Universiteit Twente

faculteit der
elektrotechniek

On training strategies for parsimonious learning feed-forward controllers

Govert Valkenburg

M.Sc. Thesis

Supervisors: prof.dr.ir. J. van Amerongen
dr.ir. T.J.A. de Vries
ir. B.J. de Kruif

28 February 2001
Report Number
001R2001

Control Laboratory
Dpt. of Electrical
Engineering

University of Twente
P.O. Box 217
NL 7500 AE Enschede
The Netherlands

Summary

This thesis addresses the question how to train a Parsimonious Learning Feed-Forward Controller (PLFFC). In Learning Feed-Forward control (LFFC) generally a well-conditioned feedback control signal is used to train a feed-forward controller, which mainly performs a function approximation. Then the feedback control signal is seen as an approximation of the inverse plant dynamics with respect to a particular reference signal. The feed-forward controller generally converges to the inverse plant dynamics.

In a PLFFC a so-called parsimonious B-spline network (P-BSN) is used for the function approximation. In a second order BSN, (the only type used in this research) a function is approximated part-wise by straight lines. In a P-BSN a multivariate function is approximated by the sum of a set of univariate (or lower-variate) sub-BSNs. It is not obvious what information should be stored in what sub-BSN. If information is stored into the wrong network, we speak of *interference*.

It turns out that the quality of training a PLFFC mainly depends on symmetry in the training paths. Due to symmetry, some effects add up to zero, which is important for avoiding interference between the sub-BSNs.

The theory was applied both in simulations and in experiments. The simulations turn out to be successful: a significant decrease of the error is achieved. It turns out that the error reduction by poor paths is about equal to the reduction by well-conditioned paths, whereas the extent to which they extract the correct mappings from the plants is really smaller, in this sense that the mappings learnt by the BSN do not equal the target functions. Furthermore it was shown that a poorly conditioned path can even cause divergence.

In experiments it was found that discontinuous relations in the plant disable the LFFC to learn the correct relations. This remains the same when PLFFC is used. This is the main reason why the performance PLFFC when applied to a real plant according to the presented theory, is limited. Furthermore in some cases the difference between reference and system states is too large. Still a significant error reduction is achieved.

The theory was proven to be correct, with some remarks to its applicability. A training procedure for PLFFC is proposed in the end.

Samenvatting

Deze scriptie behandelt de vraag hoe een Parsimonious Learning Feed-Forward Controller (PLFFC, spaarzame lerende vooruitregeling) dient te worden getraind. In Learning Feed-Forward Control (LFFC, lerende vooruitregeling) wordt het uitgangssignaal van een goed geconditioneerde teruggekoppelde regeling gebruikt om een vooruitregeling te laten leren. Deze vooruitregeling bestaat uit een functie-approximator. Het terugkoppelingsregelsignaal wordt gezien als een benadering van de inverse proces-dynamica, toegepast op het referentie-signaal, welke zo geleerd wordt.

In een PLFFC wordt een Parsimonious B-Spline Network (P-BSN, spaarzaam B-spline netwerk) gebruikt voor de functie-approximatie. In een tweede-orde BSN (het enige type gebruikt in dit onderzoek) wordt een functie benaderd door een aantal rechte lijnstukken. In een P-BSN wordt een multivariabele functie benaderd door de som van meerdere univariabele (of 'lager'-variabele) sub-netwerken. Het is niet op voorhand bekend welke informatie in welk netwerk dient te worden opgeslagen.

Het blijkt dat het trainen van een PLFFC goeddeels afhangt van de symmetrie in het trainingspad. Door deze symmetrie middelen bepaalde effecten precies uit, wat belangrijk is voor het voorkomen van foutief opslaan van informatie.

De theorie is zowel in simulaties als in experimenten toegepast. De simulaties zijn succesvol: een goede reductie van de fout blijkt te worden gehaald. Het blijkt dat de foutreductie bij slechte paden ongeveer gelijk is aan die bij goede paden, maar dat goede paden veel beter in staat zijn de fysische functies uit het proces te abstraheren. Dit laatste wordt beoordeeld door de inhoud van een netwerk na leren te vergelijken met de fysische functies. Verder is aangetoond dat een slecht pad divergentie kan veroorzaken.

In experimenten bleek dat discontinue relaties binnen het proces verhinderen dat LFFC goed leert. Dit blijkt net zo zeer voor PLFFC te gelden, waardoor de theorie uit deze scriptie slechts beperkt toepasbaar bleek in de praktijk. Verder is soms het verschil tussen referenties en systeemtoestanden te groot. Niettemin is een goede foutreductie behaald.

De theorie uit deze scriptie blijkt correct te zijn, al verdient toepassing verder onderzoek. Een trainingsprocedure voor PLFFC wordt tenslotte gegeven.

Preface

Give me one viewpoint outside of nature, and I will demistify her to her heart - Archimedes

The present report is the result of half a year of hard work at the Control Laboratory. I tend to say it was a good time doing, but as Archimedes stated: one cannot judge on something one is part of. The same holds for the results of my research. For the time being, they look fairly satisfactory, but their true value can only be established a long time afterward.

Concerning their present apparent value, I think I should pay gratitude to my supervisors. I thank Job van Amerongen and Theo the Vries for the opportunity to carry out this assignment, as well as for the fruitful discussions. A special word of appreciation I speak to Bas de Kruif. Bas' intensive guiding has been an important cornerstone for this project. Without, the outcomes would certainly been different, and then I do not necessarily mean 'better'. Bas, we do have very different ways of thinking, which I think have their merits one to another. Good luck with your Ph.D. Thesis!

Furthermore I should thank Belle, Hanneke and Marion, for the supporting company, and sometimes the confronting remarks: these proved to be the most important *archimedean points* in my recent life.

Thanks for my parents, for supporting me all the time. Thanks to Jochem and Wessel for being Jochem and Wessel, without whom I would not be Govert.

Finally I should thank Adolphe, Nicolo and Igor, for making this life bearable at all.

Govert Valkenburg

Enschede, february 2001

Nomenclature

Throughout this thesis the following notation is used.

γ	learning rate of Learning Feed-Forward Control
d	spline width
D	set of samples
I	unity matrix
J	cost function
k	discrete time index
m	mass of the translator
$\boldsymbol{\mu}$	membership vector of B-spline network
$\mathbf{p}$	cross-correlation vector
ω_b	bandwidth of the control system
ω_l	high-pass cut-off frequency
p	probability
r	reference signal
R	auto-correlation matrix
R^{-1}	inverse of matrix R
$\widetilde{R^{-1}}$	part-wise inverse of matrix R
s	Laplace-operator
S	sensitivity function
T_s	sample time
u	control signal
$\hat{u}$	output of function approximator/ approximation of u
$\overline{u}$	average of a control signal over a number of samples
$\mathbf{w}$	weight vector of B-spline network
$\hat{\mathbf{w}}$	optimal weight vector of B-spline network

Contents

Chapter 1

Introduction

This thesis describes the results of an M.Sc.-assignment at the Control Laboratory. The purpose of the assignment is to formulate a training strategy for so-called parsimonious learning feed-forward controllers (PLFFCs). In this chapter a short introduction to PLFFCs and their specific features is given. In the last section of this chapter, the goal of the project is formulated, and an overview of the remainder of this thesis is given. We start with the question why learning is relevant in control theory at all.

1.1 Why learning?

The most commonly formulated control problem, is a situation where we have a plant that does not by itself behave the way we want it to. In order to make it behave conform our demands, we need to apply a control algorithm to the plant. The quality of this control algorithm depends, among other things, on our knowledge of the plant. This knowledge is often limited. This can be caused by low-precision production processes (fabricated plants being slightly different from what the supplier tells), by poor modelling due to complexity of the process (one could think of environmental processes), by neglect of high-frequency dynamics, or e.g. by slight changes in the plant as time proceeds.

In this study we consider plants of which we do not know the entire dynamics. The objective is to have these dynamics learnt by a function approximator, and then use this learned mapping to correct for the dynamics. This technique is referred to as *learning control*. In this thesis a specific form of learning control, namely PLFFC, will be addressed.

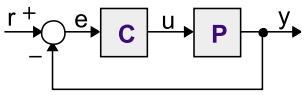

Figure 1.1: A SISO feedback control system

1.2 Parsimonious learning feed-forward control

1.2.1 Learning feed-forward control

In figure 1.1 a SISO feedback control system is shown. The respective blocks C and P represent the compensator and the plant. r is the reference signal, u is the control signal, e is the error and y is the output signal. The following sequence of equations holds:

$$\begin{aligned} e &= r - (PC)e \\ e(1+PC) &= r \\ e &= (1+PC)^{-1}r \end{aligned} \tag{1.1}$$

We now continue:

$$\begin{aligned} u &= Ce \\ &= (1+PC)^{-1}Cr \\ &= \frac{r}{C^{-1}+P} \end{aligned} \tag{1.2}$$

It now follows that if the transfer function of C is such that the error e is generally small, the signal u practically equals $P^{-1}r$, which is exactly the optimal feed-forward signal (Åström and Wittenmark, 1997, p. 234). Generally this means that C has a large proportional gain, which may be upper bounded by stability criteria.

If we apply this feed-forward control signal a priori, i.e. as a direct function of the reference and without measuring the error, we have a so-called feed-forward controller (FFC). See figure 1.2 for a SISO feedback control system, enhanced with a feed-forward path.

In our case the mapping from reference r to the feed-forward signal is not known in advance, so it should be learnt first. The system in figure 1.3 depicts such a learning feed-forward controller, which is adjusted as a result of previous control actions. In the picture, the block named L represents a learning strategy. In this project an LFFC is implemented by using a B-spline network (see also subsection 1.2.2) as a function approximator.

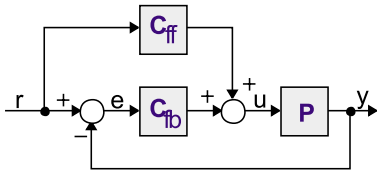

Figure 1.2: A SISO feedback-feed-forward control system

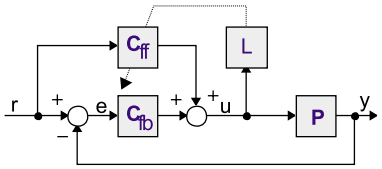

Figure 1.3: SISO learning feed-forward control system

1.2.2 B-spline networks

Artificial Neural Networks (ANNs) constitute a class of function approximators. ANNs generally learn a mapping from a known quantity (e.g. a reference signal) to an unknown quantity (e.g. the control signal applied to the plant). By learning, we get an approximation, and can thus use this knowledge to increase performance.

A B-spline Network (BSN) is a member of the class of ANNs. In a BSN a function is approximated by a number of basic splines (a special type of low-order polynomials), each acting on a finite input range with a certain weight. The principle of function approximation by a BSN is depicted in figure 1.4. A more thorough description is given in section 2.1. In this project a BSNs will be used to learn the mapping from the reference signal to the control signal. Velthuis (2000, p. 13) and (De Vries, Velthuis and

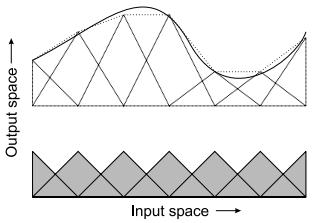

Figure 1.4: Function approximation by a univariate BSN

Van Amerongen, 2000) give a number of advantages of this type of network:

- A system employing a BSN does not suffer from local minima with

respect to the optimality criterion, since the output is a linear function of the network weights. This means that parameters converge to certain limit points, regardless of their initial values.

- Local learning is well-supported, since B-splines have only a finite support, and only a finite number of B-splines have a membership at a certain input value.
- Tuneable precision is possible: the accuracy of the approximation by the BSN can easily be influenced by changing the number of B-splines on a certain input range.
- A BSN is rather transparent, in this sense that its contents can be interpreted intuitively.
- A BSN has few design parameters, compared with other kinds of ANNs.

The main disadvantage of a BSN is that it generally suffers from the so-called curse of dimensionality, which will be addressed in the next paragraph.

A second disadvantage is that due to the local support of the splines, the generalisation ability of a BSN in general is limited, compared to ANNs with a larger support of its basis functions.

1.2.3 Parsimonious networks

For a BSN, the number of network weights increases exponentially with the number of network inputs. From figure 1.5 it is clear that a network featuring two inputs with each domain split up in n splines needs n^2 network coefficients. Beside on memory usage, this also has its implications on training demands: the required amount of training data increases linearly with the number of coefficients, so it increases exponentially with the number of inputs. This phenomenon is known as the *curse of dimensionality* (Brown and Harris, 1994, p. 321).

One way to avoid this problem is to replace the bivariate network with two univariate networks, as shown in figure 1.6. This new configuration is called a *parsimonious network* (Velthuis, 2000; Bossley, 1997, p.100).

From the figures it can be seen that the configuration of figure 1.5 has different properties than the configuration of figure 1.6. In the bivariate network the output depends on only one point in the input space, which is uniquely given by the pair of input values (r_1, r_2). By contrast, in the univariate networks the output depends on two points in different input spaces, r_1 and r_2, which influence both exactly one network.

There are conditions a function has to meet, in order to be approximated by a parsimonious network instead of a multivariate network. Knowledge is needed regarding the question whether the multivariate target function can

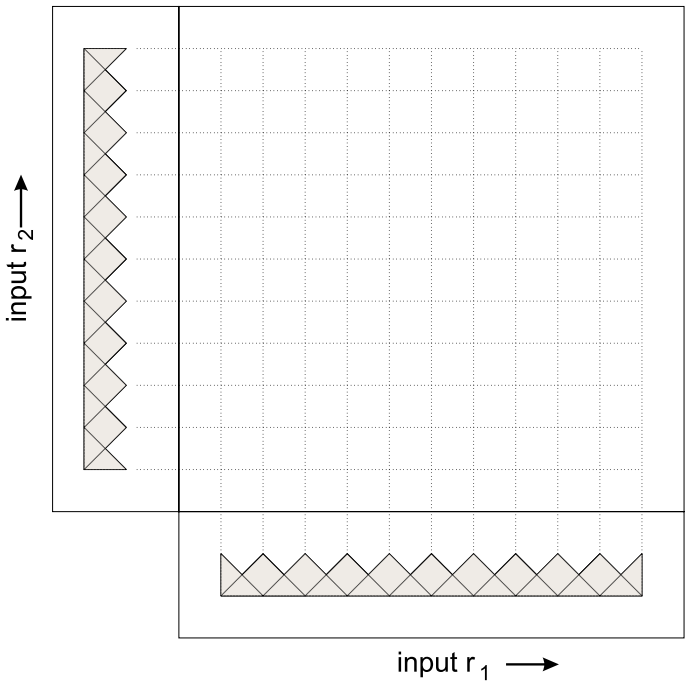

Figure 1.5: Bivariate B-spline network

be expressed as the sum of univariate functions. I.e., we should be able to write:

$$u = u(r_1, r_2) = u_1(r_1) + u_2(r_2) \tag{1.3}$$

Velthuis (2000, p. 101) explains this ANalysis Of VAriance (ANOVA) for systems with an arbitrary number of inputs. From this ANOVA-representation it follows that a reduction of the number of network coefficients is possible if in the expression

$$\begin{aligned} u &= u(r_1, r_2, \cdots, r_n) \\ &= u_0 + \sum_i u_i(r_i) + \sum_{i,j} u_{i,j}(r_i, r_j) + \cdots + u_{1,2,\cdots,n}(r_1, r_2, \cdots, r_n) \end{aligned} \tag{1.4}$$

a sufficient number of terms from the right-hand side (at least the last one) cancels.

Besides avoiding the curse of dimensionality, there is another advantage of the second configuration above the first: it has a higher generalisation ability.

Nevertheless, the configuration also has some disadvantages. The main disadvantage is the problem that adjusting the network parameters is not

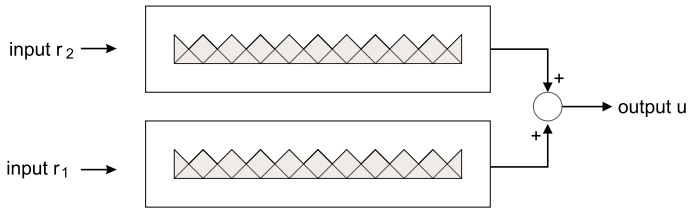

Figure 1.6: Combination of two univariate B-spline networks

trivial: to achieve a certain output of the overall network for a certain combination of input r_1 and input r_2, an infinite number of combinations of outputs of the networks are valid, since the outputs of the univariate networks are added (see figure 1.6). This has its implications on the training sets: although training sets might be smaller, a higher demand may be put on its properties. Furthermore, because of possible interference between several sub-network and between several splines in one network, multiple session training may be needed. This thesis addresses strategies to solve these problems.

With the knowledge from this section and the previous sections, we can state the following definition: with *parsimonious learning feed-forward control* (PLFFC) we indicate the class of systems where parsimonious networks are incorporated as function approximators in a feed-forward control system.

1.3 Case: PLFFC for the linear motor

The theory formulated in this thesis has been applied to a mathematical model of a linear motor, as well as to the physical linear motor itself. A linear motor can be seen as is a moving mass with one degree of freedom, namely a linear movement. The mass receives its thrust force by magnetic induction. Several applications of linear motors have been described, ranging from the Maglev and Transrapid gliding trains, to high-precision scanning plants for medical purposes. The linear motor considered in the simulations, has a mass of approximately 37 kg, and a free range of 0.5 m. The physical linear motor, made by Tecnotion, has a mass of approximately 5 kg, and a free range of 0.5 m too.

The linear motor comprises some interesting features relevant to this project. First, its mass may deviate a little from its specifications. This deviation can be learnt. Second, it suffers from cogging, a force due to non-homogeneity of the magnetic field. Third, the motor suffers from mechanical friction. And fourth, it suffers from commutation inaccuracy due to variations in placement and strength of the permanent magnets, which brings about an undesirable force.

Because of the non-homogeneity of the magnetic field, the latter depends on both position and velocity. The other features depend only on acceleration, position and velocity respectively.

These four undesirable properties are well-suited for learning by a PLFFC. The phenomena are assumed to simply add up, so a decomposition of the sum is possible. If indeed all undesirable behaviour is due to either of these phenomena, it can be compensated for by means of a PLFFC.

A more accurate description of the linear motor and the decomposition for PLFFC is given in chapter 2.

1.4 Aim, methods and thesis outline

The objective of this project is to formulate a training procedure, in order to let a parsimonious B-spline network learn correctly. The goal of this project is formulated as follows:

> *To formulate a framework of principles on learning strategies for parsimonious learning feed-forward controllers, as well as a design procedure to develop reference paths for training a parsimonious learning feed-forward controller.*

To this end, first an analysis of both univariate and bivariate BSNs is performed. A number of principles are presented with foundations (chapter 2), and tested empirically.

From within the framework constituted by these principles, reference paths can be defined (section 3.1). These reference paths are applied in simulations with a univariate PLFFC, in simulations with a bivariate PLFFC, and in simulations with a bivariate PLFFC with measurement noise (chapter 3).

After that, some paths are applied in a real experiment (chapter 4). It turns out that the possibilities, such as e.g. data storage and mathematical manipulations, in physical experiments are much more restricting than in simulations, so not everything valid in simulations can be confirmed in experiments. Furthermore, we will see that the difficulties of LFFC will disturb the experiments partly.

From these results some conclusions will be drawn. We will find sufficient reasons to accept the formulated theory, be it with some important remarks to its applicability. From this, a procedure for training a PLFFC is given. Therefore useful recommendations for future research will be given (chapter 5).

Chapter 2

Theory

In this chapter the theoretical background of the project is addressed. First the properties of BSNs are discussed, both the univariate and bivariate families, as well as their parsimonious versions. Furthermore noise and stability are discussed briefly, as well as a more pragmatic approach to train BSNs. Then the linear motor model and its implications on PLFFC are addressed.

2.1 B-spline networks

2.1.1 Univariate B-spline networks

In a B-spline network a function is approximated by a number of B-splines. B-splines (basic splines) are low-order polynomials with special properties. For more formal definitions of B-splines the reader is referred to Brown and Harris (1994) and Velthuis (2000). In this project only second order BSNs are used. These incorporate first order polynomials, i.e. straight lines. In this case a BSN can be seen as a linear interpolator. The approximation is illustrated in figure 2.1. The function (bold line) is approximated (dotted

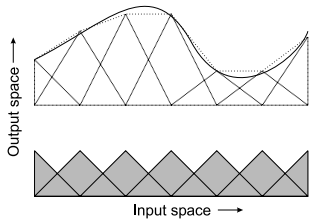

Figure 2.1: Function approximation by a univariate BSN

line) by assigning a weight ('value') to each B-spline (blank triangles). For clarity the distribution of the B-splines is depicted below the approximation (grey triangles).

