But it might also be possible to find more efficient learning algorithms for this model, if the recursive aspect is somehow exploited.

More research is required to investigate all these claims.

B-2 SLM kernel

One suggestion for further research could be the link between SLMs and kernel methods. A *kernel* is defined as a function $k : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$, and is usually interpreted as a measure of similarity between two datapoints. These measures can be used, among others, to classify or cluster datapoints, or to extract features from high-dimensional data.

Two popular kernels are the linear kernel $k(\varphi_1, \varphi_2) = \varphi_1^T \varphi_2$ and the Gaussian kernel $k(\varphi_1, \varphi_2) = e^{-b||\varphi_1 - \varphi_2||^2}$. The basis functions from Section 3-3 can also be used as a kernel. For basis function κ, the kernel can be defined as $k(\varphi_1, \varphi_2) = \frac{1}{\tilde{n}} \kappa(\varphi_1)\kappa(\varphi_2)$, where $\tilde{n}$ is the number of basis functions. These kind of kernels have been analysed for neural networks with a sigmoid activation function, and for RBF networks [51].

In this section it will be shown that the SLM can also be used as a kernel, and that under some conditions, the result is equivalent to the composite kernel

$$k(\varphi_1, \varphi_2) = (\varphi_1^T \varphi_2 + 1) e^{-b||\varphi_1 - \varphi_2||^2}.$$

This will be called the local linear kernel in this section.

Recall the function $SLM(\varphi) = \sum_{k=1}^{\tilde{n}} \kappa_k(\varphi)(\mu_k \varphi + \nu_k)$, with $\kappa_k(\varphi) = e^{-\beta_k ||\varphi - \gamma_k||^2}$. In the proof of Lemma 1, it was shown that this can be written as a basis function expansion:

$$SLM(\varphi) = \sum_{k=1}^{\tilde{n}} \lambda_k \kappa_k(\varphi) z(\varphi), \tag{B-6}$$

with $\lambda_k = [\mu_k \ \nu_k]$ and $z(\varphi) = [\varphi^T \ 1]^T$. If the vector $\varphi \in \mathbb{R}^l$ is extended with a fixed element $\varphi^{(l+1)} = 1$, this can be denoted as:

$$SLM(\varphi) = \sum_{k=1}^{\tilde{n}} \sum_{j=1}^{l+1} \lambda_k^{(j)} \kappa_k(\varphi) \varphi^{(j)}, \tag{B-7}$$

where (j) denotes the j-th element of a vector, or the j-th column of a matrix. Since the parameters $\lambda_k^{(j)}$ correspond to the basis coefficients, which can be seen as output weights of the model, the term $\sum_{j=1}^{l+1} \kappa_k(\varphi)\varphi^{(j)}$ can be seen as a new basis function. The proposed SLM kernel then takes the form: $k(\varphi_1, \varphi_2) = \sum_{j=1}^{l+1} \kappa_k(\varphi_1)\varphi_1^{(j)} \kappa_k(\varphi_2)\varphi_2^{(j)}$.

Since $\kappa_k(\varphi)$ depends on randomly chosen parameters, the function values $k(\varphi_1, \varphi_2)$ can be seen as random variables. For $k(\varphi_1, \varphi_2) = \kappa_k(\varphi_1)\kappa_k(\varphi_2)$, it has already been shown [51] that the expectation is equal to the RBF kernel under some assumptions[1]. This result can be generalised to the SLM kernel. Under the same assumptions as those of the RBF kernel, let $f(\gamma) = \frac{1}{(2\pi\sigma^2)^{l/2}} e^{-\frac{1}{2\sigma^2}||\gamma||^2}$ be the probability distribution of parameters γ_k. As $\sigma \to \infty$, we have:

