Proceedings of the 41st IEEE
Conference on Decision and Control
Las Vegas, Nevada USA, December 2002

WeA06-1

Support-Vector-based Least Squares for learning non-linear dynamics[1]

Bas J. de Kruif and Theo J.A. de Vries[2]

Abstract

A function approximator is introduced in this paper that is based on Least Squares Support Vector Machines (LSSVM) and on Least Squares (LS). The potential indicators for the LS method are chosen as the kernel functions of all the training samples similar as with LSSVM. By selecting these as indicator functions the indicators for LS can be interpret in a support vector machine setting and the curse of dimensionality can be circumvented. The indicators are included by a forward selection scheme. This makes the computational load for the training phase small. As long as the function is not approximated good enough, and the function is not overfitting the data, a new indicator is included. To test the approximator the inverse non-linear dynamics of a linear motor are learnt. This is done by including the approximator as learning mechanism in a learning feed-forward controller.

1 Introduction

The performance of many industrial machines critically depends on their ability to let some end effector track a desired motion. Designing machines that maximise this ability is a truly mechatronic challenge, because mechanical system properties as well as electronics and control design may have a large influence on the tracking accuracy. As an example of this, one may think of the difficulties faced by designers of lithographic equipment for the exposure of wafers or of component mounters for production of printed circuit boards. In our research, we specifically consider the design of servo controllers for motion systems in a mechatronic setting. Such controllers will always involve a feedback component in order to deal with plant uncertainty and to obtain good disturbance suppression. To be able to track a motion with small errors, a feed-forward controller can be used in addition. The feed-forward controller generates the control signal from the reference (the desired motion) and is not error driven. The feed-forward controller can be chosen as the (pseudo-)inverse of the plant.

[1]This research has been made possible due to the support of the Euregio "German-Dutch Mechatronic Innovation Centre"

[2]Both authors are affiliated to the Drebbel Institute of Mechatronics, University of Twente, The Netherlands

Instead of mathematically computing the required feed-forward compensation, it can also be determined from the feedback control signal by using a function approximator. This may have distinct advantages, specifically for motion systems, as also unknown and non-linear system properties such as friction can be compensated in this way. The function approximator should learn the feed-forward control signal as a function of the relevant commanded plant states. This learning scheme is known as feedback error learning [10] and the control configuration has been called Learning Feed-Forward Control (LFFC) [12, 16]. Several examples of successful application of LFFC have been realised, e.g. Path tracking for an autonomous mobile robot [12]; Accurate positioning with a linear motor motion system [6].

It is of crucial importance that the function approximator that is contained in LFFC is computationally inexpensive because it has to output a control signal every sample instance. An approximator that is based on a set of basic functions is often used because of this. Because the space is divided into regions, the number of weights of this kind of approximators grows exponentially with the dimension of the input space. When the function to be approximated depends on several inputs, the learning process will fail for such approximators, something which is known as the curse of dimensionality. Large memories, difficult training and bad generalisation are the problems we are than dealing with [2, 6]. This is a major restriction for LFFC because motion systems commonly have a minimum input dimension of three and for a multi degree-of-freedom system this dimension increases substantially.

Recent insights in the field of statistical learning have shown alternative ways to approximate functions that are not liable to the curse of dimensionality [15]. Specifically, so-called Support Vector Machines (SVM) have shown to be good approximators for regression problems in high input dimensions. In previous work, we have shown that these techniques can also be used in LFFC [4], as a SVM is not computationally demanding when calculating an output for a given input. However, the computational load of the training process of an SVM is large, which implies that self-tuning or adaptive functionality cannot be implemented in SVM-based LFFC. This is a serious drawback. So-called Least Squares Support Vector Machines (LSSVM) [14] are an improvement in this sense. Unfortunately, these approximators are not sparse, which implies that a computationally demanding pruning process is necessary [5, 14]. Therefore, our research is aimed at finding a function approximator that does not suffer from the curse of dimensionality and is computationally cheap at the same time, both in terms of training and application. In this paper, we propose a function approximator that better meets these demands. We demonstrate its utility by considering a case, being an LFF-controlled linear motor motion system.