In this project the BSNs are updated off-line. This means that during operation the contents of the network is left unaffected, and between two runs the new contents of the network is calculated. Two reasons motivate this decision. Firstly, it provides more accurate time averaging, on which most of the ideas in this thesis are based. Off-line time averaging of a discrete-time signal $u(k)$, given by

$$\overline{u} = \frac{1}{K} \sum_{k=1}^{K} u(k) \tag{2.1}$$

with K the number of samples and k the discrete time index, is different from the on-line approximation

$$\hat{u}(k) = \gamma \cdot u(k) + (1 - \gamma) \cdot \hat{u}(k-1) \tag{2.2}$$

of the time average, where γ is a learning factor. In the online case the relevance of a sample to the average decreases when the sample was taken further back in time, whereas in this study all samples are considered equally important. (Note that by using a time varying learning factor γ, we can transform the online averaging into the off-line version. This might be interesting for larger training sets and time-varying processes, but it is left out of scope in this study.)

Secondly, off-line learning prevents instability due to dynamical behaviour of the learning loop (L-block and feed-forward controller, see figure 1.3), simply because there is no dynamical behaviour: network weights are updated only when the plant is not running. With online learning, the properties of a BSN allow a learning action in a certain sample to influence the output of the BSN at the next sample. This dynamical behaviour may cause the loop to become unstable. This dynamical instability should not be confused with divergence of the network parameters, which can still occur in the off-line case.

Learning

Every point r in the (one-dimensional) input space corresponds exactly to one vector of the form $\boldsymbol{\mu}(r) = (\mu_1(r), \mu_2(r), \cdots, \mu_N(r))^T$. Here every coefficient $\mu_i(r)$ indicates the extent to which the corresponding B-spline number i contributes to the output for this r. This extent is usually referred to as the *membership* of spline i at point r.

Let *weight vector* $\mathbf{w} = (w_1, w_2, \cdots, w_N)^T$ be the vector with the magnitudes of the B-splines. Then the output $\hat{u}$ of the network for a certain r is given by:

$$\hat{u}(r) = \boldsymbol{\mu}(r)^T \mathbf{w} \tag{2.3}$$

We now want to minimise the difference between this approximation $\hat{u}$ and the target function u for the entire domain of r. Therefore we write down the following cost function:

$$J_c = \int_{r_0}^{r_1} \left(\hat{u}(r) - u(r)\right)^2 dr \tag{2.4}$$

with r_0 and r_1 respectively the minimum and maximum values for r. Verwoerd (2000) showed that the optimal solution of $\mathbf{w}$ is given by

$$\mathbf{w} = R^{-1}\mathbf{p} \tag{2.5}$$

where R is the auto-correlation matrix defined by

$$R = \int_{r_0}^{r_1} \boldsymbol{\mu}(r)\boldsymbol{\mu}(r)^T dr \tag{2.6}$$

and $\mathbf{p}$ is the cross-correlation vector defined by

$$\mathbf{p} = \int_{r_0}^{r_1} u(r)\boldsymbol{\mu}(r) dr \tag{2.7}$$

However, this relation is valid for continuous functions only (hence the subscript 'c' in the cost function). In order to apply to (2.5) the entire function $u(r)$ needs to be known, whereas in practice we will only have a limited number of samples of u. So we need a new cost function which is minimised for the available number of samples. The following discrete cost function approximates the continuous one for a set D of samples:

$$J_d = \sum_{k \in D} \left(u(r(k)) - \hat{u}(r(k))\right)^2 \tag{2.8}$$

(Note that this function is lower bounded by zero if the number of samples is smaller than or equal to the number of B-splines.) Now again (2.5) holds (Verwoerd, 2000), but R and $\mathbf{p}$ are defined in a different manner:

$$\begin{aligned} R &= \sum_{k \in D} \boldsymbol{\mu}(r(k))\boldsymbol{\mu}(r(k))^T \\ &= \sum_{k \in D} \begin{bmatrix} \mu_1(k)^2 & \mu_1(k)\mu_2(k) & \cdots & \mu_1(k)\mu_N(k) \\ \mu_2(k)\mu_1(k) & \mu_2(k)^2 & \cdots & \mu_2(k)\mu_N(k) \\ \vdots & \vdots & \ddots & \vdots \\ \mu_N(k)\mu_1(k) & \mu_N(k)\mu_2(k) & \cdots & \mu_N(k)^2 \end{bmatrix} \end{aligned} \tag{2.9}$$

$$\begin{aligned} \mathbf{p} &= \sum_{k \in D} u_r(k)\boldsymbol{\mu}(k) \\ &= \sum_{k \in D} \begin{bmatrix} u_r(k)\mu_1(k) \\ u_r(k)\mu_2(k) \\ \vdots \\ u_r(k)\mu_N(k) \end{bmatrix} \end{aligned} \tag{2.10}$$

For a second order BSN R generally has the form

$$R = \begin{bmatrix} \mu_{11} & \mu_{12} & 0 & \cdots & \cdots \\ \mu_{21} & \mu_{22} & \mu_{23} & 0 & \cdots \\ 0 & \mu_{32} & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \mu_{N-1,N} \\ \vdots & \vdots & 0 & \mu_{N,N-1} & \mu_{NN} \end{bmatrix} \tag{2.11}$$

Here μ_{ij} is the sum over all samples of the according membership cross term. It is not unthinkable that a certain spline i remains unvisited, i.e., no sample in the support of spline i is in the training set. Then the corresponding μ_{ii} is zero, as well as its neighbour cross-terms, thus resulting in a singular matrix R. In this case R cannot be inverted regularly, but it can be inverted partwise, i.e. all square submatrices that are on the main diagonal of R and that are not singular can be inverted, and be placed on the corresponding place on the diagonal of partwise inverse $\widetilde{R^{-1}}$. In this case the network can be seen as a set of sub-networks, each acting on a limited sub-domain of the original network. See also appendix A for this partwise inversion.

Regularisation

Since training sets generally suffer from noise, disturbances and imperfect reference paths, differences between the target function and its approximation can be large. Especially with badly conditioned training sets, network coefficients can blow up to very large values (R being nearly singular, as a result of poorly visited splines). To avoid this, regularisation is introduced. To this end, we introduce an enhanced cost criterion (applying definitions (2.7) and (2.6) for $\mathbf{p}$ and R) respectively:

$$\begin{aligned} J_{d,r} &= \left[\sum_{k \in D} \left(u(r(k)) - \hat{u}(r(k)) \right)^2 \right] + \mathbf{w}^T Q \mathbf{w} \\ &= \sum_{k \in D} \left(u(r(k)) \right)^2 - 2\mathbf{w}^T \mathbf{p} + \mathbf{w}^T R \mathbf{w} + \mathbf{w}^T Q \mathbf{w} \end{aligned} \tag{2.12}$$

The only difference between this regularised discrete cost function, and the former discrete cost function, is that a quadratic penalty is put on the absolute value of the network weights. This penalty is quantised by matrix Q which should be positive semi-definite (i.e. $\mathbf{v}^T Q \mathbf{v} \geq 0 \ \ \forall \ \mathbf{v}$), and such that $Q + R$ is regular (i.e. invertible). From the general appearance of R (see (2.11)) it follows that these demands are generally met if Q is a positive diagonal matrix.

To optimise this criterion, we take the gradient with respect to $\mathbf{w}$, and set it equal to zero, yielding:

$$\frac{\partial J_{d,r}}{\partial \mathbf{w}} = -2\mathbf{p} + 2R\mathbf{w} + 2Q\mathbf{w} = 0 \tag{2.13}$$

This gives the optimal solution for $\mathbf{w}$:

$$\hat{\mathbf{w}} = (R + Q)^{-1}\mathbf{p} \tag{2.14}$$

Now the remaining question is how to find a sensible value for Q. In a well-conditioned training set, we do not want to experience any influence of Q onto the learning process. Now let's assume that $Q = I \cdot 10^{-4}$, with I the unity matrix. In this case the matrix Q is significant only on those elements on the main diagonal where R has an accordingly small (or smaller) element. We should now realise that all elements on the main diagonal of R are sums of quadratic memberships of certain splines. We can interpret the square root of these sums as the second order norm of the extent to which the spline was visited. This norm is at most order 10^{-2} if the square is at most order 10^{-4} (which was assumed from the fact that Q is relevant at all). So Q is relevant only on those splines, which were visited with a membership of 10^{-2} or less (counted by the second order norm). We can safely call this *badly visited*, and assume that with $Q = I \cdot 10^{-4}$ the influence of regularisation is justified. Any visit with a membership larger than 10^{-2}, which in case of a second order BSN equals a single visit to 98% of the support of the spline, will cause the regularisation to become irrelevant.

This regularisation comes close to the use of the partwise inversion algorithm previously described. The main difference is that partwise inversion deals with singular matrices, which contain sub-matrices which are by themselves well conditioned. Regularisation on the other hand, takes care of matrices which are nearly singular (but not necessarily truly singular) by adding a small-valued non-singular matrix to it. This means that zero-valued elements will now become nonzero, thus coming up with virtual visits to splines which are not visited at all, and making the final function approximation less reliable. From this we can state that partwise inversion is preferred if the sub-matrices are well conditioned. Should this not be the case, regularisation can be applied. In practice this means that partwise inversion should always be applied, and only be replaced by regularisation if network contents blow up to implausible large values.

2.1.2 Bivariate B-spline networks

Analogously to a univariate BSN, we can interpret a bivariate second order BSN as a two-dimensional linear interpolation table. For a bivariate BSN learning is slightly different (Verwoerd, 2000). To get a learning rule, the network should be transformed in such a way, that a new univariate network arises. This univariate network can now be dealt with in the same way as described before. For more details the reader is referred to Verwoerd (2000, p. 23 and further). The main problem is now that the auto-correlation matrix R is generally not regular. This means that inversion is impossible, and that learning can only be performed by a sub-optimal algorithm (which

after a number of learning episodes still converges to the optimal solution). In the case of our bivariate BSN this yields the following update rule:

$$\Delta \mathbf{w} = \gamma \cdot Diag^{\widetilde{-1}} \left[\sum_{k \in D} \boldsymbol{\mu}(\mathbf{r}(k)) \right] \left[\sum_{k \in D} \boldsymbol{\mu}(\mathbf{r}(k)) \left(u(\mathbf{r}(k)) - \hat{u}(\mathbf{r}(k)) \right) \right] \quad (2.15)$$

See appendix A for the *Diag*-operator and partwise inversion of a matrix. Here γ is the learning rate of the BSN, which in case of incorporation in an LFFC is the same as the learning rate of the LFFC.

2.2 Parsimonious B-spline networks

From now on we classify parsimonious networks after the 'highest variate' network contained in it. This means that e.g. a parsimonious network containing two univariate networks and one bivariate network, is classified as bivariate. One should not find oneself confused by the fact that the parsimonious network itself has perhaps even four inputs, although still it is not classified as four-fold multivariate.

2.2.1 Univariate parsimonious B-spline networks

Four principles with foundations are given in this subsection. They concern the features of a training data set, as well as the strategies one should follow when updating a network.

Consider a parsimonious network with an arbitrary number of inputs $r_1 \cdots r_N$ and an according number of univariate networks. Let the overall target function $u(r_1, r_2, \cdots, r_N)$ (i.e. the function which should be approximated sufficiently accurately by the network after sufficient training) be the sum of the partial target functions $u_1(r_1)$, $u_2(r_2)$, $\cdots$, $u_N(r_N)$. Let $\mathbf{r}(k) = (r_1(k), r_2(k), \cdots, r_N(k))^T$ be the vector of input values as a function of discrete time index k (reference trajectory). Let $n, m \in [1, 2, \cdots, N], n \neq m$. Then the following principles hold.

Principle 2.1 *For a small neighbourhood Δr_{n0} of an arbitrary but certain value r_{n0} of input r_n, the learning of a function $u_n(r_n)$ does not suffer from interference by any function $u_m(r_m)$, if the values of $u_m(r_m(k))$ for this input vector sequence $\mathbf{r}(k)$ add up to zero over the $(r_1, r_2, \cdots, r_{n-1}, r_{n+1}, \cdots, r_N) \times \Delta r_{n0}$-subspace of $\mathbf{r}$, i.e. if there is no correlation between the functions on this subspace for the given sequence $\mathbf{r}(k)$.*

This means that a number of values of r_n in Δr_{n0} has to occur a number of times with different values of r_m, such that the corresponding values of target function $u_m(r_m)$ add up to zero, i.e. statistical correlation is zero.

Foundation 2.1 First consider the case where r_1 and r_2 are discrete variables. Then Δr_{n0} has zero width. Let u be written as (according to the ANOVA representation):

$$u(\mathbf{r}) = u_1(r_1) + u_2(r_2) \tag{2.16}$$

In this case the foundation is straightforward. Let two vectors $\mathbf{r}(1)$ and $\mathbf{r}(2)$ occur, with the values $(r_1(1), r_2(1))$ and $(r_1(2), r_2(2))$, with $r_1(1) = r_1(2)$ and $u_2(r_2(1)) + u_2(r_2(2)) = 0$. The average of the two function values resulting from these pairs is:

$$\begin{aligned}
\overline{u} &= \frac{1}{2}\left(u(\mathbf{r}(1)) + u(\mathbf{r}(2))\right) \\
&= \frac{1}{2}\left(u_1(r_1(1)) + u_1(r_1(2)) + u_2(r_2(1)) + u_2(r_2(2))\right) \\
&= u_1(r_1(1)) + \frac{1}{2}\left(u_2(r_2(1)) + u_2(r_2(2))\right) \\
&= u_1(r_1(1))
\end{aligned} \tag{2.17}$$

This means that the average equals the value of $u_1(r_1(1))$, which is exactly the value learnt by the r_1-network for this value of r_1. It is not influenced by values of u_2, since these add up to zero. It is obvious that the foundation also holds for larger numbers of samples, and larger numbers of partial functions u_n.

Now consider the case where $r_{1,2}$ are continuous variables. In this case it is unlikely that two identical values of r_1 will occur. Therefore we should not consider the exact value of r_1, but a small neighbourhood of it. Say a set of K samples has its values of r_1 in a Δ-neighbourhood of $r_1(1)$, with this neighbourhood smaller than the spline width. Then the average value of this set of samples, as far as it is relevant to the spline, is given by (with μ the membership of this certain spline):

$$\begin{aligned}
\overline{u} &= \frac{1}{K}\sum_{k=1}^{K} u(\mathbf{r}(k))\mu(\mathbf{r}(k)) \\
&= \frac{1}{K}\sum_{k=1}^{K} u_1(r_1(k))\mu(r_1(k)) + \frac{1}{K}\sum_{k=1}^{K} u_2(r_2(k))\mu(r_1(k)) \\
&\approx \frac{\mu(r_1(\cdot))}{K}\sum_{k=1}^{K} u_1(r_1(k)) + \frac{\mu(r_1(\cdot))}{K}\sum_{k=1}^{K} u_2(r_2(k))
\end{aligned} \tag{2.18}$$

In the last step it was assumed that since Δ is small in comparison to the spline width, μ will be approximately constant for the set of samples. The index k to r_1 was replaced by a dot, to indicate that the value of r_1 no longer depends on k. In this case the second sum adds up to zero (premiss of the principle).

So far we have neglected the cross terms of μ in (2.6). This is justified from the fact, that since Δ is small, we can consider the sum above as a new set of samples of $u_1(r_1)$ (without interference of u_2!), which will be incorporated in (2.6) correctly.

Principle 2.2 *For a small neighbourhood Δr_{n0} of an arbitrary but certain value r_{n0} of input r_n, the learning of a function $u_n(r_n)$ does not suffer from interference by any function $u_m(r_m)$ if, for this trajectory $\mathbf{r}(k)$, this $u_m(r_m)$ has an expected value 0 over the $(r_1, r_2, \cdots, r_{n-1}, r_{n+1}, \cdots, r_N) \times \Delta r_{n0}$-subspace of $\mathbf{r}$ corresponding to this r_{n0}, and the number of occurrences of this value of r_n within this neighbourhood is large enough.*

Foundation 2.2 Actually this is a generalisation of the first principle. If the number of occurrences of this value of r_n is large enough, the sum over the $(r_1, r_2, \cdots, r_{n-1}, r_{n+1}, \cdots, r_N) \times \Delta r_{n0}$-subspace of $\mathbf{r}$ corresponding to this value of r_n will be zero because of the expected value of zero. We should not neglect the fact that the expected value depends on the trajectory $\mathbf{r}(k)$. The following statistical property holds for a sufficient number of samples (Bhattacharryya and Johnson, 1977):

$$\frac{1}{K}\sum_{k=1}^{K} u_2(r_2(k)) \approx E_{\mathbf{r}(k)}[u_2] = 0 \tag{2.19}$$

In this case the foundation is valid. The subscript $\mathbf{r}(k)$ in the expected value $E_{\mathbf{r}(k)}$ is shown to emphasize the dependence of E on the trajectory.

Principle 2.3 *If a function to be learnt is odd-symmetrical in zero, forcing this symmetry reduces the interference from and to other functions.*

This principle exploits the first and second principle. If all measurements for negative values of r_n are rotated by 180 degrees around the origin, the density of measurements is practically doubled on the positive domain, which improves the statistical reliability. Besides, the following foundation is valid.

Foundation 2.3 Again first we discuss the case where r_1 and r_2 are discrete variables. Let $u_2(r_2)$ be odd-symmetrical, i.e. $u_2(-r_2) = -u_2(r_2) \forall r_2$. Now let two vectors $\mathbf{r}$ occur, with values $(r_1(1), r_2(1))$ and $(r_1(2), r_2(2))$, with $r_1(1) = r_1(2)$ and $r_2(1) = -r_2(2) \geq 0$. Now rotate the sample corresponding to $\mathbf{r}(2)$ (because of its negative value of r_2) by 180 degrees around the origin. We then get:

$$\begin{aligned}
\overline{u} &= \frac{1}{2}\left(u(\mathbf{r}(1)) - u(\mathbf{r}(2))\right) && (2.20)\\
&= \frac{1}{2}\left(u_1(r_1(1)) - u_1(r_1(2)) + u_2(r_2(1)) - u_2(r_2(2))\right) && (2.21)\\
&= \frac{1}{2}\left(u_2(r_2(1)) + u_2(-r_2(2))\right) && (2.22)\\
&= u_2(r_2(1)) && (2.23)
\end{aligned}$$

Regardless of the value of u_1, this summation yields the value of $u_2(r_2(1))$. Because of the odd symmetry also the value of u_2 for $r_2(2)$ is known.

Now for continuous variables $r_{1,2}$ the generalisation goes analogously to the second part of foundation 2.1. It is again obvious that this foundation also holds for larger numbers of samples, and larger numbers of partial functions u_n. It should be noted that in the reference path opposite values of r_2 have to occur with identical values of r_1.

The statement that forced symmetry also reduces interference *to* other networks, is understood from the fact that a well-trained network leaves a cleaner residue than a poor-trained one.

Principle 2.4 *Consider two networks, that are to learn target functions that are of the same order of magnitude. If the inputs of the two networks are completely uncorrelated (i.e. statistical correlation is zero, which does not necessarily mean that the inputs are independent), the order in which they learn does not influence the quality of the learning. Moreover, no difference is brought about by the fact the second learning network uses the residue of the first network or the original input signal.*

If the inputs are correlated, the order does influence the quality of the learning (assumed that the second network takes the residue of the first as its input). In that case the network with the smallest number of splines should learn first.

Foundation 2.4 The first part of the principle follows obviously from principle 2.1. If there is no correlation, the data of one function does not influence the learning of the other function. Since this data is not correlated to the input, it does not matter whether it is subtracted from the data set (i.e. using the residue) or not (i.e. using the original input signal).

The second part is less obvious. Again we use principle 2.1. (Its generalisation in principle 2.2 also holds, but is omitted in this foundation.) The maximal size of neighbourhood Δr_{n0} (i.e. the largest size at which we can still qualify it as sufficiently small - we are not using this size quantitatively here, yet only qualitatively) depends on the spline width of the network. This means: the larger a spline is, the more points which add up to zero are allowed to deviate from this central value r_{n0}. We can say that a network is expected to generalise more if it consists of larger splines.

We may not directly compare the spline widths of two networks, since their inputs may consist of different quantities (e.g. position and velocity: comparing a spline width of 0.01 m with one of 0.1 m/s is a pointless activity). Nevertheless, we should take care that the occurring values of both inputs are well-distributed within their domains. In this case the number of splines is a competent measure for a sort of 'normalised spline width'.

This altogether supports the statement that the network with the largest generalisation, i.e. the network with the (relatively) largest splines, i.e. the

network with the smallest number of splines, suffers the least from interference by non-target functions. So this network should learn first, thus yielding a residue with the least possible interference for other networks.

If the target functions are not of the same order of magnitude, we should question whether this principle holds. This case is not considered here.