$$\begin{aligned}
E[k(\varphi_1, \varphi_2)] &= \int_{\mathbb{R}^l} \sum_{j=1}^{l+1} \kappa(\varphi_1, \beta, \gamma)\varphi_1^{(j)} \kappa(\varphi_2, \beta, \gamma)\varphi_2^{(j)} f(\gamma) d\gamma \\
&= \sum_{j=1}^{l+1} \varphi_1^{(j)} \varphi_2^{(j)} \int_{\mathbb{R}^l} e^{-\beta||\varphi_1 - \gamma||^2} e^{-\beta||\varphi_2 - \gamma||^2} \frac{1}{(2\pi\sigma^2)^{l/2}} e^{-\frac{1}{2\sigma^2}||\gamma||^2} \\
&= \sum_{j=1}^{l+1} \varphi_1^{(j)} \varphi_2^{(j)} e^{-\tilde{\beta}||\varphi_1 - \varphi_2||^2} \\
&= (\varphi_1^T \varphi_2 + 1) e^{-\tilde{\beta}||\varphi_1 - \varphi_2||^2},
\end{aligned} \tag{B-8}$$

[1] The parameters γ_k are assumed to be normally distributed with zero mean, and the result holds as the variance tends to infinity. The distribution of the parameters β_k is not taken into account.

where the results for the RBF kernel have been used in the third step. So the proposed local linear kernel is $k(\varphi_1, \varphi_2) = (\varphi_1^T \varphi_2 + 1)e^{-\tilde{\beta}||\varphi_1 - \varphi_2||^2}$.

This result not only opens the way for the SLM to be used for problems like classification, clustering and dimension reduction, where many different kernel methods exist. It also leads to a new interpretation of the SLM. The SLM can be seen as a Monte Carlo approximation of the local linear kernel. Instead of using the whole continuous distribution of the weights γ_k, a finite number of Monte-Carlo samples are taken from this distribution. Another interpretation is that the SLM is an approximation in primal space of a kernel method using the local linear kernel. Many kernel methods look at the dual version of the same problem. Similar insights have been obtained recently for recurrent networks and recurrent kernels [52].

Bibliography

[1] D. M. Wolpert, Z. Ghahramani, and M. I. Jordan, "An internal model for sensorimotor integration," *Science*, vol. 269, pp. 1880–1882, 1995.

[2] S. S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall Englewood Cliffs, NJ, 2007.

[3] K.-i. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural networks*, vol. 6, no. 6, pp. 801–806, 1993.

[4] R. R. Mohler, *Bilinear Control Processes: With Applications to Engineering, Ecology and Medicine*. Academic Press, Inc., 1973.

[5] I. Leontaritis and S. A. Billings, "Input-output parametric models for non-linear systems part i: deterministic non-linear systems," *International journal of control*, vol. 41, no. 2, pp. 303–328, 1985.

[6] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky, "Nonlinear black-box modeling in system identification: a unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.

[7] P. Van Overschee and B. De Moor, "Subspace identification for linear systems: theory, implementation, applications," *status: published*, 1996.

[8] L. Ljung, "System identification: theory for the user," *PTR Prentice Hall, Upper Saddle River, NJ*, 1999.

[9] L. Ljung, "Prediction error estimation methods," *Circuits, Systems and Signal Processing*, vol. 21, no. 1, pp. 11–21, 2002.

[10] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[11] A. Censi and R. M. Murray, "Bootstrapping bilinear models of robotic sensorimotor cascades," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 4318–4325, IEEE, 2011.

[12] V. Verdult, *Non linear system identification: a state-space approach*. Twente University Press, 2002.

[13] S. Billings and S. Fakhouri, "Identification of a class of nonlinear systems using correlation analysis," *Electrical Engineers, Proceedings of the Institution of*, vol. 125, no. 7, pp. 691–697, 1978.

[14] K. Narendra and P. Gallman, "An iterative method for the identification of nonlinear systems using a hammerstein model," *Automatic Control, IEEE Transactions on*, vol. 11, no. 3, pp. 546–550, 1966.

[15] E.-W. Bai, "A blind approach to the hammerstein–wiener model identification," *Automatica*, vol. 38, no. 6, pp. 967–979, 2002.