Because the function approximator is based on (LS)SVM and LS, these will be treated first in section 2. With this background the method is introduced in section 3. Subsequently, the method is used as learning mechanism in the simulation-based case study, which is presented in section 4. The paper will end with conclusions in section 5.

2 Function Approximation - Basics

Consider a given set of training samples $\{x_k, y_k\}_{k=1\ldots N}$, in which x_k is the input vector and y_k is the corresponding target value for sample k. The goal of function approximation is to learn the underlying relation between the input and the target value. After learning, a function approximator implements a mapping $x \rightarrow \hat{y}(x)$ that can be evaluated for any x. The performance of a function approximator is measured by means of the approximation error e_k, which is defined as $e_k = y_k - \hat{y}(x_k)$. This section summarises the theory of two learning mechanisms, namely LS and LSSVM.

2.1 Least Squares

The Least Squares method learns a linear relation between a set of indicators $f_k(x), k = 1 \ldots n$ and the output $\hat{y}$:

$$\hat{y}_{\text{ls}}(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \ldots + \alpha_n f_n(x) \qquad (1)$$

In this equation, the α's are the n parameters that get a value during the training by minimising the summed squared approximation error over all examples. The indicators can be non-linear functions of the input vectors. Define the matrices X and Y containing respectively the indicators for all the k samples and the target values:

$$X = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \\ f_1(x_N) & f_2(x_N) & & f_n(x_N) \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \qquad (2)$$

in which the subscript of x and y denotes the sample number. Then the minimisation problem is given by the following matrix expression:

$$\min_{\alpha} \|X\alpha - Y\|_2^2 \qquad (3)$$

It can be solved by setting the derivatives equal to zero, resulting in the normal equations [7, 13]:

$$(X^T X)\,\alpha = X^T Y \qquad (4)$$

2.2 LSSVM

With SVM, the relation underlying the data is represented as:

$$\hat{y}_{\text{svm}}(x) = w^T \phi(x) + b \qquad (5)$$

in which ϕ is a mapping of the vector x to some feature space, b is the bias and w is a weight vector of the same dimension as the feature space. The mapping $\phi(x)$ is commonly non-linear and makes it possible to approximate non-linear functions. In the case of LSSVM we search for that weight vector that will give the smallest summed square approximation error over all samples (in the case of Vapnik's SVM another criterion is used). A regularisation parameter γ is used to avoid overfitting. This results in the minimisation problem:

$$\min_{w,b} \mathcal{I}(w,e) = \frac{1}{2} w^T w + \frac{1}{2}\gamma \sum_{k=0}^{N} e_k^2, \qquad (6)$$

with equality constraint

$$y_k = w^T \phi(x_k) + b + e_k. \qquad (7)$$

This problem can be solved using optimisation theory [1]. Instead of minimising the primary objective (6) with constraints (7), a Lagrangian can be formed of which the saddle point gives the optimal values. The optimal values can be found by setting the derivatives of the Lagrangians equal to zero. By removing the w and the e through substitution, a set of linear equations is found by which the optimal values can be calculated:

$$\left[\begin{array}{c|c} 0 & \vec{1}^T \\ \hline \vec{1} & \Omega + \gamma^{-1} I \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline Y \end{array}\right]. \qquad (8)$$

In this equation $\vec{1}$ is a column vector of appropriate dimension filled with ones, α is a vector with the Lagrangian multipliers and Y is previously defined. The elements of matrix Ω equal $\Omega_{kl} = \langle \phi(x_k), \phi(x_l) \rangle = \Phi(x_k, x_l)$ and is a matrix which contains all the innerproducts between the training samples *in the feature space.* The function $\Phi(x_k, x_l)$ is called a kernel function. The innerproduct is defined as $\langle \phi(x_k), \phi(x_l) \rangle = \phi(x_k)^T \phi(x_l)$. The mapping $\phi(x)$ from input space to feature space does not have to be made explicitly in order to calculate the elements of Ω; it can be calculated in the input space. Possible mappings result in an approximation by polynomials, splines or radial base functions [15].