2.2.2 Multivariate parsimonious B-spline networks

As stated before, the ANOVA representation gives a decomposition of a multivariate function. The general notation of (1.4) is repeated here:

$$\begin{aligned} u &= u(r_1, r_2, \cdots, r_n) \\ &= u_0 + \sum_i u_i(r_i) + \sum_{i,j} u_{i,j}(r_i, r_j) + \cdots + u_{1,2,\cdots,n}(r_1, r_2, \cdots, r_n) \end{aligned} \tag{2.24}$$

We can see here, that there is no longer a unique ANOVA-representation of a function, when partial functions have (among others) the same input variables. Any function $u_i(r_i)$ can also be incorporated in a function $u_{\cdots,i,\cdots}(\cdots, r_i, \cdots)$. Mathematically spoken, there is no such thing as an optimal decomposition, since any decomposition yields the original function. However, from the physical reality we can intuitively define a desired solution. In the physical reality, the target function *is* a composition of several physical functions. We now want the decomposition by our function approximator to match the function in the same way. This means that the content of the 'lowest-variate' functions is maximal, since the projection of bivariate functions on any axis are zero. *This assumption is essential for the correctness of this reasoning.* This implies that the univariate networks should be updated first. In this case the information stored in the function is concentrated as much as possible into the left-hand terms of the ANOVA-representation, i.e. into the functions u_i and u_j. In practice this equals the situation with the highest generalisation ability.

It is important to note that the decomposition which follows from the ANOVA representation, is not necessarily in accordance with the inverse of the physical composition of the function. With respect to our linear motor system this means the following. If the projection of the commutation on either the position-axis or the velocity axis is nonzero, parts of the commutation will be learnt by the position or velocity network. In this case the approximation $\hat{u}(\mathbf{r})$ may perform exactly the same as $u(\mathbf{r})$, but the partial functions $u_{\cdots,i,\cdots}$ will not have a one-to-one relation with the physical features (such as e.g. cogging and friction) as described in section 2.5. This gives us a hard time evaluating the performance of the network, since we can no longer compare the partial approximations with the partial targets functions.

Principles 2.1, 2.2 and 2.3 also hold for bivariate BSNs. The first and second principle obviously hold, since a bivariate BSN can be transformed to a univariate BSN such that a unique bilateral relation exists (Verwoerd, 2000). The third principle also holds, but a new definition of odd symmetry should be given. In the bivariate case odd symmetry of a bivariate function to a certain reference variable r_n means:

$$u_{m,n}(r_m, -r_n) = -u_{m,n}(r_m, r_n) \ \ \forall \ \ r_n, r_m \tag{2.25}$$

Symmetry in *both* variables r_m and r_n did not occur in the present study. For the sake of completeness it is given here, though. This *point symmetry in the origin* is mathematically represented as:

$$u_{m,n}(-r_m, -r_n) = -u_{m,n}(r_m, r_n) \ \ \forall \ \ r_n, r_m \tag{2.26}$$

It can now be seen that the third principle still holds, provided that other reference signals (and their corresponding control signals) are kept equal.

The fourth principle does not obviously hold. It is hard to compare bivariate and univariate networks. The only point of reference we have, is the ANOVA-representation, from which we should decide heuristically which network should be updated first.

2.2.3 Pragmatic approach

Idema (1996, p. 8) provides a more pragmatic approach, which needs more a priori knowledge. He proposes to design the reference paths in such a way that one target function is dominant for this movement. E.g. a cogging force function should be learnt at low velocity, in order to keep friction forces out of scope. The vulnerability of this approach lies in the fact that one doesn't necessarily know in advance how low this 'low velocity' should be, or in general, when a function is dominant or not. So provided that symmetrical paths can be found,the approach postulated in this thesis, based on more generally valid theory, is to be preferred.

Nevertheless, the approach by Idema is of additional value to our here-presented approach. In chapter 3 we will see that sometimes input signals cannot be chosen uncorrelated. In that case the approach by Idema is of concern, in order to reduce interference as much as possible.

A similar approach is presented by Steenkuijl (1999). Velthuis (2000, p. 163) calls this method *'rather heuristic in nature'*, since prior knowledge is required to a high degree. Steenkuijl (1999, p. 29) states that simultaneous training of several networks is not possible, since with one single path it is impossible to make one target function dominant. In case we should indeed not succeed to train with a single path, we could address the methods by Steenkuijl and Idema.

2.3 Noise and frequency behaviour

Generally a network can learn a function, as long as its input is correlated to its output. We know that white noise is not correlated at all. Now consider a white noise, added to a target function. From 2.2 we may now conclude that, provided the number of samples on each spline is large enough, the noise will not have any effect on a BSN, since it is not correlated to the input of the function.

If a spline is crossed at relative high speed, then a relative small number of samples is available. This implies that such a fast visited spline should be visited more often than splines crossed at lower speed, in order to guarantee a sufficient number of samples to filter out noise.

A frequency transfer function of a BSN is hard to give, if its input is not given by time (but e.g. in our case: by a reference path). This is considered irrelevant for this study, so it is left out of scope. Verwoerd (2000) gave an analysis on frequency behaviour of BSN's, but this analysis was valid only for BSNs with time as their only input variable. A frequency analysis is not considered here, but it is recommended for future research, since it necessarily has its impact on noise and stability considerations.

2.4 Stability

As stated before, stability is of minor concern in this study. Nevertheless, there is no point in creating an algorithm if its application is insecure. Therefore this topic is addressed briefly here.

Two kinds of instability are relevant in an LFFC. First there is instability due to dynamical behaviour of the learning loop. This one is excluded by the fact that learning is performed off-line only. During operation (be it in simulation or in experiment), the contents of the LFFC are left unaffected.

Second, there is instability in the network parameters, also known as divergence. It is not by definition to be the case, that network parameters converge at all. Verwoerd (2000) also addressed this in his thesis, but this only holds for time-indexed LFFC. Velthuis (2000) also addresses this problem. It is stated there, that a stability condition can be derived, if all inputs but one of the BSNs are constant. This is not the case in the present study. The paper by Velthuis, De Vries, Schaak and Gaal (2000) gives some stability conditions for spline widths, but in our case a problem arises from the fact that there is no equivalent time corresponding to the spatial spline width. Furthermore the condition on the learning rate postulated in this paper only holds for repetitive motions.

In our study, we decided to take a small learning rate, in order not to compromise stability.

2.5 Linear motor model

Two models of a linear motor have been used. The first is a simplified model. It consists of a moving mass with one degree of freedom. It incorporates only a cogging force and a non-linear friction model. The model is depicted in figure 2.2.

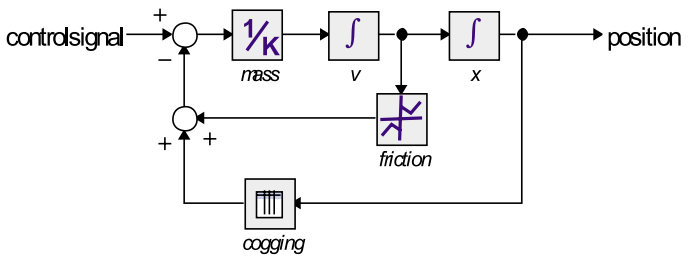

Figure 2.2: Second-order model of the linear motor, incorporating cogging and non-linear friction

The second model is more realistic: besides cogging and friction also commutation is incorporated. This second model is depicted in figure 2.3. The commutation is modelled as the multiplication of velocity with a special mapping from position to commutation. In section 2.5.3 this choice is explained. This model was simulated both with and without measurement noise.

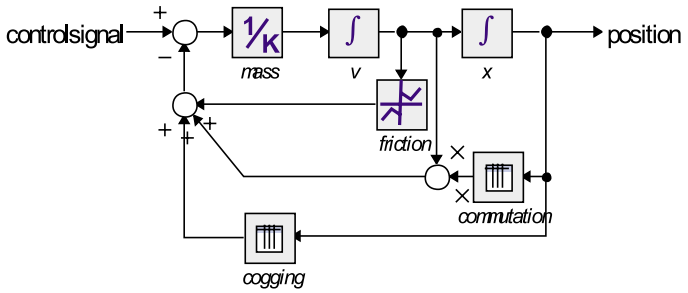

Figure 2.3: Second-order model of the linear motor, incorporating cogging, non-linear friction and commutation

In the sequel we will now only use r_1 for the reference position, r_2 for the reference velocity and r_3 for the reference acceleration. The subscripts for cost functions J, control signals u and signal approximations $\hat{u}$ are maintained accordingly.

2.5.1 Cogging

In reality a deterministic (though unknown) force results from the fact that the magnetic field in a linear motor is not homogeneous. If all magnets were placed perfectly, and all magnets were equal in strength, this so-called *cogging force* would be a periodic (sinus-like) function of the position. In reality, however, the magnets are not equal in strength, and their placement is not perfect. This brings about a relationship which is only nearly periodic, with variations in both its amplitude and its frequency. The modelled cogging force characteristic is depicted in figure 2.4.

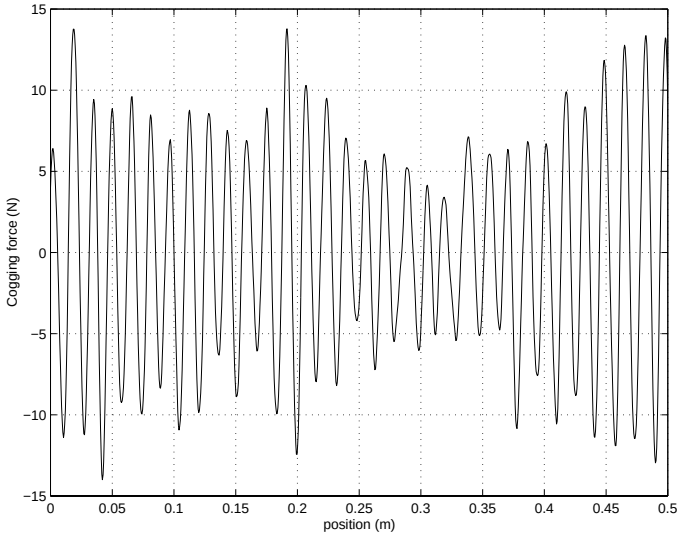

Figure 2.4: Cogging characteristic

2.5.2 Friction

To model the friction a modified Stribeck model was chosen. The parameters of this non-linear friction function are chosen rather arbitrarily. It is not chosen according to a real application, but rather such that it comprises a well-discernible non-linearity within the working range of velocity in the present model. Several representations of the effects described by Stribeck (1902) are known. In this study a modified version of the formula for friction from Spreeuwers (1999, p. 63) was used. In this formula all *signum*-functions were replaced by *tanh*-functions with a (sufficiently large) gain acting on the input, thus yielding the following formula:

$$F_f(v) = K\left(v + K_s \tanh(\alpha_t v) \cdot e^{\alpha_e v^2} + K_c \tanh(\alpha_t v)\right) \tag{2.27}$$

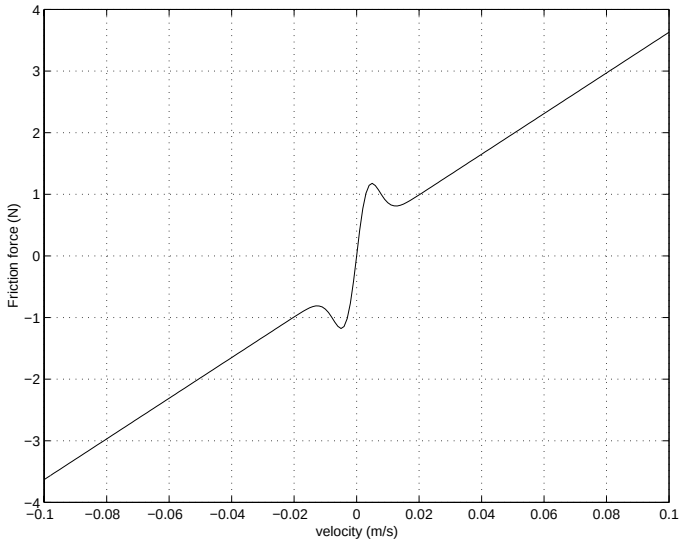

Figure 2.5: Friction characteristic

with $K = 33$, $K_s = 0.05$, $\alpha_e = 20000$, $K_c = 0.01$, and $\alpha_t = 200$. v is the velocity on which the friction depends. The friction force characteristic is depicted in figure 2.5.

The replacement of the *signum*-functions by *tanh*-functions was made to avoid a discontinuity around zero. In reality the friction characteristic comprises a step-discontinuity at zero, and a negative slope near zero. For larger velocities the friction approaches a linear function of speed. This replacement significantly increases simulation speed, at the cost of losing reality. Moreover, this modification simplifies the experiments: generally discontinuities are impossible for a LFFC to learn, since at low velocities the difference between the reference velocity and the true velocity will generally be large. Future research will be needed on learning with more realistic characteristics and dynamical friction models, to which a pass has been made by Spreeuwers (1999).

2.5.3 Commutation

As stated before, the commutation is modelled as a multiplication of the velocity and a special commutation table.

In reality, commutation is a switch of the magnetic field, actively brought about to make the linear motor work at all. This switch should be synchronized with the transitions through the fields of the permanent magnets. Due to imperfect placement and imperfect strength of the permanent magnets, perturbations may result from the poor synchronizing. From this, it follows that the perturbation depends on both position and velocity.

The commutation as a function of position and velocity is depicted in figure 2.6. The lack of reality was accepted at the benefit of homogeneity: with this value, all occurring values for internal forces are of the same order of magnitude, in this sense that their values are not negligible with respect to each other. The smallest maximum value is about 1 N, the largest about 15 N.

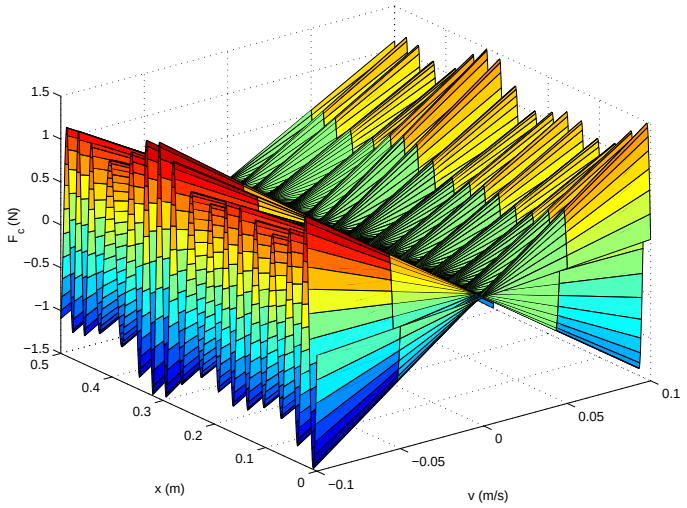

Figure 2.6: Commutation force

2.5.4 Noise

The real linear motor setup suffers from measurement inaccuracy. An accuracy of 10^{-6} m is guaranteed, which we interpret as uniformly distributed noise with a variance $\sigma^2 = \frac{1}{12} \cdot (10^{-6})^2$ (Van Amerongen and De Vries, 1999, p. 169). In our simulations the measurement noise was modelled as a gaussian noise with a standard deviation of 10^{-6} m, which is $\sqrt{12}$ times higher than with the uniform noise model. This margin was taken to guarantee sufficient excitation from the noise, since gaussian and uniform noise sources are not interchangeable straight away. Replacing a measurement inaccuracy by a noise signal is in fact incorrect, but it was the easiest way in this case. Finding more accurate solutions was not aimed within the current project time.

2.6 Parsimonious learning feed-forward control for the linear motor

2.6.1 Linear motor model without commutation

We now want to incorporate the theory of parsimonious networks to the previously described linear motor, neglecting the commutation force. As we can see from section 2.5, for the first (most simple) model the only useful effects to be learnt are cogging, friction and inertia. The latter can be interpreted as a coefficient in the force as a function of the acceleration. This sums up to three internal forces, each depending on only one input variable, namely the cogging depending on the position, the friction depending on the velocity, and the deviation of mass, resulting in a function of acceleration. We can now conclude from the ANOVA representation, that in this case three univariate BSNs suffice. No multivariate BSNs are needed.

Networks will learn the control signals as long as they are correlated to their inputs. For the position network this means that any part of the control signal which is correlated to the reference position will be learnt. The cogging force and the control signal act on the system in an identical way, namely as a force acting on the translator mass, i.e. on the input of the first integrator in figure 2.2. This means that the optimal feed-forward control signal is exactly equal to the cogging characteristic. The same goes for velocity and acceleration networks.

Now the only problem is the fact that the network uses the reference position as an input, whereas the learnt force depends on the real position. It is assumed here that this difference is small, because of the well-conditioned parameterisation of the PD-compensator. In a situation where this difference is not small enough, we might have a problem. We can imagine that this *does* occur with the velocity network when a discontinuous friction model is used: at velocities near zero, the deviation of acceleration (and to some extent of the velocity and position as well) will be relatively large.

2.6.2 Linear motor model with commutation

Now a model with commutation is considered. As argued before, the commutation is dependent on both the position and the velocity. From the ANOVA representation it follows that now a bivariate network is needed.

The main problem arising from this additional network is the fact that any signal related to the position can be learnt by both the position network and by the (position, velocity)-network. The same goes for signals related to the velocity. This means that the physical composition does not necessarily equal the composition made by the parsimonious BSN. This requires special care with reference paths. Other aspects as stated for the motor model without commutation, also hold for the motor model with commutation.

2.7 Network choices and cost functions

In order to evaluate simulations, first some criteria have to be formulated with respect to which the simulation results have to be compared. For each network a different criterion has to be specified. We briefly address them here.

2.7.1 Position network

The position ranges from 0 to 0.5 m. The cogging force was modelled by a linear interpolation table with 1000 entry points. It comprises 32 'cogging periods', so the rule of the thumb to take about 15 splines per period yields a number of 500 splines for the position network. The cost function was defined as:

$$J_1 = \frac{1}{N}\sum_{n=0}^{N}\left(u_1(\frac{n}{N}\cdot 0.5) - \hat{u}_1(\frac{n}{N}\cdot 0.5)\right)^2 \tag{2.28}$$

N was chosen 49900. This equals 100 summation points for each spline. This rather large number invokes long calculation times, but smaller numbers turned out not to yield reliable cost functions.

For the present configuration this cost function is lower-bounded by $J_{1,min} = 1.0\cdot 10^{-3}$. This was found by learning from a manipulated data set, which consisted of cogging values exactly on these 49900 summation points. Due to its homogeneity and high density, this set is considered persistent, i.e. it contains the maximal amount of information in the most unambiguous way.

2.7.2 Velocity network

The velocity ranges from -0.1 to 0.1 m/s. The friction force was modelled as a continuous function, and learnt with a BSN containing 35 splines. This number is the result of some preliminary tests: a smaller number does not enable the visibility of the nonlinear properties near zero (only a linear function with an offset was visible, then), whereas larger numbers put higher demands on training sets in order to update all weights sufficiently accurately.

The cost function was defined analogously:

$$J_2 = \frac{1}{N}\sum_{n=0}^{N}\left(u_2(\frac{n-N/2}{N}\cdot 0.1) - \hat{u}_2(\frac{n-N/2}{N}\cdot 0.1)\right)^2 \tag{2.29}$$

Here N was chosen 1400, which again equals 100 summation points per spline.

Following the same procedure as for the position network, we find that for the present configuration, with r_2 ranging from -0.1 to 0.1 m/s, J_2 is lower-bounded $J_{2,min} = 0.85 \cdot 10^{-3}$.

It is worth investigating if a non-homogeneous spline distribution yields solutions with a higher performance: the non-linearity in the target-function is found near the origin only, whereas for larger values the function is nearly linear. This could inspire us to apply a high spline density for small values, and a lower density for larger values, thus needing less splines in general. The current learning algorithm as implemented in Matlab can only deal with homogeneous distribution. And besides, the goal of this project was to find solutions in general terms. Addressing the distribution uses a priori knowledge, and thus loss of generality. On the other hand, it is worthwhile to use knowledge once that it is available.

2.7.3 Acceleration network

The acceleration ranges from -0.1 to 0.1 ms^{-2}. In the present simulations the only property to be learnt by the acceleration network is the inertia. In this case the force is a linear function in the acceleration. Then two splines should suffice. However, the present learning algorithm can handle odd symmetry only if an odd number of splines is used, so a number of three was the result. In this case N=200 suffices in the equation:

$$J_3 = \frac{1}{N} \sum_{n=0}^{N} \left(u_3(\frac{n - N/2}{N} \cdot 0.1) - \hat{u}_3(\frac{n}{N} \cdot 0.5) \right)^2 \tag{2.30}$$

This again equals 100 summation points per spline.

This cost function is lower-bounded by zero, because the force due to inertia is supposed to be a linear function in the acceleration. Since it also crosses the origin (no offset), it can be perfectly approximated by a BSN with three splines, of which the middle one has zero weight.