[16] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: Unifying the algorithms and accelerating convergence," *Neural Networks, IEEE Transactions on*, vol. 11, no. 3, pp. 697–709, 2000.

[17] M. Lukoševičius and H. Jaeger, "Survey: Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[18] J. Stark, D. Broomhead, M. Davies, and J. Huke, "Takens embedding theorems for forced and stochastic systems," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 30, no. 9, pp. 5303–5314, 1997.

[19] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[20] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.

[21] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *Systems, Man and Cybernetics, IEEE Transactions on*, no. 1, pp. 116–132, 1985.

[22] J.-S. Jang and C.-T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *Neural Networks, IEEE Transactions on*, vol. 4, no. 1, pp. 156–159, 1993.

[23] R. Battiti, "First-and second-order methods for learning: between steepest descent and newton's method," *Neural computation*, vol. 4, no. 2, pp. 141–166, 1992.

[24] C. R. Rao and S. K. Mitra, "Generalized inverse of a matrix and its applications," *J. Wiley, New York*, 1971.

[25] T. Greville, "Some applications of the pseudoinverse of a matrix," *SIAM review*, vol. 2, no. 1, pp. 15–22, 1960.

[26] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[27] Y. Miche, B. Schrauwen, and A. Lendasse, "Machine learning techniques based on random projections.," in *ESANN*, 2010.

[28] K. Neumann, C. Emmerich, and J. J. Steil, "Regularization by intrinsic plasticity and its synergies with recurrence for random projection methods," 2012.

[29] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *Neural Networks, IEEE Transactions on*, vol. 17, no. 4, pp. 879–892, 2006.

[30] O. Nelles, *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer, 2001.

[31] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, no. 16, pp. 3056–3062, 2007.

[32] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *Neural Networks, IEEE Transactions on*, vol. 9, no. 1, pp. 224–229, 1998.

[33] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: four layers versus three," *Neural Networks, IEEE Transactions on*, vol. 8, no. 2, pp. 251–255, 1997.

[34] S. A. Billings, H.-L. Wei, and M. A. Balikhin, "Generalized multiscale radial basis function networks," *Neural Networks*, vol. 20, no. 10, pp. 1081–1094, 2007.

[35] F. Fernández-Navarro, C. Hervás-Martínez, J. Sanchez-Monedero, and P. A. Gutiérrez, "Melm-grbf: a modified version of the extreme learning machine for generalized radial basis function neural networks," *Neurocomputing*, vol. 74, no. 16, pp. 2502–2510, 2011.

[36] K. J. Hunt, R. Haas, and R. Murray-Smith, "Extending the functional equivalence of radial basis function networks and fuzzy inference systems," *Neural Networks, IEEE Transactions on*, vol. 7, no. 3, pp. 776–781, 1996.

[37] R. Murray-Smith and T. A. Johansen, "Multiple model approaches to nonlinear modeling and control," 1997.

[38] T. A. Johansen, R. Shorten, and R. Murray-Smith, "On the interpretation and identification of dynamic takagi-sugeno fuzzy models," *Fuzzy Systems, IEEE Transactions on*, vol. 8, no. 3, pp. 297–313, 2000.

[39] G. Baudat and F. Anouar, "Kernel-based methods and function approximation," in *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, vol. 2, pp. 1244–1249, IEEE, 2001.

[40] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. Cambridge Univ Press, 1998.

[41] P.-Y. Oudeyer and F. Kaplan, "What is intrinsic motivation? a typology of computational approaches," *Frontiers in Neurorobotics*, vol. 1, 2007.

[42] J. Tani, "Model-based learning for mobile robot navigation from the dynamical systems perspective," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 26, no. 3, pp. 421–436, 1996.

[43] J. Tani, "Proposal of chaotic steepest descent method for neural networks and analysis of their dynamics," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 75, no. 4, pp. 62–70, 1992.