The output of the approximator can be calculated for new values of x with α and b. The output is given by

$$\begin{aligned} \hat{y}(x) &= \langle w, \phi(x) \rangle + b, \\ &= \left\langle \sum_{k=1}^{N} \alpha_k \phi(x_k), \phi(x) \right\rangle + b, \\ &= \sum_{k=1}^{N} \alpha_k \langle \phi(x_k), \phi(x) \rangle + b, \\ &= \sum_{k=1}^{N} \alpha_k \Phi(x_k, x) + b. \end{aligned} \tag{9}$$

The curse of dimensionality is circumvented because the input space is never divided into small regions: the *dimension* of the input space plays no role. Equation (9) only contains the innerproducts of an input with the training samples in some feature space and this results in a scalar, irrespective of the dimension of the input vector. Hence, the number of parameters depends not on the input dimension but on the number of training samples. But, to approximate the training data, not all the training samples are required. A large portion of the parameters α_k can be made equal to zero. If the α_k is zero, the kernel function doesn't have to be evaluated for that training sample to calculate the output. Those training samples for which α_k are non-zero are called *support vectors.*

In LSSVM, the Lagrangian multipliers get a value which is proportional to the error at that training sample. This means that (unlike in Vapnik's SVM) nearly all the multipliers are non-zero and are required for the calculation of the output (9). If the number of training samples is large, this becomes infeasible. To circumvent this, pruning is applied: the learning process is executed repeatedly, and samples that only have a small influence on the output are omitted in consecutive steps. In [14] it is argued that the sample with the smallest absolute multiplier (i.e., the smallest error in the *current* pass) should be omitted. However, this approach is suboptimal; in [5] a pruning scheme is introduced that minimises the approximation error that results *after* omission. This approach performs significantly better, although the computation load increases.

2.3 Evaluation

If we evaluate both learning methods and compare their properties, we may note the following:

- The calculation of the output of both function approximators is identical if: The Least Squares indicator functions are chosen equal to the LSSVM kernels of the training samples: $f_k(x) = \Phi(x_k, x)$; To account for the LSSVM bias b, the first row of X in (2) should contain only 1's.

  Thus the support vector methodology gives rise to a new set of indicator functions for least squares that can be interpret from a support vector setting.

- In LSSVM, all the training samples are mapped to a feature space and in this space a linear function is sought that minimises the sum of squared errors. The number of multipliers is equal to the number of training samples giving rise to a set of linear equations with dimension $N+1$ (one extra for the bias). The number of parameters, the Lagrangian multipliers, is reduced by omitting samples with little influence. This pruning leads to a loss of (valuable) information, as the omission of samples not only implies a reduction of parameters, but also a reduction of points at which the approximation is evaluated. In the least squares method one tries to minimise directly the sum of squared error by changing the Lagrangians. This makes it possible to use only a subset of the training samples to construct the indicators, while all training samples are used for the training of these indicators.

- LSSVM initially uses a parameter for each training sample and subsequently reduces this number. This is computational unattractive, because it means that the iterative solution of matrix equation (8) starts with a maximum dimension, which is subsequently reduced. The calculation time is large if the number of samples is large relative to the number of support vectors required for the approximation. In least squares, one is able to work the other way around: start with few indicators, and include more if the performance is too low. This is computationally more attractive. Neither method is guaranteed to find the optimal subset [11].

3 Support-vector-based Least Squares

Based on the above observations, we propose a new Least Squares learning method that is based on support vector machine:

- Indicator functions are chosen conform point 1 above. That is, kernels at a training sample as used in (LS)SVM are potential indicator functions.
- A limited number of indicator functions is selected out of the N candidates, with N equal to the number of training samples. For this, we need to specify an indicator selection procedure.
- Learning is done recursively; as long as the performance measured over all training points is not sufficient and overfitting has not yet occurred, the number of indicators is increased in consecutive steps. For this, we need stop criteria.

3.1 Indicator selection

Consider an illustrative example, refer figure 1. We have collected a set of 25 data points of the function $y = \sin(x-0.25)/(x-0.25)$ given by the black line. We wish to approximate the function with the new method by means of radial base functions, i.e., *potential* indicator functions are $f_k(x) = \langle \phi(x_k), \phi(x) \rangle = \Phi(x_k, x) =$

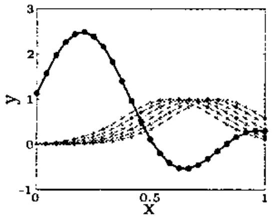

Figure 1: The elements of the training set and 6 of the potential indicators

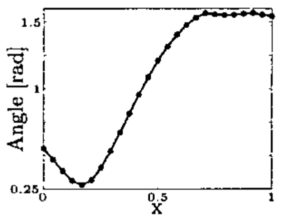

Figure 2: Angle between the error vector and the potential indicators

$\exp(-\|x - x_k\|_2^2), k = 1 \dots 25$. Six of these potential indicator functions are shown in gray in the figure. So, potential indicators differ in where their centres are located. Out of all these indicators a sufficiently large yet minimal set should be selected to approximate the training data. This is a known problem in regression [3, 7, 9, 11].