2.7.4 Position-velocity network

As argued before, the commutation force was modelled as a multiplication of the velocity with a special commutation table. The commutation table was generated with the same number of local maxima as the cogging function, so on the position axis the same number of splines, i.e. 500, was chosen. On the velocity axis the function is linear, so three splines would suffice. However, to be able to see eventual nonlinearities, a number of 5 was chosen. This means that a 5×500 splines bivariate network is needed. Given the fact that this yields 2500 splines, a wish for 100 summation points per spline *on each direction* would yield a $(5-1)\times(500-1)\times100\times100 = 19,960,000$ summation points. This large number implies long calculation times. This convinces us to give in a significant amount on accuracy. Since even 10 summation points

per spline took about an hour to calculate the cost function, we decided to take only 1 summation point on each spline, on each direction. This choice was supported by the fact that Matlab matrix-operations can now be used, which saves a lot of time, too. In this case the cost function is represented by:

$$J_{1,2} = \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \Big(u_{1,2}(\frac{n}{N} \cdot 0.5, \frac{m - M/2}{M/2} \cdot 0.1) - \hat{u}_{1,2}(\frac{n}{N} \cdot 0.5, \frac{m - M/2}{M/2} \cdot 0.1) \Big)^2 \tag{2.31}$$

with N=500 and M=5. This cost function is lower bounded by zero, because of the following. On the velocity axis the function is linear, and can thus be approximated perfectly. In the position direction the summation points of the cost function exactly coincide with the interpolation knots of the BSN. This means that on every summation point, the error can become zero, thus yielding a zero cost. This is not necessarily the optimal solution, but that is the cost of increasing the speed at which the cost function is calculated.

Chapter 3

Simulation

In this chapter the simulations performed in this study are discussed. First the design of the simulations is discussed. This concerns learning speed, path properties and spline distributions. Then the first simulation is discussed, which concerns the verification of principle 2.4 (learning order). The second trough fourth simulation concern respectively the linear motor without commutation, with commutation, and with both commutation and measurement noise.

The apparently strange naming of the paths was due to history: several more paths have been tried, and the resulting names correspond to computer files.

3.1 Simulation design

In section 2.5 a detailed description of the linear motor model was given, so it is not repeated here. The models of figure 2.2 and figure 2.3 were embedded in a control loop containing a PD-compensator and a learning feed-forward controller as shown in figure 1.3.

Simulations were performed in the simulation program 20-Sim. This program is developed by Control Lab Products, a spin-off company from the Control Laboratory, University of Twente. 20-Sim provides numerous possibilities to implement models, such as bond graphs, block diagrams and iconic diagrams. In this study only block diagrams were used.

Each experiment contains only 5 iterations, because the capacities of the personal computer, on which 20-Sim and Matlab where running, proved to be limiting. Many iterations together invoke a lot of computing capacity. More iterations would be necessary if convergence issues were of concern.

3.1.1 Learning speed and convergence

For time-indexed LFFC, stability criteria are known (Velthuis et al., 2000). From this we may assume that for path-indexed LFFC a learning rate chosen too large may cause divergence of the network weights, although formal proofs are not found in literature. An expression for the upper bound on a stable learning factor is not known, so we propose to choose the learning factor rather conservatively, and then increase it until divergence occurs. In this project it was found that a learning factor of 0.2 suffices for most situations.

3.1.2 Regularisation

During preliminary experiments, it was observed that sometimes the cross-correlation matrix of the velocity network was badly conditioned. Therefore it was decided to apply regularisation to the velocity network during all simulations, in order to have comparable conditions for all simulations. As argued in chapter 2, the regularisation matrix was chosen $1.0 \cdot 10^{-4} \cdot I$, with I the unity matrix. Here a safety measure on parameter divergence was taken at the cost of learning accuracy. During the preliminary experiments, large weights did not occur in the position and acceleration networks, so regularisation was not applied to them.

3.1.3 Paths: order and coverage

A reference path $\mathbf{r}$ is called *persistently exciting* for a linear system, when the following holds for all parameters (Löhnberg, 2000) (notation to comply with the rest of this study):

$$p(x|\theta_1)_r \neq p(x|\theta_2)_r \Leftrightarrow \theta_1 \neq \theta_2 \tag{3.1}$$

where x is a system state, θ_i an arbitrary value of parameter θ and r the reference path. To say the same in words: the signal $\mathbf{r}$ should be such, that we can tell from measurements, what the system parameters are, with acceptable uncertainty. With this in mind, we formulate the following demands on the reference signal for our nonlinear system.

As argued before, four properties of the linear motor system should be learned: inertia, cogging, friction and commutation. The first three properties all depend on one system state only (acceleration, position and velocity, respectively). The last one depends on two of them (velocity and position). The objective is to learn a control signal, for phenomena each depending on one or more of these system states. So for learning the cogging, we should cover the entire position domain, for learning the friction the entire velocity domain, and for learning the inertia we should cover the entire acceleration domain. For learning the commutation, we should cover the entire (position,velocity)-domain, i.e. every position should be passed with every

relevant velocity. For covering the entire acceleration domain a third order signal (i.e. a position reference of which the third derivative, the so-called *jerk*, is nonzero) is needed. By proper manipulation we can shape the signal such that it also covers the position, velocity and (position,velocity)-domain. One could argue that several second order paths with different accelerations may also suffice to cover the acceleration domain sufficiently. This is correct, but a larger number of movements is needed in that case. Therefore a third-order movement is considered to be more efficient.

The (position,velocity)-domain cannot be covered completely (only think of the fact that at the edges of the position domain, velocity will generally be small). The effect of this only becomes manifest if references are prescribed which use parts of the network that have not been trained. So training references should cover at least those parts which are relevant for future references.

The reference should be created such that the signal can be followed well by the control system. Hence a robust control system is needed already before learning takes place. In simulation a model was used which was provided with the software (linear motor model from 20-Sim), which has a well-conditioned controller. This topic is not considered in this thesis.

3.1.4 Paths: symmetry

The principles in section 2.2.1 allow us to state, that generally a reference path comprising a high grade of symmetry, enables the networks to learn well. A simple example of a reference path is depicted in figure 3.1. The upper three curves represent respectively the position r_1, velocity r_2 and acceleration r_3 as a function of time (transient representations), the lower three curves represent the velocity as a function of position, the acceleration as a function of position, and the acceleration as a function of velocity (phase-plane representations).

From now on, let the target function of BSN1 be denoted with u_1, and u_2 and u_3 be defined accordingly. Notice that these functions are not explicitly available, but are only defined as 'the optimal feed-forward control signals to compensate for respectively cogging, friction and inertia'.

We see that the (r_1, r_2) phase-plane is symmetrical in both its vertical and horizontal axes. The same goes for the (r_2, r_3) phase-plane. Using forced symmetry, we can now guarantee that the learning of $u_1(r_1)$ will not interfere with the learning of $u_2(r_2)$ and vice versa, and the same goes for the learning of $u_2(r_2)$ and $u_3(r_3)$. A problem arises when we look at the (r_1, r_3)-plane. There is no symmetry, so the premises of the principles in section 2.2.1 are no longer applied to. Therefore we cannot conclude that $u_1(r_1)$ and $u_3(r_3)$ will not interfere in their mutual learning. Of course it still can go well, but this will not be by means of the postulated principles.

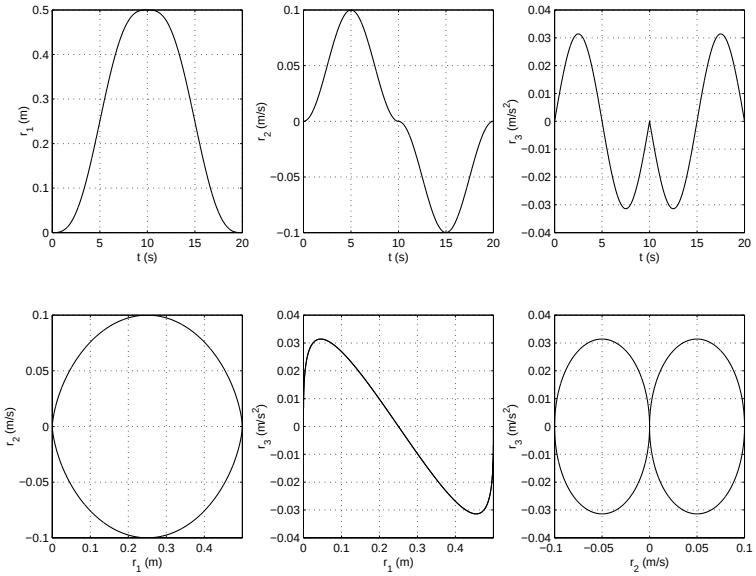

Figure 3.1: Example reference path

3.1.5 Spline distributions

The spline distribution of a network is influenced by the accuracy at which a target function should be approximated. Furthermore the available amount of learning data and its distribution put limits on the spline density. For the simulations in this chapter the splines were chosen in such a manner that both the learning and the approximation were expected to be appropriate: a trade-off was made between high precision of the approximation (i.e. a large number of splines), and a high training speed (i.e. a low number of splines). Besides, a spline density chosen too high may result in instability because of a lacking high frequency suppressing (this is again hard to prove for path-indexed LFFC, but the analysis by Velthuis (2000) on time-indexed LFFC should guide as a warning). The splines were distributed homogeneously, with the number of splines as described in section 2.7:

Network	**Number of splines**
r_1	500
r_2	35
r_3	3
(r_1, r_2)	500×5

3.2 Simulation 1: Optimal learning sequence

The first simulation was performed to test principle 2.4. The model used is the complete linear motor model, yet without measurement noise. A very rich path was used (path $P9$, see the description in section 3.4; the exact properties of the path are not of importance, so the path is not discussed here), in order to gain sufficient information. The only important property of the path is that the velocity is statistically uncorrelated with both the position and the acceleration, whereas the position and acceleration do have a correlation. The position-velocity network was not trained at all, because it should be the last one to be trained: in order to have the optimal decomposition (see section 2.2.2, where the desired decomposition was discussed), we want the maximum information to be stored into the univariate networks.

Only one run was performed, after which the cost of the various networks was evaluated. Learning was performed in different orders, and every time subsequent networks used the residue of the preceding networks. One could argue that performing only one run is not enough for filtering out the feedback controller dynamics, but the differences after one run were already so significant, and simulation times that long, that this was thought to be acceptable.

After one learning episode, the costs for different orders were:

Order	$\mathbf{J_1}$	$\mathbf{J_2}$	$\mathbf{J_3}$
1 2 3	0.74	$1.2 \cdot 10^{-3}$	0.31
1 3 2	0.74	$1.2 \cdot 10^{-3}$	0.31
2 1 3	0.74	$1.2 \cdot 10^{-3}$	0.31
2 3 1	0.021	$1.2 \cdot 10^{-3}$	$0.33 \cdot 10^{-3}$
3 1 2	0.021	$1.2 \cdot 10^{-3}$	$0.34 \cdot 10^{-3}$
3 2 1	0.021	$1.2 \cdot 10^{-3}$	$0.34 \cdot 10^{-3}$

where 1 corresponds to position, 2 to velocity and 3 to acceleration. From these results we may conclude that principle 2.4 holds in practice. First of all, it is clear that the cost of the velocity network (with cost function J_2) is not influenced at all by the learning order, nor does the sequence number at which the velocity network is trained influence the other networks. This corresponds to the fact that the velocity is not at all correlated with either acceleration or position. Second, it is clear that *both* the position network (cost function J_1) and the acceleration network (cost function J_3) are trained best if the acceleration network is trained before the position network is learned. Because of the smallest number of splines, we expect the acceleration network to have the largest generalisation ability, and indeed it yields the best results if it is trained first. And of course, if the first network is trained better, the residue available to the subsequent network is 'cleaner',

so learning will be performed better as well. The slightly smaller value of J_3 when network 2 is trained first (fourth row in the table), is considered to be insignificant. Nevertheless a future investigation may be worthwhile to find out, whether there is a foundation to say that networks with an uncorrelated input should be trained first.

Tests with noise have not been performed, but from section 2.3 we may expect noise not to have any influence on the optimal order, nor does the order influence the sensitivity of the LFFC to noise. This statement is supported by the outcomes of simulation 4, where it is shown that the influence of noise on BSNs is small.

3.3 Simulation 2: Simple LM

Three paths have been applied to the simplified linear motor model. In each of these three simulations the conservative learning factor of $\gamma = 0.2$ was used. In all simulations the optimal learning order according to principle 2.4 was used, i.e. first the acceleration network was trained, then the residue was used to train the velocity network, and finally the second residue was used to train the position network.

Path P1

Firstly reference path $P1$, see figure 3.2, was applied. It is seen from the figure that in the (r_1, r_3)-plane there is no symmetry (besides the point symmetry with respect to the origin, but in this study this is not a relevant type of symmetry). With this path we expect the according networks to learn badly, and the r_2-network to learn better.

Path P2

Secondly the path $P2$, depicted in figure 3.3, was applied. We can see that the (r_1, r_3)-plane is still not symmetrical, but at least it is covered more homogeneously. We expect this path to perform better than the previous path $P1$.

Path P4

Thirdly the path $P4$ was applied. It is shown in figure 3.4. This path is mostly symmetrical, but a little disorder is introduced: the points with zero velocity do occur at different positions, instead of several times on the same place. The philosophy is as follows. We should note the fact that the acceleration network has three splines on its entire input range, whereas the position network has 500 splines. This means that some given tracking error will generally be relatively large compared to the B-spline width of the

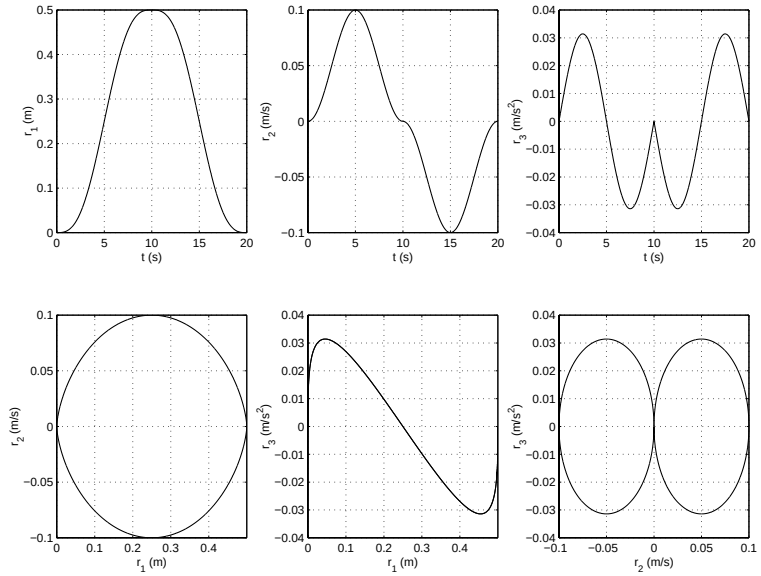

Figure 3.2: Simulation reference path P1

position network, whereas it will be relatively small compared to the spline width of the acceleration network. (We should not neglect the fact that generally the tracking of the position will be better than the tracking of the acceleration, but the difference in spline density is considered large enough to overcome this problem.) With this path, a certain value of r_1 occurs with very different values of r_3. It is now more probable (but not certain!) that the effect on which principle 2.2 was based occurs. We expect this path to have the best learning capabilities.

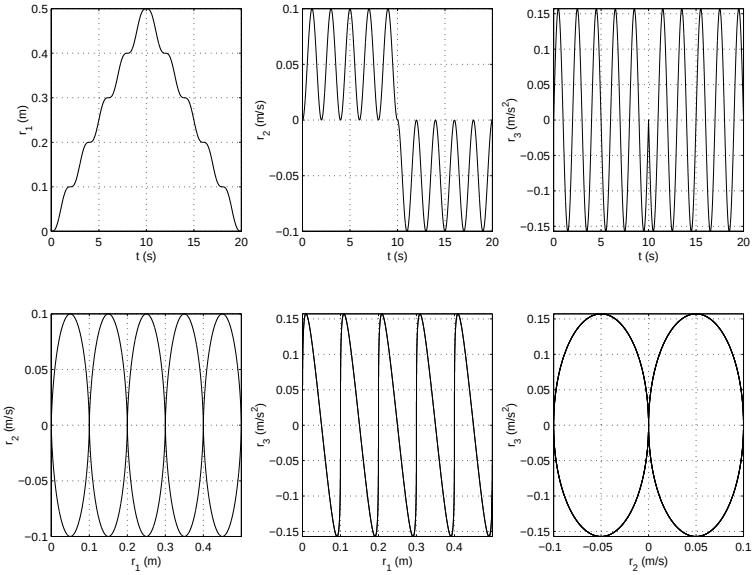

Figure 3.3: Simulation reference path P2

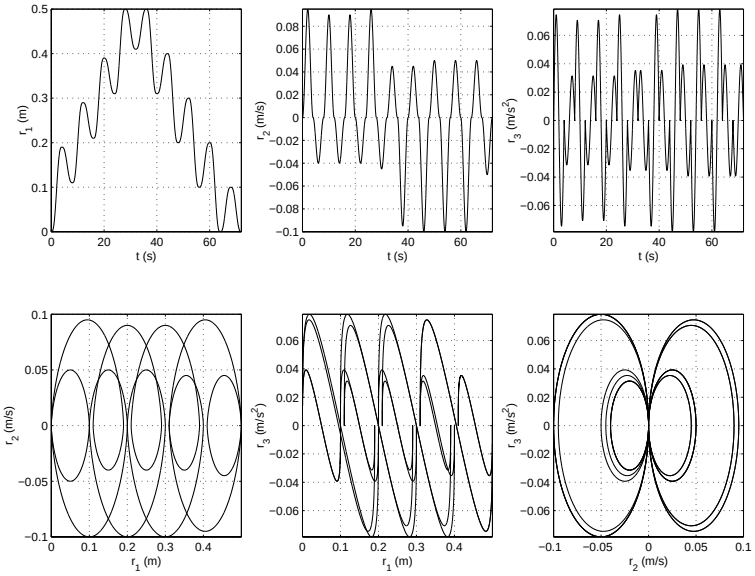

Figure 3.4: Simulation reference path P4

Results

In figure 3.5 the results of simulations with these three paths are presented. The upper left picture represents the cost of the position network as a function of the iteration step. The upper right represents the cost of the velocity network, and the middle left picture represents the cost of the acceleration network. Their values cannot be compared mutually. The lower left figure depicts the RMS-value of the positional error. The lower right represents the RMS-value of the output of the PD-compensator.

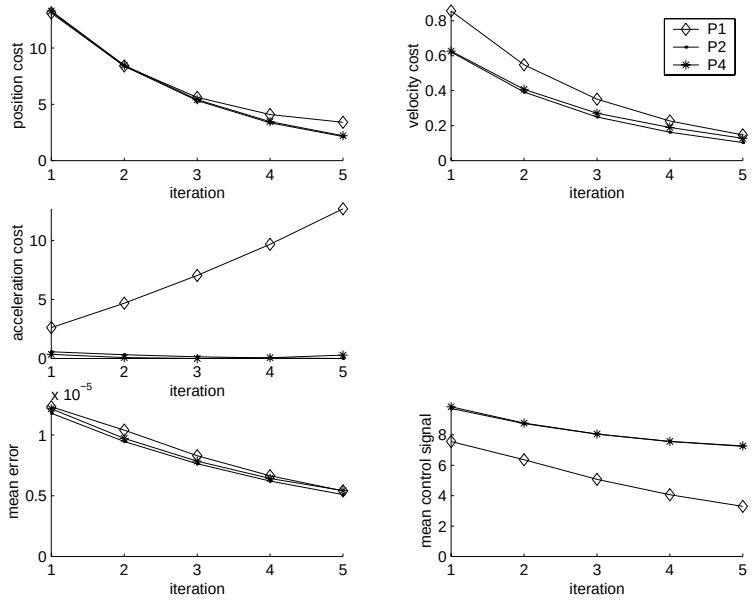

Figure 3.5: Results of simulation 2

From the position network cost we can conclude the following. For a small number of iterations, say up to three, there is no difference between the network performances of the paths. But when we evaluate after the fifth iteration, we see that the cost of path $P1$ is about 50% higher than the other two paths. Initially the position network converges with the same speed, but after some time it proves to converge to sub-optimal values.

The velocity network shows different results. Here path $P1$ starts slightly worse, but seems to converge to the same optimal cost. After 5 iterations we see that path $P4$ yields a 25% higher cost than $P2$, which is contrary to our expectation.