[44] R. Saegusa, G. Metta, G. Sandini, and S. Sakka, "Active motor babbling for sensorimotor learning," in *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pp. 794–799, IEEE, 2009.

[45] J. Schmidhuber, "Adaptive confidence and adaptive curiosity," Citeseer, 1991.

[46] J. Schmidhuber, "Curious model-building control systems," in *In Proc. International Joint Conference on Neural Networks, Singapore*, Citeseer, 1991.

[47] F. Kaplan and P.-Y. Oudeyer, "Maximizing learning progress: an internal reward system for development," in *Embodied artificial intelligence*, pp. 259–270, Springer, 2004.

[48] R. Der and G. Martius, *The Playful Machine: Theoretical Foundation and Practical Realization of Self-Organizing Robots*, vol. 15. Springer-Verlag Berlin Heidelberg, 2012.

[49] P.-Y. Oudeyer, "Intelligent adaptive curiosity: a source of self-development," 2004.

[50] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep machine learning-a new frontier in artificial intelligence research [research frontier]," *Computational Intelligence Magazine, IEEE*, vol. 5, no. 4, pp. 13–18, 2010.

[51] C. K. Williams, "Computation with infinite neural networks," *Neural Computation*, vol. 10, no. 5, pp. 1203–1216, 1998.

[52] M. Hermans and B. Schrauwen, "Recurrent kernel machines: Computing with infinite echo state networks," *Neural Computation*, vol. 24, no. 1, pp. 104–133, 2012.

Glossary

List of Acronyms

DoF	degrees of freedom
PEM	Prediction Error Method
SIM	Subspace Identification Method
NFIR	Nonlinear Finite Impulse-Response model
FIR	Finite Impulse-Response model
NARX	Nonlinear AutoRegressive eXogenous model
ARX	AutoRegressive eXogenous model
NARMAX	Nonlinear AutoRegressive Moving Average model with eXogenous input
ARMAX	AutoRegressive Moving Average model with eXogenous input
LTI	linear time-invariant
BJ	Box-Jenkins model
OE	Output-Error model
RNN	Recurrent Neural Network
LLM	Local Linear model
ELM	Extreme Learning Machine
RC	Reservoir Computing
RPM	Random Projection Method
RBF	Radial Basis Function
IAC	Intelligent Adaptive Curiosity

SLM Shuffled Linear Model

TSM Takagi-Sugeno Model

List of Symbols

α_k	Basis function coefficients ($\alpha_k \in \mathbb{R}^n$).
β_k	Dilation parameters ($\beta_k \in \mathbb{R}^l$ or $\beta_k \in \mathbb{R}$).
γ_k	Translation parameters ($\gamma_k \in \mathbb{R}^l$ or $\gamma_k \in \mathbb{R}$).
κ	Mother function ($\kappa : \mathbb{R}^l \to \mathbb{R}$). Can also be seen as a kernel.
φ	Regression vector: the past data that is used to predict the future output data ($\varphi \in \mathbb{R}^l$)
θ	Parameters of the predictor function ($\theta \in \Theta$). θ consists of parameters α, β and γ.
ϵ	Prediction error, or difference between predicted and measured output ($\epsilon \in \mathbb{R}^n$)
$\hat{y}$	Predicted or estimated output ($\hat{y} \in \mathbb{R}^n$)
LP	Learning progress, or decrease in prediction error.
$\tilde{n}$	Model order: number of basis functions in the model.
TLE	Time-loop error or reconstruction error: the difference between the regressor and the output of the inverse of the predictor function.
E	Total error, for example the sum of squared prediction errors ($E \in \mathbb{R}$)
f	Predictor function ($f : \mathbb{R}^l \to \mathbb{R}^n$)
g	State transition function of a state-space model
h	Output function of a state-space model
U	Set of possible control values ($U \subseteq \mathbb{R}^m$).
u	Input, or actuator data ($u \in \mathbb{R}^m$). Also called control value.
x	State of the system
y	Output, or sensor data ($y \in \mathbb{R}^n$)