It follows from (2) that each indicator generates a column vector in X with a dimension equal to the number of samples N. The residual of the current approximator for the given training set is given as $e = Y - X\alpha$, which is also a vector of dimension N. Now we wish to add an indicator. Out of all potential indicators, we wish to select the one that most significantly decreases the summed squared approximation error. If an indicator could be found that is linearly related to the residual vector e, the inclusion of that indicator would make the residual zero. The presence of such an indicator is not likely; however, its closest match is that potential indicator vector that will generate a column vector in X that makes the smallest absolute angle with the residual vector e. This potential indicator should thus be selected. The same is found in [3, 8, 11] to minimise the sum squared error.

The proposed indicator selection procedure is shown in figure 2; it plots the angle between the function to be approximated and all potential indicator vectors. If no indicator has been selected yet, the training samples equal the elements of the error vector. It can be seen that the minimal angle occurs at $x \approx 0.17$ and hence this sample defines the first indicator to be selected, which makes sense intuitively.

3.2 Stop criteria

When it is known which potential indicator will give the largest error reduction, it still has to be decided whether or not to included this candidate indicator as an actual indicator. There are two reasons for not including it: The approximation is good enough, or the candidate indicator is fitting noise rather than a part of the function. The first reason simply requires a stop criterion that specifies what value is acceptable for the summed squared approximation error. The second reason for stopping can be tested with statistics.

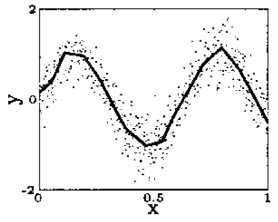

Figure 3: $\sigma^2 = 0.1$

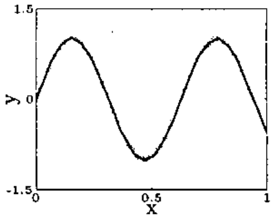

Figure 4: $\sigma^2 = 0.001$

To avoid fitting noise, a candidate indicator should not be added to the set of indicators if its weight α_i is known, with some given probability, to be zero. To test this, we can calculate this probability; if it is smaller than some user-defined bound, the candidate indicator should be included, because its weight is not likely to be zero. To test this, define the extra sum of squares S as the difference between the summed squared approximation errors over all samples before and after the inclusion of the candidate indicator. In [7] it is shown that S is distributed as χ_1^2 if and only if the indicator weight is zero:

$$p = \frac{S^2}{\sigma^2} \sim \chi_1^2 \Longleftrightarrow \alpha_i = 0 \tag{10}$$

In here σ^2 is the variance of the noise. If the indicator weight is zero, S should be χ_1^2 distributed if the additive noise on Y is Gaussian. For example, the summed squared approximation errors changes from 11 to 10 while the variance of the noise is 0.25. This gives a realisation of the variable p of 4. From the χ_1^2 distribution it is found that the probability of a realisation larger than 4 is 4.6%. Thus we know with probability 95.6% that the indicator is unequal to zero.

$$P(\alpha_i = 0 \,|p \geq 4) = 4.6\% \rightarrow P(\alpha_i \neq 0 \,|p \geq 4) = 95.6\% \tag{11}$$

In figure 3 and figure 4 a sine function based on data corrupted by noise of different variances is approximated. The function was approximated by a piecewise linear approximation and an indicator was still included if it was unequal to zero with 95% probability. The dots are the training samples and the line is the approximation. In figure 3 approximation is stopped earlier than in figure 4, because, due to the high noise level, it would earlier start fitting noise.