In the description of path $P4$ it was stated that the introduction of a little disorder increases the probability of principle 2.2 to hold. This does *not* hold for the acceleration network, and the price is a decreasing performance of the position network. The result is that on a certain position the different

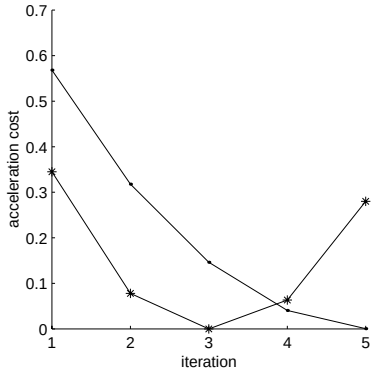

Figure 3.6: Zoom of third plot in 3.5

values of the acceleration do no longer add up to zero. We see here that the principles result in worse network contents rather than improving it. The deterioration is apparently small in the phase-planes, but the reasoning sound plausible, since even a small perturbation can be significant, due to the small spline dimensions.

The cost on the acceleration network shows a clear development. For clarity, a zoom of the lower two curves is depicted in figure 3.6. The path $P1$ causes the network to diverge. Apparently the path delivers insufficient information on the inertia, causing network to provide an over-correction, which needs compensation from the feedback controller. This inspires us two qualify the path as badly conditioned.

In the last graph we see, that the feedback control signal decreases for every path, even for paths that are considered to be training improperly. We should probably explain this from the fact that a neural network can always approximate a certain data set. In casu this means that a path can always be learnt to some extent, thus reducing the feedback control signal. This thought is supported by the fact that the positional error is decreasing as well. From this we should conclude that both the error reduction and the feedback control signal reduction on the path itself are *not* appropriate to evaluate the quality of learning. To overcome this, a validation path should be used. However, in the current simulations this was not needed, since the network contents can be compared with the theoretical optimal values. This is a global criterion, whereas a validation path is still subjective in this sense, that it may not necessarily test the quality of learning in an exhaustive way.

The reason why especially path $P1$ yields the best feedback control signal reduction, is probably found in the fact that this is the simplest path, so its control signal can be learnt the best.

We should now explain *why* a certain path can cause divergence. In figures 3.7 and 3.8 the phase plane of the reference acceleration and the

measured acceleration are plotted for path $P1$ and $P4$ respectively.

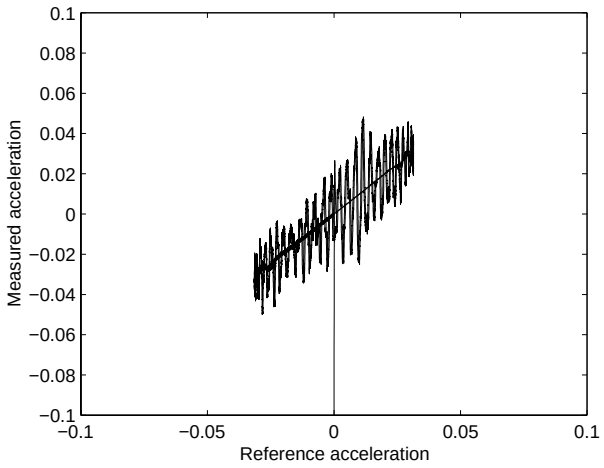

Figure 3.7: Reference acceleration versus measured acceleration, path $P1$

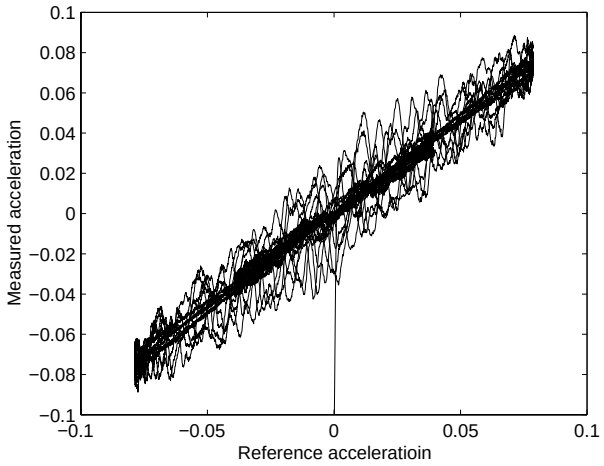

Figure 3.8: Reference acceleration versus measured acceleration, path $P4$

We see that for path $P1$ the relative difference is significantly larger than for path $P4$. Furthermore for path $P4$ the amount of data located close to the central diagonal is larger. We know that a phase shift being too large can cause divergence, although in case of a non-linear system and non-periodical signals phase shift is not a well-suited concept. Nevertheless we can say that a (relative) difference between reference and measurement being too large, can cause the same troubles we find when too much phase shift occurs. Actually, it is a premiss of LFFC that this difference is small

enough. We may say that this difference is too large with path $P1$, and appropriately small with path $P4$, although a sharp distinction cannot be made in this case.

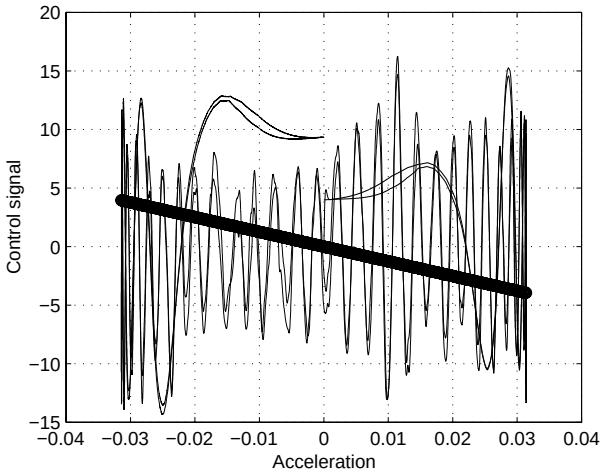

Figure 3.9: The control signal versus the reference acceleration, path $P1$

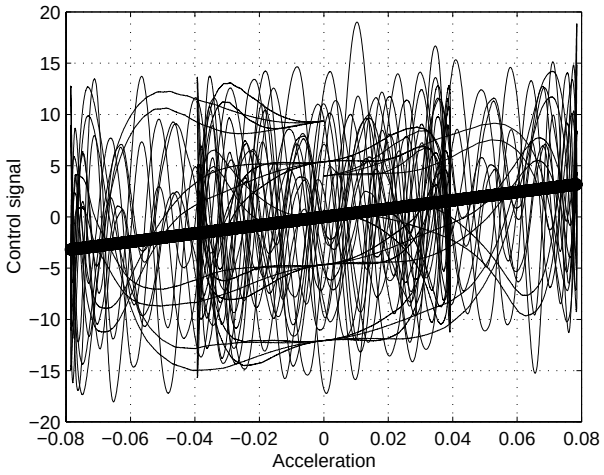

Figure 3.10: The control signal versus the reference acceleration, path $P4$

This reasoning is supported by figures 3.9 and 3.10. In these figures the control signal is plotted versus the reference acceleration. The central bold line represents the average as learned by the acceleration BSN in the first episode. We see that in case of $P4$ the bold line neatly has a slope approximately equal to the mass of the translator (37 kg), whereas in case of $P1$ even a negative slope is found. This indicates that the difference between

the reference acceleration and the measured acceleration is too large. We should conclude that the premiss of LFFC, i.e. that the difference should be small enough, is not met. This is due to the relatively small accelerations in path $P1$, thus the deviation resulting from the cogging and friction being relatively large. So we should question whether it is the poor quality of the path (only the extent to which it meets the principles is meant here), or merely the fact that the acceleration domain is visited sparsely (which is a quality of the path as well).

Concludingly we can say, that path $P2$ is the best, because it does not seem to diverge. Its performances are only slightly better than path $P4$. The reason is to be found in the disorder as addressed before, which first was thought to improve performance. Furthermore path $P1$ is a poor path, as expected. However, the expectation was only based on the principles in chapter 2, whereas it was indicated here that some other properties may have serious influences on convergence too, in casu the error of the acceleration being too large.

3.4 Simulation 3: LM with commutation

In this simulation the linear motor model is extended with the commutation. Now a bivariate BSN is needed.

Path P9

First the path $P9$ depicted in figure 3.11 was applied. In preliminary tests, it was found that this path, when applied with learning factor $\gamma = 0.2$, gave acceptable results.

The first part of the path (up to $t = 70$ s) consists of cycloids. This part is meant to deliver information for the univariate networks. The second part (from $t = 70$ s) consists of movements at constant velocity (third order acceleration, i.e. the third derivative (jerk) of the position is a piece-wise constant function of time). This second part is supposed to deliver information for the bivariate network: all positions are passed at three velocities (each velocity occurring both in positive and negative direction). This is exactly enough to cover all 5 splines in the velocity-direction of the bivariate network.

We expect this path to learn well, because it does meet the requirements for principles 2.1, 2.2, and 2.3 to a high extent.

Path P9b

The second path applied, $P9b$ (figure 3.12), consists of only the first 70 seconds of $P9$. This means that the explicit training of the bivariate network

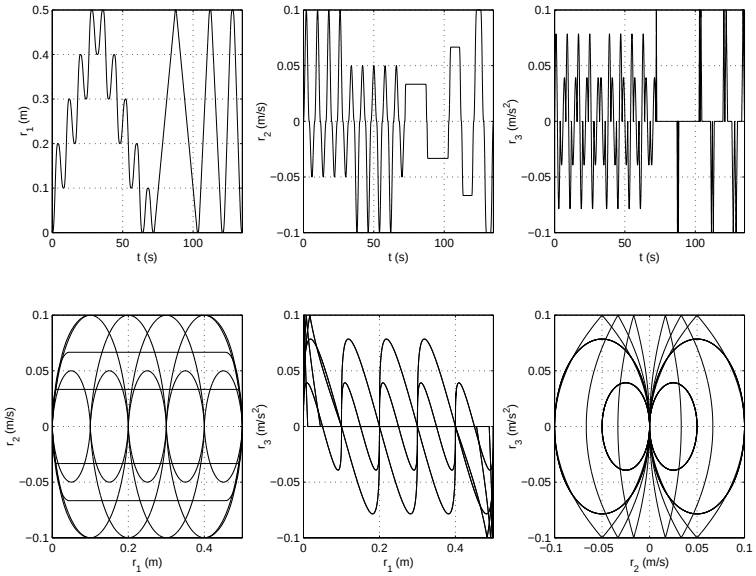

Figure 3.11: Simulation reference path P9

is omitted. Since the movement still covers a large part of the (r_1, r_2)-plane, we may still expect the network to learn a bit. Because of the nonzero acceleration at this part, we expect the interference into the bivariate network from the acceleration network to be filtered out worse. We expect this path to train moderately well.

Path P9c

The third path applied, $P9c$ (figure 3.13), consists of the first 35 seconds of path $P9$. This path does hardly comprise symmetry in the phase plane representations. Furthermore it does not cover the entire (r_1, r_2)-domain. We expect this path to train badly, so it is performed in order to investigate whether the principles really make a difference.

Path P1

The fourth path applied, $P1$, is the same as the first path applied to the linear motor model without commutation (see figure 3.2). We expect this path to perform worse than the previous paths, because the signal is less rich: there are not sufficient data points to train the acceleration network properly. The point in carrying out this simulation, is to be found in the fact that it does not comprise stagnations within the working range of the linear motor, only at its boundaries. Since stagnations may result in dirty control signals (because of the nonlinear friction), it is possible that the lack

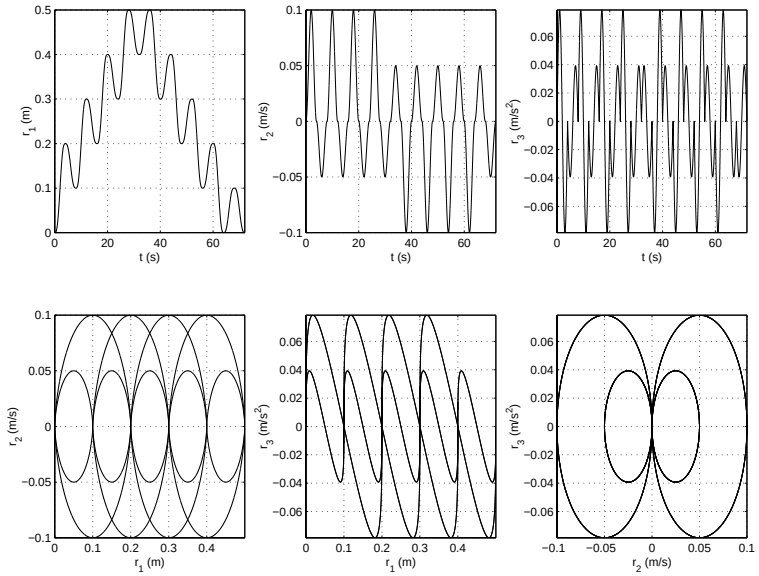

Figure 3.12: Simulation reference path P9b

of stagnations yields a better training, even when the principles are poorly met.

Results

In figure 3.14 the results of this simulation are represented identically, except for the fact that the middle right graph now represents the development of the cost of the position-velocity network as a function of the iteration. The graphs now represent respectively: the position network cost, the velocity network cost, the acceleration network cost, the position-velocity network cost, the RMS value of the error, and the RMS value of the output of the feedback controller, each of them represented as a function of the iteration step.

We see a nice convergence of the cost on the position network. Path $P1$ performs the worst, the others perform equally well. From this we can already dispose of our figuring, that the lack of stagnations may outperform paths that may even better meet the requirements for the principles. The lack of symmetry in the (position, acceleration) plane (see fifth graph in figure 3.2) probably causes interference, thus giving a worse performance.

In the velocity cost, we again see a convergence. What catches the eye, is the fact that path $P1$ joins the best performers, whereas now path $P9c$ performs the worst. The latter is in accordance with our expectation. The former is rather surprising, since the most important range of the velocity network is near stagnation (this is the most intricate part of the friction

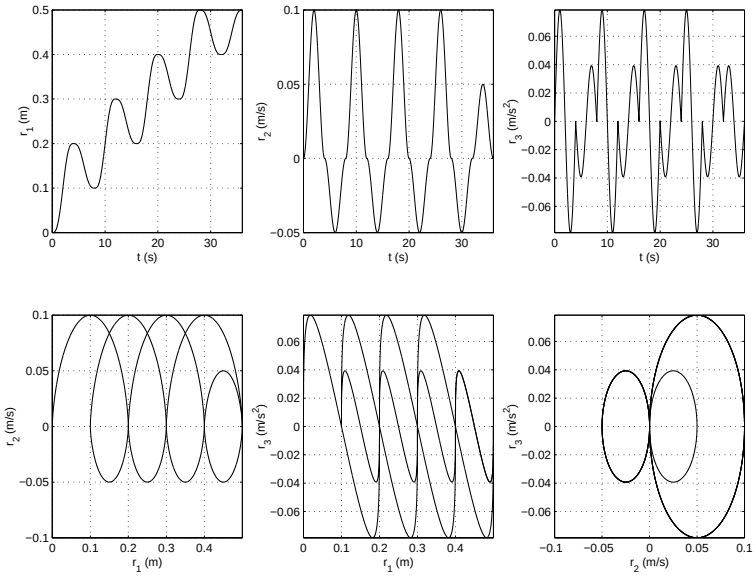

Figure 3.13: simulation reference path P9c

characteristic), which is now badly visited. What we have here, is a nice confirmation of principle 2.1: even although the velocity domain is not covered with a large number of different positions or accelerations, still the symmetry neatly has the effect it is suspected to have.

In the cost of the acceleration network, we see a divergence on path $P1$. The explanation for this is exactly the same as given in the interpretation of simulation 2. We see that $P9$ and $P9b$ perform equally well, and both go to the low value of $4 \cdot 10^{-4}$. This is not surprising, since the part with linear movements of path $P9$ is supposed to influence the velocity network only. In figure 3.15 it is seen that path $P9c$ causes divergence at the acceleration network

What really bothers, is the fact that after a few iterations, the cost on the position-velocity network starts to increase again. Probably for such a complex network (i.e., a large number of splines, hence a relatively small number of samples for each spline), the learning rate is still too large. Because of the simulations taking much time, a new simulation with even a lower learning rate was not performed. Further investigation is needed on this point.

Secondly one could argue that, like described for the acceleration network in the previous simulation, some difference between the references and the according system states may be too large to enable convergence. It will now be argued, that this is *not* the case. In figure 3.16 the phase-plane for path $P1$ is shown. The second and third graph show zooms on the same graph,

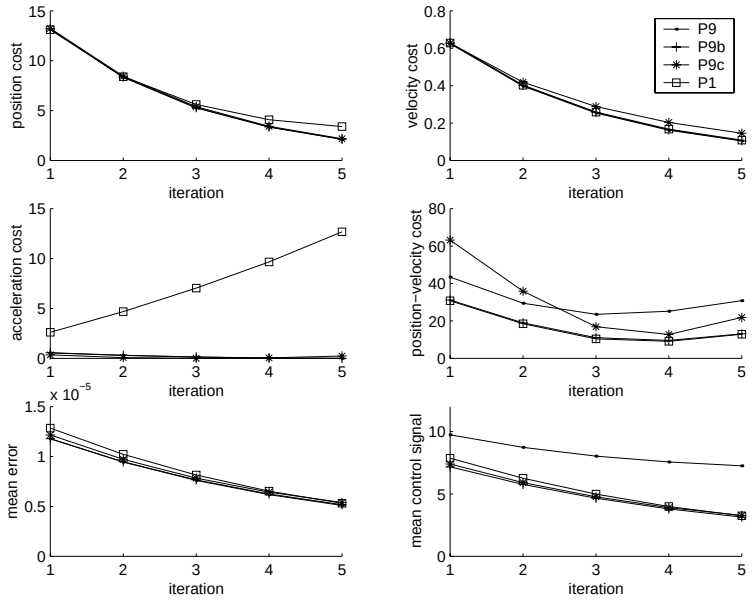

Figure 3.14: Results of simulation 3

in the outer left and uppermost parts respectively. In the zooms the pins represent the difference between the reference (position, velocity)-pair and the measured (position, velocity)-pair. The head of the pin represents the reference, the tail the measured value. The differences visible in the zooms are the largest occurring ones. What we see is that the maximum error in the position is about $10 \cdot 10^{-6}$ m, and the maximum velocity error is about $0.5 \cdot 10^{-3}$ m/s, both of which are small compared with the spline dimensions. Even the handful of irregularly positioned pins in the outer left part of the first zoom, are small compared to the spline dimension of 1 mm by 0.05 m/s. So in this case the LFFC-premiss, i.e. that the error should be small compared with the spline dimensions, is justified.

A third explanation could be looked for in the fact, that poor learning abilities of the acceleration network cause the (position, velocity)-network to diverge. This was confirmed by an extra simulation, performed *without* an acceleration network. In this simulation only the paths $P1$ and $P9b$ were applied, on a set of 10 iterations. The results are displayed in figure 3.17. What we see now, is that with $P9b$ we have convergence on all networks, within the current iteration axis. It should be noted that the vertical axis of the (position, velocity)-cost does not start at zero, so the relative decrease of the cost is rather small. This is probably due to the fact that the acceleration is not learnt: the training set available for the (position, velocity)-network is 'polluted' with the acceleration-related forces.

Furthermore we see an increasing cost at the position network for path

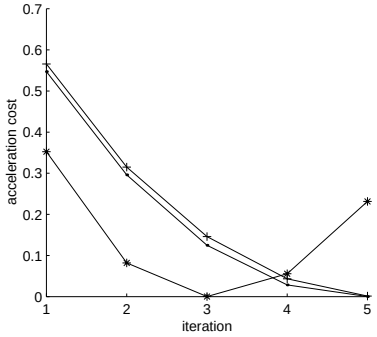

Figure 3.15: Zoom of third graph in figure 3.14

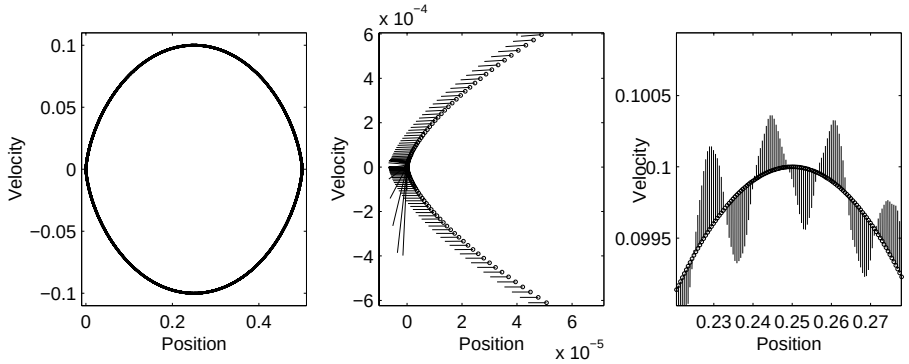

Figure 3.16: Phase shift in the (position, velocity)-plane with path $P1$

$P1$ after a number of iterations. It is is assumed that this was already the case in previous simulations, only it was not seen that time because of the shorter iteration axis. We should from now on assume that a poor path (i.e. $P1$) can cause divergence, and that a poor relation between the reference and a true state (i.e. the acceleration) can cause divergence as well.

Now we return to the original simulation. Perhaps even the most surprising, is the fact that with respect to the (position, velocity)-network path $P1$ beats all the others. This draws our attention, since the (position, velocity) network is not covered well at all (see figure 3.2). This implies that the network should not train well by any means. We should conclude that due to the small number of splines on the velocity axis (5) requires only a sparse coverage of the (position, velocity)-plane is needed.

In this simulation the same surprising feature occurs: the path which is considered performing the worst, yields the highest reduction of the control signal. We should follow the same reasoning as in simulation 2, i.e. that a simple path is easy to learn.

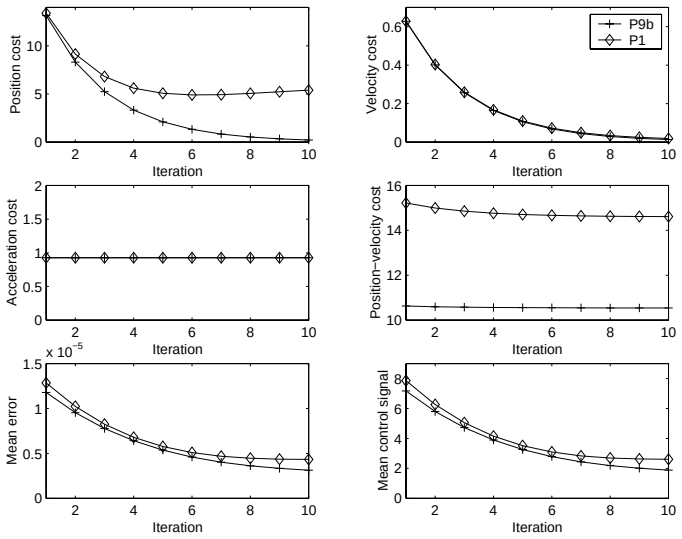

Figure 3.17: Simulations without acceleration network

The mean error and the mean control signal display a monotonous decrease for all paths.

For the position-velocity network, path $P9b$ performs the best, followed by $P9c$ at a 50% difference and $P9$ at a 100% difference. Apparently the linear movements at the end disturb the learning of commutation rather than supporting it. We currently do not have an explanation for this, so further investigation is needed.

Concludingly we can say that, according to our expectations, path $P9$ performs the best. Path $P1$ is worthless because of its divergence at the acceleration and position networks, but considering its other features, it performs surprisingly well. Furthermore, $P9b$ and $P9c$ perform slightly worse, according to our expectation. With respect to the result on the position-velocity network, we can say that linear movements are not as useful as we expected, and besides these prove to yield divergence for unknown reasons.

3.5 Simulation 4: LM with commutation and noise

In the last set of simulations, exactly the same experiments were performed as in simulation 3, only with a gaussian measurement noise, its standard deviation $\sigma = 1\mu$m chosen as explained in 2.5.4. The results are depicted in the usual manner in figure 3.18.

We only notice subtle differences with simulation 3. The interpretation given for simulation 3 practically holds for simulation 4. The only significant difference is found in the mean feedback control signal, which increases with

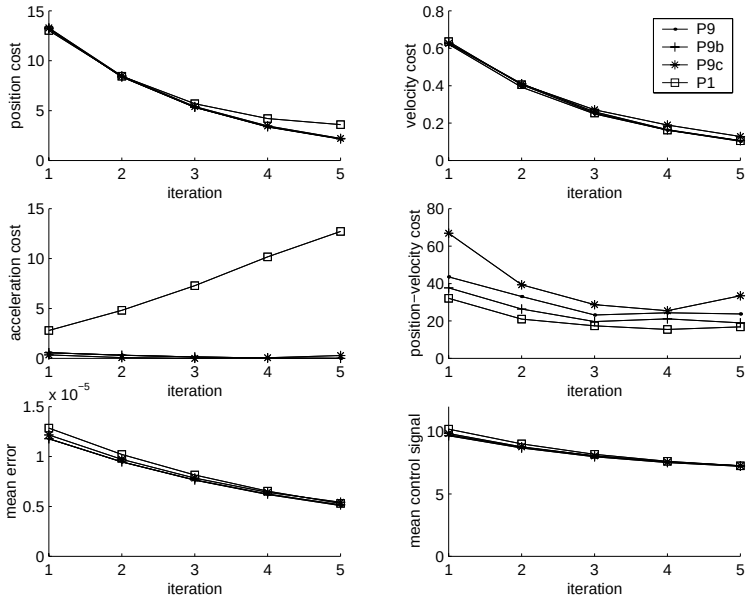

Figure 3.18: Results of simulation 4

amounts up to 25%. It is common that measurement noise increases the RMS value of a feedback control signal (its nett effect may still be zero, but this does not go for the power generated from it, which can be seen as frequency components added to the 'clean' steering signal). Apart from that, this experiment confirms our theory that noise hardly influences learning of BSNs: because of its lack of correlation, it doesn't have a nett effect.

One more small difference is that now path $P9b$ performs slightly better than path $P9$ with respect to the (position, velocity)-network, in accordance to our expectations. However, the difference is too small to judge on, and besides, this expectation was invalidated by the simulation without noise.

In figure 3.19 the curve of $P9$ and $P9b$ almost exactly coincide, so they are visible as a single line.

In this simulation again, we see that the path which is considered performing the worst, manages to reduce the feedback control signal the best. We may again conclude that the mean control signal is not very suited for judging on learning.

3.6 Discussion

In this chapter the results of the simulations have been discussed. Beside the positive outcomes, we should not neglect the following items, which need special care.

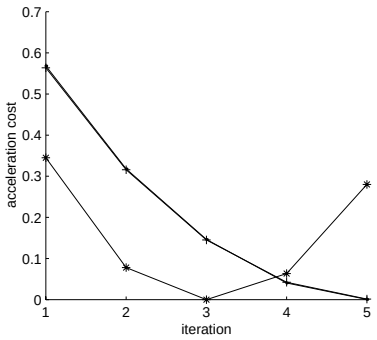

Figure 3.19: Zoom of third graph of figure 3.18

Firstly it was found that the order principle mainly holds. Nevertheless it is surprising that it turns out to be optimal (be it at only a small headstart) if the network with the uncorrelated function (i.e. the velocity network) is trained first, whereas in the principle it was stated that in this case the order of the network should not influence the quality of learning. An explanation should be looked for.

Secondly it was found that some paths cause divergence. Partly this was due to meeting the principles poorly. But certainly not less important, it was found that a path designed poorly, can cause the system not to meet the LFFC premiss, i.e. the error in the states should be small. Furthermore it was proven by others that the learning factor influences convergence. The lack of a well-formulated criterion puts a limit on the accuracy of the simulation design. A convergence analysis if path-indexed LFFC would brighten these problems.

Thirdly an explanation was not found for the fact that movements at constant speed disturb the learning of the (position, velocity)-network rather than supporting it.

Fourthly it was found that the reduction of both the mean feedback control signal and the error on the paths themselves are poor measures of the quality of learning. Using the cost functions for the various networks is better, but then the problem remains that the costs cannot be compared mutually, so an eventual trade-off cannot be made. In this sense a validation path would be of additional value.

Chapter 4

Experiments

In this chapter the experiments with the linear motor are addressed. The linear motor (LM) is made by Tecnotion. It is controlled with a personal computer with a Pentium-II 233MHz. First a brief description of the LM is given, followed by the tuning procedure for the feedback controller. Then the results of the experiments are given, and followed by a discussion.

In this chapter again names of the paths are due to history.

4.1 Tecnotion linear motor

The LM receives its thrust force from a set of coils, mounted above a grid of permanent magnets. A photograph of the lab setup is shown in figure 4.1. The translator of the LM approximately has a mass of 5 kg. The LM

Figure 4.1: The Tecnotion linear motor

can provide a thrust force of 930 N, which limited by the hardware. The maximum allowed acceleration is 10 ms^{-2}, the maximal velocity is 2 m/s. The LM has a free range of 0.74 m. Since the translator itself has a length

of 22 cm, and approximately 10 cm is taken as a safety margin on the left side, and about 2 cm on the right side, a working range of 40 cm was used in the experiments. The LM suffers from all features we have discussed before: inertia deviation, friction, cogging and commutation inaccuracies. Just like with the simulations, this means that we need three univariate BSNs, and one bivariate BSN. Unfortunately only univariate networks are available in the current software environment, so commutation inaccuracies cannot be corrected for. It was decided not to implement a bivariate BSN, since the available time was limited.

4.2 PID-design

In order to obtain the correct feed-forward signal, we first need to design a robust feedback control. A PID compensator was used, in series with a roll-off filter, such that the following transfer function was achieved (Coelingh, 2000, p. 76):

$$\begin{aligned} C_{pi}(s) &= K \cdot \frac{\tau_i s + 1}{\tau_i s} \\ C_{ds}(s) &= \frac{\tau_d s + 1}{\tau_d \beta s + 1} \\ C_h(s) &= \frac{1}{\tau_h s + 1} \\ C(s) &= C_{pi}(s) \cdot C_{ds}(s) \cdot C_h(s) \\ &= K \cdot \left(\frac{\tau_i s + 1}{\tau_i s}\right) \cdot \left(\frac{\tau_d s + 1}{\tau_d \beta s + 1}\right) \cdot \left(\frac{1}{\tau_h s + 1}\right) \end{aligned} \tag{4.1}$$

Here K is the proportional gain of the compensator, τ_i the integration time constant, τ_d the differentiation time constant, and τ_h the roll-off time constant. The PID was implemented on a computer. The discrete time conversion was performed as a backward Euler differentiation approximation, i.e. the Laplace-variable s was replaced with the discrete time transfer function as follows:

$$s \approx \frac{z - 1}{z T_s} \tag{4.2}$$

With this computer a sample time $T_s = 1$ ms was chosen. The main reason was to limit the number of sampling points which have to be stored. Of course control intervals and sampling intervals for off-line learning purposes do not necessarily coincide, but creating an 'under-sampling' method would require more implementation time than available. At this sample rate the amount of processor time during a sample interval proved to be sufficient. Furthermore the sample period limits the bandwidth of the control system. At the sample frequency of 1 kHz, a bandwidth of 350 rad/s (56 Hz) could be achieved. This proved to be sufficient to a high degree for the relatively

slow paths used, although at a higher bandwidth a higher active stiffness would probably reduce the error due to friction and cogging. The bandwidth ω_b was chosen from preliminary experiments: for a good performance, the bandwidth should be as large as possible, but it is limited by the control sample time. It is defined as *the frequency for which the sensitivity function S first crosses the 0 dB line from below* (Coelingh, 2000, p.78). The sensitivity function S is defined by:

$$S(s) = \frac{1}{1 + P(s)C(s)} \tag{4.3}$$

where $P(s)$ and $C(s)$ are the transfer-functions of respectively the plant (i.c. the ideal mass, see below) and the compensator. After calculating the controller parameters, a Bode plot of the sensitivity function will be given.

The PID parameters were assigned using the design rules formulated by Coelingh (2000). Here the LM is viewed to as an ideal mass, i.e. without friction and other forces. The following system parameters were chosen:

Parameter	**Value**	**Unit**
m	5	kg
ω_b	350	rad/s
ω_l	1	rad/s
β	0.1	
S_l	10^{-5}	

Here m is the mass of the translator, ω_b the bandwidth of the control system, ω_l the frequency below which signals are cut off, and S_l the maximum value of the sensitivity function below the cut-off frequency ω_l. The mass presented here is probably lower than the true mass, but this conservative estimate was chosen in order to avoid too high stiffness: the mass used for dimensioning influences the proportional gain linearly, which may cause instability if it is too large.

Then the parameters of the PID-compensator are calculated using Coelingh (2000, p. 86):

$$\tau_d = \frac{1}{2\omega_b\sqrt{\beta}} \tag{4.4}$$

$$K = 2\omega_b^2 m \left(\frac{\omega_b^2\tau_d^2\beta + 1}{\omega_b^2\tau_d^2 + 1} \right) \tag{4.5}$$

$$\tau_i \leq \frac{S_l K}{\omega_l} \tag{4.6}$$

$$\tau_h < \beta\tau_d \tag{4.7}$$

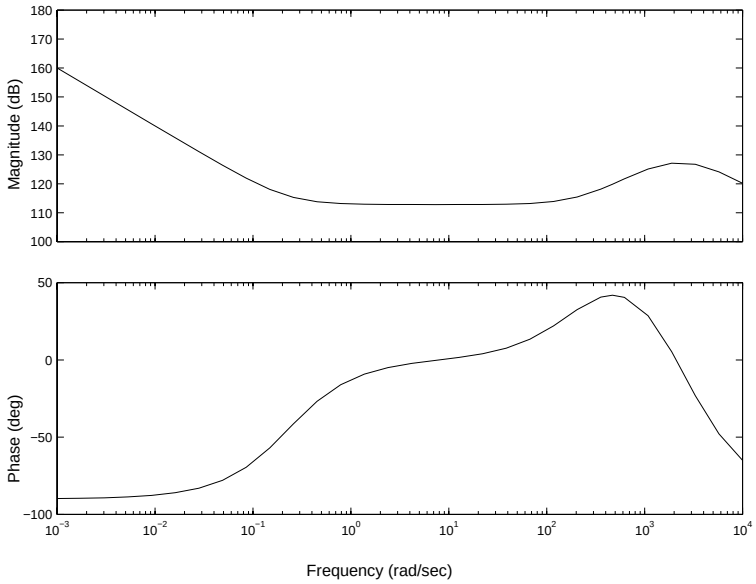

Figure 4.2: Bode plot of the PID-compensator with roll-off filter

The values of $\tau_h = 0.9 \cdot \beta\tau_d$ and $\tau_i = \frac{S_l K}{\omega_l}$ were chosen. The former should be as large as possible to limit the controller bandwidth, the latter as large as possible to get sufficient gain at low frequencies. This evaluated to the following numerical values:

Parameter	**Value**	**Unit**
τ_d	$4.5 \cdot 10^{-3}$	s
K	$0.44 \cdot 10^{6}$	
τ_1	4.4	s
τ_h	$0.41 \cdot 10^{-3}$	s

A Bode plot of the compensator is given in figure 4.2. A Bode plot of the achieved sensitivity function is given in 4.3.

The implementation of the PID controller was already available at the lab.

4.3 Experiment design

During preliminary experiments, it turned out that at high velocities and accelerations, the phenomena named before were hardly visible: at higher speeds, the kinetic energy of the moving mass is relatively large (it increases quadratically with the speed), compared to the work committed by cogging and commutation forces. From these results it was decided to use only low

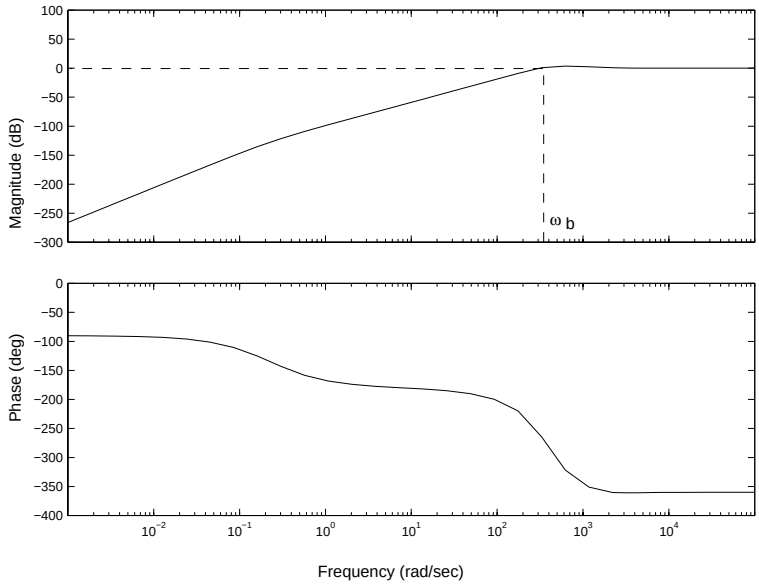

Figure 4.3: Sensitivity function of controlled mass

velocities and accelerations, of 0.11 m/s and 0.05 ms^{-2} respectively. Two paths were used to train the PLFFC. A third path was used to evaluate the quality of the learning of the other paths. The third path was considered complex enough to yield an appropriate evaluation. In fact the training paths are rather simple in nature. This results from the fact that a limit is put on the length of the paths: due to limited computer resources, only about one minute can be stored. With longer paths probably more information can be extracted from the LM. The first path, $P12$, is depicted in figure 4.4. It consists of a number of cycloid parts. As we look at the (position, acceleration)-plane, we see that it is well covered. Within the restrictions of the control setup, this a rather good achievement. We expect this path to train well.

The second path, $P13$, is displayed in figure 4.5. We see that the (position, velocity)-plane is covered worse, so with this path we expect the system to suffer from correlation more that with the previous path.

The last path, $P14$, is used for evaluation only. It consists of some fast and slow cycloid parts, and fast and slow parts at constant velocity (acceleration with a third order movement, i.e. finite and partwise constant jerk). The basic philosophy of the experiment is to find out what error reduction can be achieved on this path, by training with the other paths. Evaluating the error reduction on the training paths themselves may also be useful, but this does not allow us to compare the paths. Furthermore, this doesn't say much at all, since a training set should never be used as a

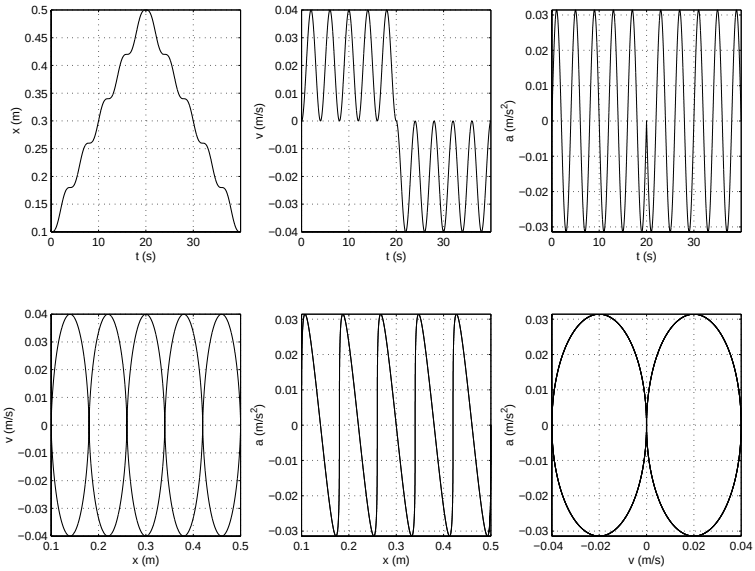

Figure 4.4: Path $P12$

validation set for an ANN.

During preliminary experiments, it was found that again in this case the optimal learning order holds. This means first training the acceleration network, followed by the velocity network and the position network.

During preliminary experiments it was also found that some data manipulation was necessary. Since stiction causes the I-action of the PID to increase, unreliably large steering values were found at small (but nonzero) velocities. (Of course, at *zero* velocity large signals are needed to conquer the stiction force, but this cannot be learnt in LFFC.) This convinced us to discard samples at low velocities. In casu this meant that only with respect to the position network samples are discarded if the velocity reaches values below 0.01 m/s. We should note that this means discarding the most intricate part of the function, but apparently we do not have a choice. An alternative solution would be to turn off the I-action, but this would increase the static error and the error at low velocities, which is not preferable.

The other networks used the entire batch of samples to update their weights.

The experiments were carried out with learning factors ranging from 1 down to 0.05.

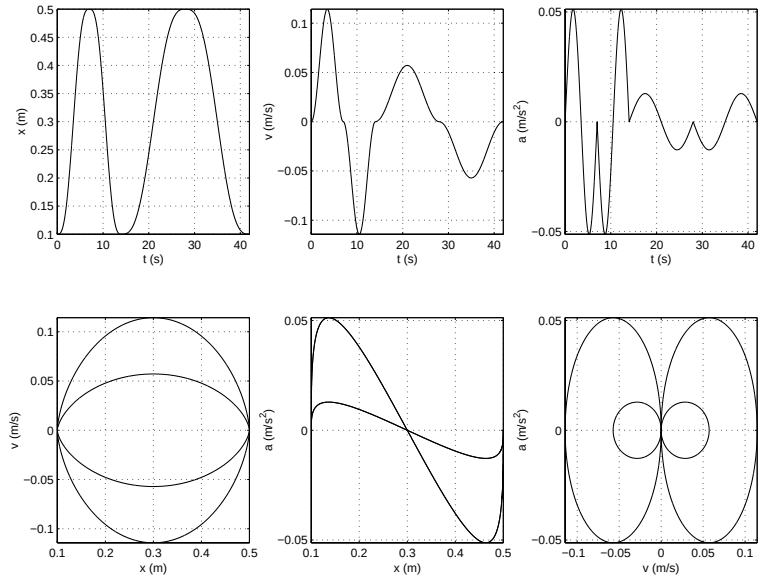

Figure 4.5: Path $P13$

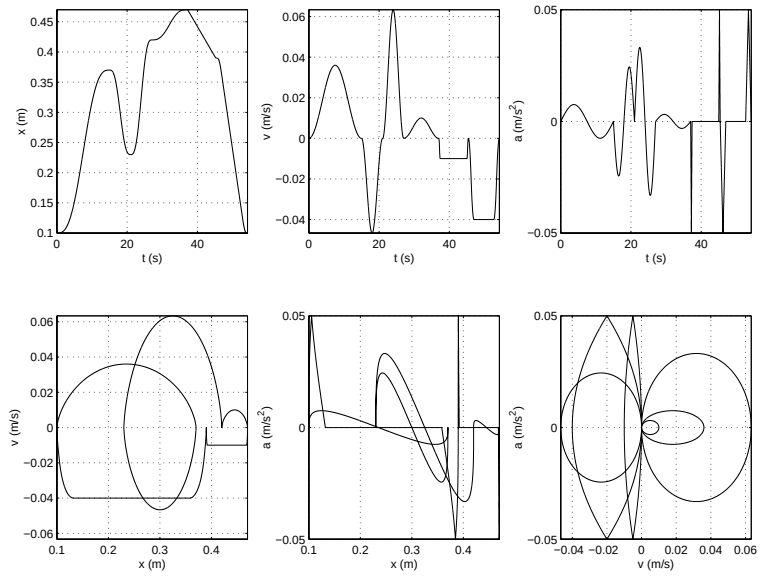

Figure 4.6: Path $P14$

	L. factor	It.	Est. mass	Mean err.	Max. err.
P12	0.05	0	9.1 kg	0%	0%
		1	28 kg	+3%	-3%
	0.5	0	91 kg	0%	0%
		1	303 kg	-25%	-30%
	1	0	182 kg	0%	0%
		1	392 kg	-34%	-39%
		2	*	-30%	-2%
P13	0.05	0	40.3 kg	0%	0%
		1	74 kg	-3%	-5%
	0.5	0	403 kg	0%	0%
		1	614 kg	-29%	-36%
	1	0	806 kg	0%	0%
		1	1023 kg	-37%	-39%
		2	*	-33%	-45%

*) An asterisk denotes a quantity that was not evaluated.

Table 4.1: Experimental results

4.4 Results

From various points of view, the results of the experiments are rather disappointing. As can be seen in table 4.1, in all experiments divergence occurred. This was primarily seen from the facts that the mass of the translator was heavily overestimated (10 to 100 times its real value), and from the fact that the increment of the mass estimation did not decrease. Furthermore, in most cases the error was reduced only in the first episode, increasing it again in following episodes. It was also seen that errors on the training paths themselves sometimes did increase even at the first iteration. These errors are not presented here. Only the error reductions on path $P14$, relative to the *original* error, are presented.

Divergence (or more precisely: incredibly large estimates for the mass) occurred both for high learning factors and low learning factors. The latter suggests that other reasons for divergence than a learning factor chosen too large should be looked for. We address the problem found in simulations, that the differences between true system states and the corresponding references are too large. First we look at the acceleration, since this was the largest trouble in the simulations. To this end we have to estimate the acceleration from the position measurement. This was done by first calculating the second gradient of the position measurement, and then filtering this gradient with an anti-causal 10^{th} order low-pass Butterworth filter, its cut-off frequency chosen 25 Hz. Choosing this frequency too low causes the

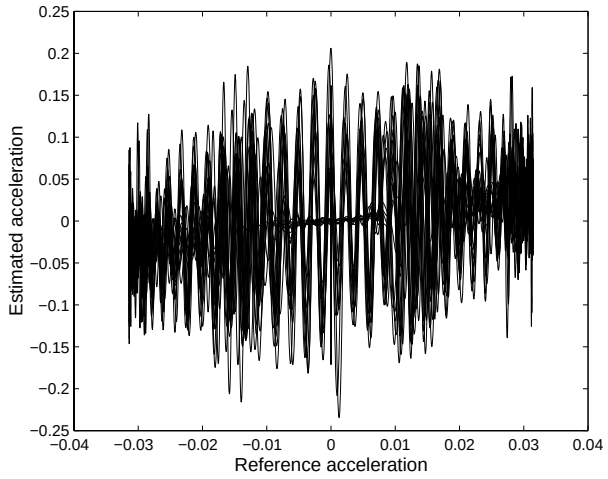

Figure 4.7: Estimated acceleration against reference acceleration, path $P12$

estimate to become unreliable, whereas at larger values only noise is found in the result. This acceleration estimate is plotted against the reference acceleration in figure 4.7 for path $P12$.

We see a trend with a slope of 1, but this trend is polluted with deviations. These deviations are large compared with the values of the acceleration. Given the evaluation of the simulations, we should assume that this is one of the main reasons for bad learning, and possibly for divergence.

Applying the same analysis to both the position and the velocity, only insignificant differences between the references and states were found. We should assume that these differences are not responsible for the divergence.

Furthermore, a projection of the control signal on the (reference) acceleration axis, depicted in figure 4.8, showed a lot of noise, and averaging this projection into a network yielded an unreliable estimate of 180 kg. This average is represented by the central bold dashed line. Its graphical representation seems to deliver a reliable average, but the numerical value invalidates it. This projection should be classified as badly suited for training. We should expect the acceleration network to leave a dirty residue because of its bad learning. Surprisingly no effects of this are found in the position and velocity networks, as addressed below.

The projection of the feedback control signal on the velocity axis is shown in figure 4.9. The bold dashed line represents the content of the network after learning. We see no reason to qualify this network as badly trained.

We finally address the projection of the feedback control signal on the position axis. This projection is depicted in figure 4.10 for path $P12$. The central bold line is the average as learnt by the network, the upper and

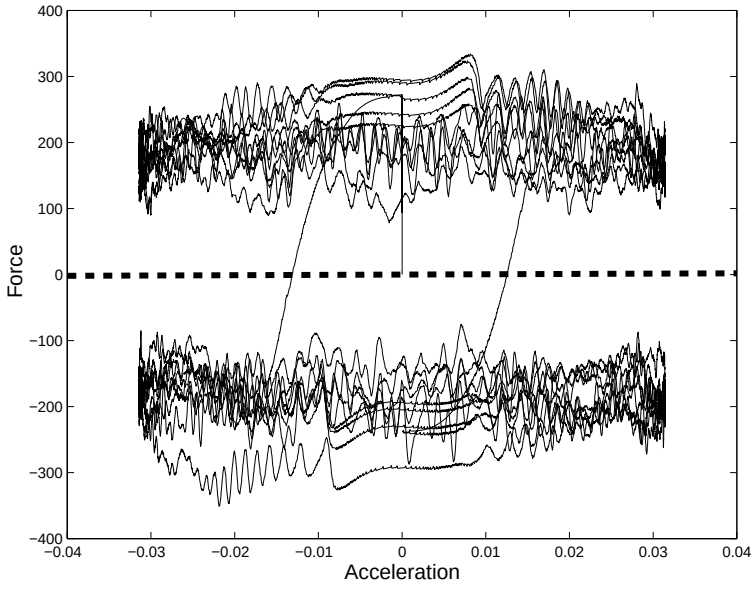

Figure 4.8: Projection of feedback control signal on acceleration axis, path $P12$

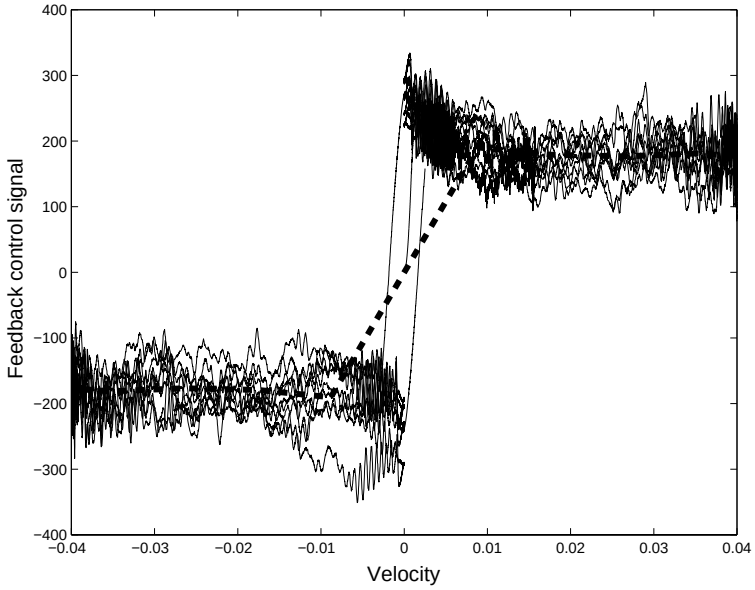

Figure 4.9: Projection of feedback control signal on velocity axis, path $P12$

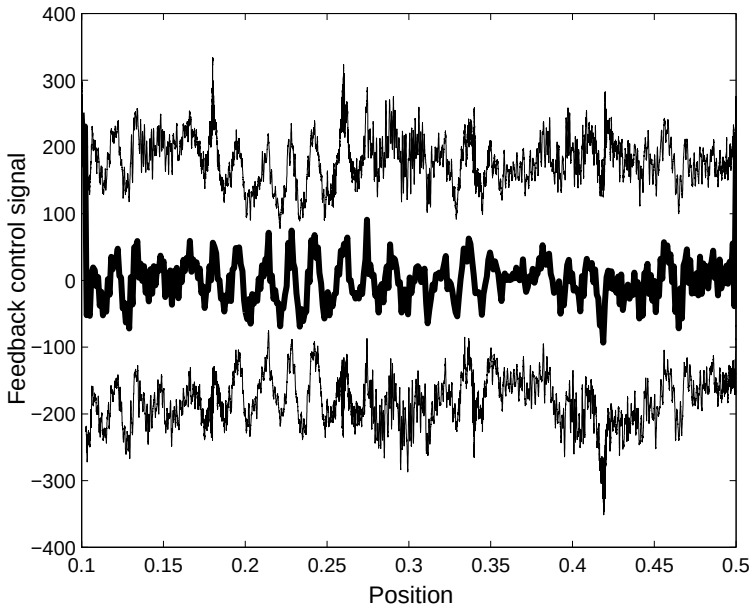

Figure 4.10: Projection of feedback control signal on position axis, path $P12$

lower thin lines are the true projections.

We see that the learnt control signal comprises sharp edges. These can probably be eliminated by applying an appropriately designed filter, but this is not investigated here. Beside these sharp edges we have no reason to qualify the contents of this network as bad.

The only reason for the occurring divergence thought of so far, was the difference between true acceleration and reference acceleration being too large. Therefore a small additional experiment was performed, where only the position and velocity were trained. This test was performed at an arbitrarily chosen learning factor 0.5. Again paths $P12$ and $P13$ were used for training, and $P14$ for evaluation. The results are given in the tables 4.2 and 4.3. Error reductions are evaluated on path $P14$, with respect to the original error. Furthermore the RMS values of the updates of the networks are given.

It.	**RMS Err**	**Max Err**	**Pos. upd.**	**Vel. upd.**
0	0%	0%	11.4	48.2
1	-27%	-36%	5.8	22.4
2	-35%	-42%	9.0	11.9
3	-32%	-40%	13.6	5.8

Table 4.2: Results of additional experiments with $P12$

It.	**RMS Err**	**Max Err**	**Pos. upd.**	**Vel. upd.**
0	0%	0%	14.4	84.4
1	-29%	-37%	16.2	41.1
2	-37%	-46%	3.6	20.7
3	-37%	-43%	8.3	10.8

Table 4.3: Results of additional experiments with $P13$

What we see is that in both cases the RMS value of the update of the velocity network decreases rapidly. This indicates convergence, although for a conclusive statement more iterations are needed. The convergence goes at the same speed (it is divided by two each step). The difference by a factor 2 is easily understood from the contents of the networks, drawn in figures 4.11 and 4.12 for path $P12$ and $P13$ respectively. The mean is taken over the full velocity domain, whereas path $P12$ covers only half as much as path 13 does. In the figures the convergence is well visible.

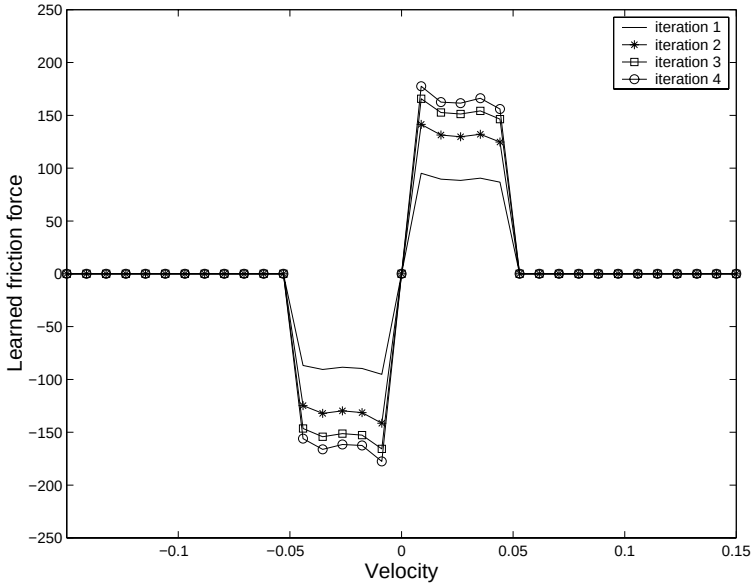

Figure 4.11: Contents of velocity network after several iterations, path $P12$

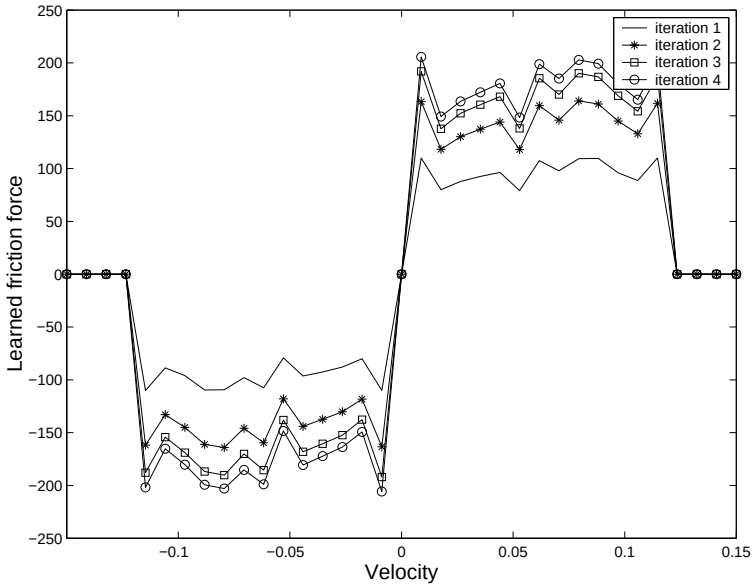

Figure 4.12: Contents of velocity network after several iterations, path $P13$

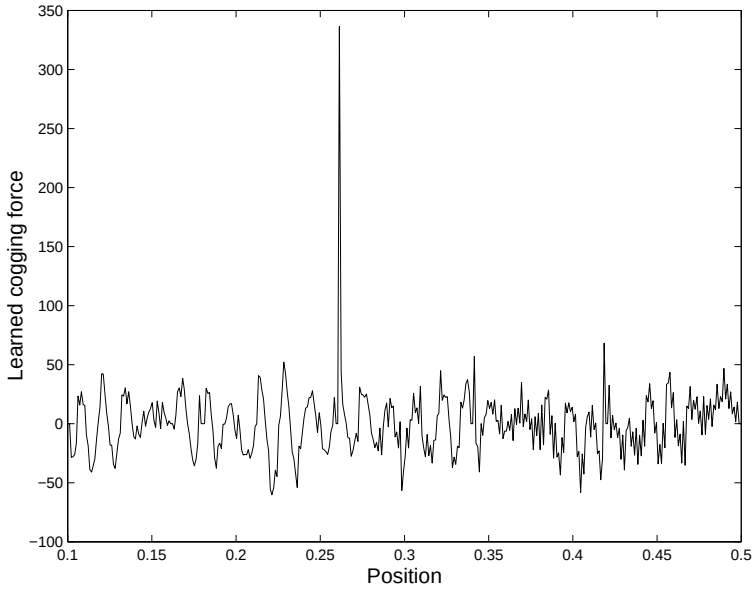

Figure 4.13: Contents of position network after several iterations, path $P12$

From these results we may conclude that it was indeed the divergence of the acceleration network causing the velocity network to learn badly. Unfortunately we do not see a confirmation of our expectation, i.e. that path $P12$ should enable learning better than $P13$. Actually, we from the velocity network we should qualify path $P13$ as better than $P12$, since a larger part of the network is filled with credible values.

For both paths we do not see any interference from other effects (i.c. cogging, inertia), but this is generally hard to see with this relative small number of splines. So we should be careful when concluding that interference from acceleration and position dependent forces is excluded by the symmetry.

Unfortunately we see a slight increase of the maximal error in the last iteration. Since the mean error still decreases, this is hard to judge on. Nevertheless, the feature should not be neglected.

The contents of the position network, depicted in figures 4.13 and 4.14 for path $P12$ and $P13$ respectively. For clarity only the result of the last iteration was depicted. The iterations in between did not comprise the same progress as with the velocity network, their development is much more arbitrary with both paths. With both paths we see a very irregular pattern, and especially some large peeks at positions where the motor stands still.

The results of the position network are much more credible, than with the original experiments. This founds our hunch that learning the inertia

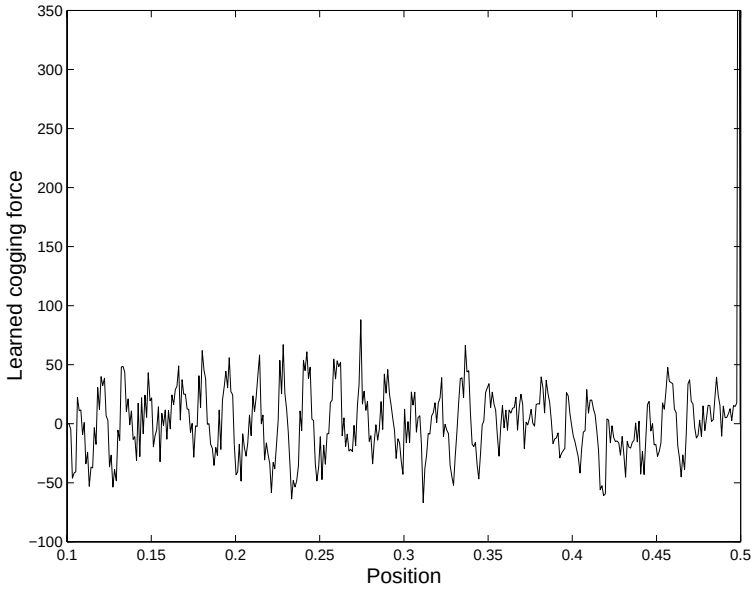

Figure 4.14: Contents of position network after several iterations, path $P13$

badly also influences the quality of the learning of the cogging. But from the contents of the networks we should conclude, that other influences must exist as well. One could think of learning speed, spline density, and the fact that the friction causes interference to the network at small velocities. Furthermore the sharp edges in the network contents may cause undesirable signals, that may have a negative influence on convergence. These options cannot be investigated within the current project time.

Unfortunately we still cannot judge on the principles very well. We may cautiously conclude that the cogging does not influence the learning of the friction, but since the friction still causes interference to the cogging, we should question the applicability of the symmetry principle. Since inertia cannot be learnt at all within the current domain, we cannot judge either on the question whether it interferes to other effects.

4.5 Discussion

In this chapter the experiments have been discussed. It was seen that convergence is hard to achieve. One source of trouble was eliminated by turning off the acceleration network, but still success was not achieved. Further research is needed.

Furthermore we should not neglect the sharp edges found in both the position and velocity networks. They will generally cause high-frequent control signals, which is undesirable. Therefore sophisticated filtering is

needed to 'smoothen' the control signal. It is not unlikely that convergence will then be achieved easier.

In the current set of experiments, both the acceleration and velocity are chosen small, in order to keep the cogging visible. On the other hand, applying these small values causes a bad tracking with respect to the acceleration, which heavily disturbs learning. From this we should question whether a real PLFFC can be trained by a single path at all. It is not unlikely that this is only possible applying various paths, which focuses our attention to the approaches by Idema (1996) and Steenkuijl (1999).

Because of these poor results, it is impossible to say whether the principles from section 2.2.1 are applicable in practice. They hold for properly conditioned data sets, but apparently it is hard to generate such a data set from a practical plant.

Chapter 5

Conclusions and recommendations

In this chapter the general conclusions of the M.Sc. assignment are drawn. Then a new training procedure for parsimonious learning feed-forward controllers is proposed. Finally, some recommendations for future research are formulated.

5.1 Conclusions

5.1.1 Results

We should state here, that some presumptions were made undeservedly. First of all the statement that the difference between the reference and the system states is small compared with the spline width, was not met in case of the acceleration. This proved to be troublesome in both the simulations and the experiments.

Second, convergence analysis was considered of minor concern. It was assumed that a learning factor chosen small enough would guarantee convergence, which proved not to be justified.

Third, in the experiments it was also assumed that symmetry neatly eliminates the effects of certain phenomena. It was seen that in some cases too large perturbations occurred to eliminate the effects: in the experiments, the large steering signals due to stagnations are not filtered out in the position network.

Beside these negative outcomes, we can still see some positive points, as addressed below.

5.1.2 Principles

Uncorrelatedness of data prevents interference from a target function to the learning of another target function. So the principles presented in section

2.2.1 generally hold:

1. If a phenomenon adds up to zero for every small subdomain of a function, it does not disturb the learning of such a function.
2. If the number of samples is large enough, it is also sufficient if only the expected value is zero.
3. If a function to be learnt is odd-symmetrical, this symmetry can be used to eliminate other influences.
4. The order in which functions are learnt, influences the quality of the learning. As a rule of the thumb, the network with the smallest number of splines should be trained first.

We should emphasize here, that these principles are mainly proven in the simulations, and hardly in experiments. Both the simulations and experiments show, that the applicability of the principles largely depends on the general condition of the LFFC. E.g. it was shown that phase shift on the acceleration can heavily disturb the learning, thus making the principles useless at all. Furthermore, it was shown that paths that meet the premisses of the principles poorly, do not necessarily perform badly, although there seems to be a correlation between these two.

5.1.3 Function decomposition and criteria

It follows from the theory, that the contents of the network after training do no necessarily equal the physical function decomposition. This is currently not taken into account when assigning the criteria. Furthermore it is not clear a priori whether the 'quality' of learning should be measured by the error reduction, or by the exact extraction of physical functions. In experiments the learnt relations are still heavily polluted, so we cannot judge on the question whether their contents approximate the true physical functions, or just a function that gives a control signal, appropriate to a certain extent. In the simulations the contents did approximate the true functions, but the conditions of the simulations were much more 'smooth' than in the experiments.

5.1.4 Richness

From the simulations it follows that richness of the paths, i.e. the extent to which it covers the input domains of the networks, is essential. It directly follows from the simulations that a rich path generally outperforms a path that covers input domains poorly.

Furthermore, while considering this richness, we should state that not only the density at which a domain is covered is important, but also the relation between the expected difference between the reference and the system

state, and the maximal values of these references. Stated more concretely: e.g. the domain of the acceleration should be chosen such, that the difference between the reference acceleration and the true acceleration is small compared with this domain. This was found in simulations and experiments. Unfortunately, in the latter case a solution was not found.

5.1.5 Symmetry

The theory presented in this thesis is a useful tool, but one of its weaknesses is found in the point that for a number of target functions symmetry is assumed. It also holds for asymmetrical functions, but in this case it is much harder to guarantee that a number of samples adds up to zero.

5.1.6 Divergence

It was noticed that in some cases a path can even cause divergence, whereas other paths cause convergence with the same learning factor. The former should be referred to as poorly conditioned. Firstly this can be caused by low richness. Secondly it can be the result from paths meeting the principles of this research poorly. And thirdly, this can result from the ratio of the error to the domain. (See 5.1.4.)

5.1.7 Applicability

From the experiments, we can question the applicability of the theory to real situations. Some extensions are necessary for application to real plants. Especially the relative error of the system states is important. This requires knowledge a priori, which actually was one of the things we tried to avoid in this project.

5.2 Training procedure

The results of this research do not replace the ideas postulated by Idema (1996) and Steenkuijl (1999), but rather should be seen as an addition to them. Here we present a training strategy combining the benefits of the existing theories.

It is assumed that the PLFFC itself is well-conditioned. This means that the composition of the parsimonious network is able to fit the decomposition of a function by the ANOVA-representation (which is, again, not unique in case of multivariate sub-functions), and that spline distributions are chosen appropriately. In casu this means that the distributions are dense enough to fit expected spatial frequencies of the target functions, and that they are sparse enough to filter out high frequencies. Furthermore it should be taken

into account that the required training time increases with the number of splines.

The feedback control should also be conditioned well, i.e. the difference between the reference variables and system states should generally be small.

It should be noted that in case of a physical experimental setup, this strategy has not proven to be successful. Nevertheless, in simulations it was successful. Furthermore it suffers from the fact, that convergence is not guaranteed a priori.

Then we propose the following steps to be taken:

1. Apply prior knowledge if available, using the approaches by Idema and Steenkuijl. If any phenomenon can be learnt by making it dominant over other phenomena, this should be pursued. For remaining phenomena, the following steps should be taken. The networks incorporating phenomena already compensated for should not be adapted any more, or at the most with a relatively small learning factor.

2. Determine the spatial conditions put on paths, by analysing the relation between different reference variables. If they are independent, choosing them uncorrelated is easy. If they are dependent, for example by a differential relation, special care should be taken when defining a reference path. In case of a differential relation, this determines the order of the reference path.

3. Determine the domain of every sub-function to be covered.

4. Try to find a path that covers all phase-planes of the reference variables in a symmetrical way (in case of symmetrical target functions), or in such a way that the average of a target function is zero on any arbitrary cross-section of the other reference variables (see principle 2.1) (in case of asymmetrical functions). The tough part, in case of asymmetrical functions, is to find out when the cross-section is zero.

5. Size the path such that it covers the domains correctly. One should take into account, that the number of samples available for every spline should be large enough. This demand depends on the expected variance of the samples within the spline. Furthermore it should be taken into account, that the speed at which transition through the domain takes place is limited by the fact that at high speed the phase shifting by the control loop can become too large. And finally, the scaling of the path should be such, that the difference between the reference variables and the system states is relatively small.

6. Apply the path to the plant.

7. Use projections of the feedback control signal on the several axes to update the function approximators with a unity learning factor. The network with the highest generalisation ability (which is generally the network with the smallest number of splines) should be updated first, and then using the residue of the control signal to update the subsequent networks. Furthermore the difference between reference variables and system states should be evaluated. If they are too large, either the path should be resized, or one should decide not to use the according networks.

8. Repeat this iteratively from 6, until a satisfactory error reduction is achieved, or otherwise go to 9.

9. If the network contents do not converge, try a lower learning factor. In this case the expected number of runs needed to train the PLFFC increases accordingly, so the relevance of this approach decreases.

10. If a lower learning factor does not achieve good results, a richer path, i.e. a path that covers the input domains better, can be tried, starting from 6.

5.3 Recommendations

5.3.1 Convergence

In this thesis convergence mostly was an issue of trial-and-error. If a powerful criterion on convergence for path-indexed LFFC systems were available, experiments and simulations would be easier to condition. In the present project, convergence sometimes was rather distracting our attention from the core. However, in the current literature no such criterion was found, and indications exist that this is hard to find. Nevertheless a challenging question remains. This thesis demonstrates that convergence may not only depend on the learning factor, but also on the path used for training. Also the idea of applying different learning factors to different networks might be worth investigating.

5.3.2 Criteria

The criteria used in this thesis do not suffice entirely for our purposes. The demands on a criterion depend on the purpose of the training, which was not taken into account here: for industrial purposes probably only the error reduction is relevant, whereas for scientific modelling and identification, a precise approximation of the physical functions is more important. This trade-off was not addressed in this thesis. Furthermore it might be worthwhile to take into account which part of the domain should be judged on:

generally a path cannot cover the *entire* input domain. In that case, the parts, that are visited poorly, cause the cost function to yield high values.

5.3.3 Regularisation and filtering

Generally in learning with a PLFFC, filtering can take place in two ways: transient and spatial. Transient filtering could consist of applying an anti-causal (zero-phase) filter to the learning signal. (Causality is not a constraint, when updating is performed off-line.)

Beside this transient filtering, also spatial filtering can be considered. Applying an anti-causal filter to the network contents (or its updates) may increase its performance: if one succeeds to achieve a higher 'degree of smoothness', high frequencies in the time domain caused by the BSN will be reduced, which may improve the overall performance.

Either type of filtering has its own benefits. At this point no judgement can be passed on them, so further investigation is needed.

Appendix A

Mathematical theory

A.1 *Diag*-operation

The *Diag*-operation is widely used, but hardly provided with notation. It is defined as follows:

Definition A.1 (Diagonal matrix of a vector) Let $\mathbf{v}$ be a vector with $\mathbf{v} = (v_1, v_2, \cdots, v_N)$. Then the *diagonal matrix* corresponding to $\mathbf{v}$ is given by:

$$V = Diag(\mathbf{v}) = \begin{bmatrix} v_1 & 0 & \cdots & \cdots & 0 \\ 0 & v_2 & 0 & \cdots & \vdots \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & v_N \end{bmatrix} \tag{A.1}$$

A.2 Partwise matrix inversion

By definition a singular matrix in non-invertible. A matrix containing empty (=zero valued) rows and/or columns does not have full-rank, and is therefore singular. However, if a matrix consists of regular submatrices, each of them located on the main diagonal, we can apply the following partwise inversion.

Definition A.2 (Partwise matrix inversion) Let R be a matrix of the form:

$$R = \begin{bmatrix} 0 & 0 & \cdots & \cdots & 0 \\ 0 & [R_1] & 0 & \cdots & \vdots \\ \vdots & 0 & 0 & 0 & \vdots \\ \vdots & \vdots & 0 & [R_2] & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix} \tag{A.2}$$

Then the *partwise inverse* $\widetilde{R^{-1}}$ of R is given by:

$$\widetilde{R^{-1}} = \begin{bmatrix} 0 & 0 & \cdots & \cdots & 0 \\ 0 & [R_1]^{-1} & 0 & \cdots & \vdots \\ \vdots & 0 & 0 & 0 & \vdots \\ \vdots & \vdots & 0 & [R_2]^{-1} & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix} \tag{A.3}$$

It is clear that for regular matrices the only submatrix complying to this definition is the matrix itself, thus yielding the partwise inverse to be equal to the regular inverse.

Of course the number of submatrices and their location on the main diagonal are arbitrary. In linear systems theory this matrix R can be interpreted as a set of decoupled systems. The partwise inversion can then be interpreted as the individual inversion of each of the systems.

Practically the same goes when R is the auto-correlation matrix of a set of samples for training a BSN. In case empty rows exist (necessarily accompanied by empty columns), the BSN can be interpreted as a set of sub-BSNs, each acting on a separate subdomain. The partwise inversion can then be interpreted as the individual inversion of the auto-correlation matrices of each of these sub-BSNs.

Appendix B

Matlab procedures

B.1 `createrefxx.m`

Files named `createrefxx.m` create paths according to the same names in this thesis. They consist of a part creating cycloid paths, and a part creating linear parts with third-order acceleration.

A version of `createrefxx.m` specially conditioned for the Tecnotion Linear motor setup is available in the directory `lmexp`.

B.2 `learn.m`

This file performs one learning episode for the 20-Sim model 'Linear Motor', as found in `20-Sim/LM50`. It assumes a matrix `result` to be present in the workspace. The first column should contain the control signal produced by the PD-compensator. The second column should contain the reference position, the third the reference velocity, and the fourth the reference acceleration, each of which are used as the input of the networks. The fifth should contain a time stamp of each sample.

Four networks are generated: a position network, a velocity network, an acceleration network, and a (position, velocity)-network. These are saved in files, which are used directly by the 20-Sim model. The networks are trained in the optimal order: acceleration, velocity, position, (position, velocity). Several options can be assigned in the code. The most important are the variables `additive` and `gammaz`. The first determines whether the network is either cleared before updating, or that the newly calculated weights are added to the old ones. The second constitutes the learning factor of the network `z`.

A version of `learn.m` specially conditioned for the Tecnotion Linear motor setup is available in the directory `lmexp`.

B.3 lookup1.m

The function `lookup1` performs a first order interpolation on a table defined by `from` and `to`.

It performs exactly like the Matlab function `interp1`, except for that it is much faster, because some safety checks are left out. For inputs outside the range of the table a linear extrapolation is used, where only the two outmost table entries are used. `index` always indicates the leftmost table entry used for either interpolation or extrapolation.

```
Usage: [RESULT,INDEX]=LOOKUP1(FROM,TO,IN)

Example:

FROM  =   [1 2 3 4 5 6 7]
TO    =   [7 5 3 1 3 5 7]

[RESULT,INDEX]=LOOKUP1(FROM,TO,3.5)

Yields:
RESULT    =   2
INDEX     =   3

Example:

[RESULT,INDEX]=LOOKUP1(FROM,TO,8)

Yields:
RESULT    =   9
INDEX     =   6
```

B.4 lookup2.m

The function `lookup2` performs an interpolation on a two-dimensional lookup-table. This function behaves just like the Matlab routine `interp2`, except for that it is much faster, because some safety checks are left out.

```
Usage: [RESULT,INDEX1, INDEX2] = LOOKUP2(FROM1,FROM2,TO, IN1, IN2)

TO is a matrix of which the contents are used as interpolation
points. FROM1 and FROM2 should contain distributions of the axes
of the matrix. IN1 and IN2 should contain the entries at which the
interpolation is desired.
```

```
For values outside the domains specified by FROM1 and FROM2 linear
extrapolation is performed, where only the two outmost are used.

INDEX1 and INDEX2 contain the lowest indexes of interpolation
points used.

Example:

FROM1     = [1 2 3]
FROM2     = [1 2 3]
TO        = [2 2 2;
             2 1 2;
             2 2 2]

[RESULT,INDEX1,INDEX2]=LOOKUP2(FROM1,FROM2,TO,1.5,1.5)

Yields:
RESULT    = 1.75
INDEX1    = 1
INDEX2    = 1
```

B.5 smartinv.m

The function `smartinv` performs the part-wise inversion as described in appendix A.

If a square matrix contains a zero-valued row, and the according column is zero-valued as well, these row and column can be left out before inversion is performed. In systems theory this equals the situation where independent systems are inverted separately. In B-spline learning it equals the situation where several sub-domains of the network are well-visited, with these domains separated by sub-domains that are not visited.

```
Usage: [AI]=SMARTINV(A)

Example:
A =   [ 1 1 0 0;
        0 2 0 0;
        0 0 0 0;
        0 0 0 4]
AI=SMARTINV(A)

Yields:
```

```
AI    =   [ 1 -0.5 0  0;
            0  0.5 0  0;
            0  0   0  0;
            0  0   0  0.25]
```

B.6 createtarget.m

This file creates the target matrix of a bivariate commutation network, as depicted in figure 2.6.

B.7 createvalmat.m

This file creates a validity matrix, on which learning can be based. It is designed rather arbitrarily only to exclude unreliable boundaries.

Appendix C

20-Sim extension `lookup2.dll`

Evaluating a two-dimensional second order BSN is the same as using a two-dimensional lookup table with linear interpolation. Control Lab Products does not provide such a table working properly. Therefore a system was implemented from the scratch, by creating a DLL-file with Microsoft Visual C++. Currently the DLL can only handle tables of 500×5 elements, where the first dimension corresponds to the first input. A 20-Sim submodel is provided to incorporate the DLL correctly. The DLL should be located either in the `20-Sim/bin`-directory or the current working directory.

The DLL opens the file `bsnxv.tab`, which should be placed in the current directory. The file should contain the table entries, in typewriter order (i.e. a table with 500 lines and 5 columns, which is read line by line).

The file uses exactly the same algorithm as the Matlab procedure `lookup2` which was described previously.

Appendix D

Files for the Linear Motor

The following files were made for the Linear Motor lab-setup. They were made in ANSI-C++, to be incorporated in existing control software. Three objects were made: a reference path generator with data-file input, a parsimonious feed-forward controller and a B-spline network that can only read files, without being able to learn.

D.1 `filepath.cc`

The code in `filepath.cc` reads a data-file with reference data, and uses this data to generate a path. The path is performed cyclic, i.e. at the end it is started from the beginning again.

The file to read should consist of four columns of floating point numbers, with the format: [time position velocity acceleration]

For example:

0.000e0	0.000e0	0.000e0	1.000e-1
1.000e-3	0.500e-7	1.000e-4	1.000e-1
2.000e-3	2.000e-7	2.000e-4	1.000e-1
3.000e-3	4.500e-7	3.000e-4	1.000e-1

describes the first four samples of a second order reference path.

Furthermore the file should apply to some other restrictions:

- The file should be homogeneous, i.e. the differences between the time indices should be equal.
- The path described by the file should be cyclic, i.e. it should stop exactly at the same point as where it started. This follows from the fact that discontinuities in the path should be avoided in order to avoid large control signals.

The object is *not* robust with respect to the following failures:

- Non-existent data files
- Non-homogeneous data files
- Oversized data-files; the size is limited by the size of the table, declared in filepath.h, which depends on compiler and computer capacities.
- Switching between different path generators. If the user starts a new path generator, for example `path2.cc`, the homing procedure is initialized wrongly. Therefore the user *should always restart the program before using a new path generator*, both in case of switching *to* and *from* the file-based generator.

D.2 `lmplffc.cc`

The code in `lmplffc.cc` combines a PID-controller and three BSNs to form a PLFFC. This object itself does not really do anything. It only instantiates the objects from `pid.cc` and `bsnfile.cc`. Furthermore it assigns appropriate names to the instances of `bsnfile.cc`, for which purpose the member variable `name` was made public.

D.3 `bsnfile.cc`

The code in `bsnfile.cc` delivers BSNs for the controller in `lmplffc.cc`. The BSNs are only capable of reading a file with weights, and then perform linear interpolation. To this end some parameters of the original file `lmbsn.cc` were replaced with default values. One important modification, was that the variable `name` was made public, in order to enable `lmplffc.cc` to identify the instances of the object.

The input file, generally with the extension `.wgt`, should contain two columns of floating point numbers, the first of which contains the input value, and the second the according weight. The file should be homogeneous, i.e. the spline width should be equal for all splines. For example:

```
-0.100e0    -1.000e0
0.000e0     0.000e0
0.100e0     1.000e0
```

constitutes a straight line through the origin, with a slope of +10, defined on the range [-0.1 0.1].

The object is *not* robust with respect to the following failures:

- Non-existent `.wgt`-files
- Oversized `.wgt`-files; the maximal size depends on the declaration in `bsnfile.h`, which on its turn is limited by compiler and computer capacities
- inputs outside the range on which the network is defined by the `.wgt`-file
- non-homogeneous spline distributions

Bibliography

Åström, K. and Wittenmark, B. (1997). *Computer-controlled systems - theory and design*, Information and systems science series, third edn, Prentice Hall.

Bhattacharryya, G. and Johnson, R. (1977). *Statistical concepts and methods*, Wiley series in probability and mathematical statistics, John Wiley and sons.

Bossley, K. (1997). *Neurofuzzy Modelling Approaches in System Identication*, PhD thesis, Faculty of engineering and applied science, University of Southampton.

Brown, M. and Harris, C. (1994). *Neurofuzzy adaptive modeling and control*, Prentice Hall International UK.

Coelingh, H. (2000). *Design support for motion control systems*, PhD thesis, Control laboratory, University of Twente.

De Vries, T., Velthuis, W. and Van Amerongen, J. (2000). Learning feed-forward control: a survey and historical note, *Proceedings 1st IFAC conference on Mechatronic systems*.

Idema, L. (1996). *Design of parsimonious learning feed-forward controllers*, Master's thesis, Control Laboratory, Universtity of Twente.

Löhnberg, P. (2000). *System identification and adaptive control*, number 024R99 in *Lecture notes*, University of Twente.

Spreeuwers, L. (1999). *Learning friction compensation*, Master's thesis, Control Laboratory, University of Twente.

Steenkuijl, J. (1999). *Design of a learning feed-forward controller for a robot manipulator*, Master's thesis, Control Laboratory, Universtity of Twente.

Stribeck, R. (1902). Die wesentlichen eigenschaften der gleit- und rollenlager, *Zeitschift des Vereins Deutscher Ingenieure* **46**(36,38): 1341–1348, 1432–1438.

Van Amerongen, J. and De Vries, T. (1999). *Meet- en regeltechniek, deel 2*, second edn, University of Twente, Control laboratory.

Velthuis, W. (2000). *Learning feed-forward control - theory, design and applications*, PhD thesis, Control Laboratory, Universtity of Twente.

Velthuis, W., De Vries, T., Schaak, P. and Gaal, E. (2000). Stability analysis of learning feed-forward control, *Automatica* **36**: 1889–1895.

Verwoerd, M. (2000). *Convergence analysis of learning feed-forward control*, Master's thesis, Control Laboratory, Universtity of Twente.

Acknowledgement

Photograph in figure 4.1 (page 51) by Job van Amerongen.

Typesetting made in LaTeX.

Index