4 Simulations

The goal of the set of simulations is the testing of the learning mechanism described above in an LFFC setting. The simulations will be performed with a model of a synchronous linear motor. Such a system can be described as a moving mass with non-linear disturbances in the form of (position-dependent) cogging and (position- and velocity-dependent) friction. The differential equation is then given as:

$$m\ddot{x} + f(\dot{x}, x) + c(x) = F \qquad (12)$$

in which f is the friction, c is the cogging, F the applied force and m the mass. If the applied force would be set equal to

$$F = m\ddot{x}_{\mathrm{d}} + f(\dot{x}_{\mathrm{d}}, x_{\mathrm{d}}) + c(x_{\mathrm{d}}) \qquad (13)$$

the actual motion would equal the desired motion if the initial conditions were the same. The subscript d means desired in this equation. The mass was assumed to be known and compensated for by a perfect fixed feed-forward. However, the functions f and c are supposed to be unknown. These functions should be learnt to give the total feed-forward force. This force can only be approximated if $\dot{x}_{\mathrm{d}}$ and x_{d} are available as inputs to the function approximator. The model that is used in the simulations is more extensive to obtain more realistic simulation. The linear transfer function also included the first three identified vibration modes. Next to this, cogging and friction were fitted to measurements and a discrete controller was used.

The training samples for function approximation are obtained from a set of repeated movements in which the feed-forward signal is first learnt as function of the time. This in fact implies that an Iterative Learning Controller is implemented with a specific choice for the learning filter, namely equal to the PD feedback controller [16]. The obtained feed-forward signal, a function of the motion time, has compensated for the state dependent effects. This time-based feed-forward signal is used as training data for a function approximator to learn the feed-forward signal as a function of the desired states. The mapping from the reference states to the feed-forward signal has been learnt for several choices of the potential indicator functions $\Phi(x_k, x)$:

- piecewise linear functions of the desired position: $f(x)$;
- piecewise linear functions of the desired position *plus* piecewise linear functions of the desired velocity: $f(x) + g(v)$;
- piecewise linear functions of both the desired position and velocity: $f(x, v)$;

To test the correctness of the approximation, a movement was made that was not incorporated during training. This reference movement goes smoothly from 0 [m] to 0.4 [m] in 3 seconds and then it goes back again. The tracking errors for the various approximators before and after training are given in figure 5.

First a learning mechanism is used that can approximate cogging only with a piecewise linear function. In

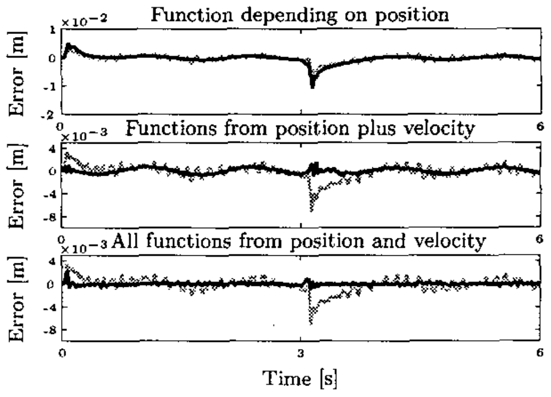

Figure 5: The tracking error for different potential indicator functions, before (gray) and after (black) training.

the top of figure 5 it can be seen that the fast fluctuation is decreased significantly. However, an error is present when the direction of movement is changed, as is a slowly fluctuating error. This slow fluctuation is due to the position dependent friction and cannot be learnt if only functions depending on the position are allowed. The error during a change in motion direction can also not be learnt with only the position as input. The set of functions from which the learning mechanism could choose in the second experiment was expanded to a piecewise linear function depending on the position *plus* a piecewise linear function depending on the velocity. As can be seen in the middle of figure 5, the error due to a changing motion direction has almost vanished, however, the slow fluctuation (due to position-dependent friction effects) is still present. In the third experiment the set of function was expanded to the set of all piecewise linear function with inputs velocity and position. The result can be seen in the bottom of figure 5. The slow fluctuation is gone and only a small error signal remains.

For comparison, a two-dimensional B-spline network has been used to approximate the same training data and evaluated with the same reference motion. B-spline networks were used in previous work on LFFC. The tracking error of the B-spline LFFC and the last simulation with LS is given in figure 6. It can be seen that the new learning mechanism outperforms the BSN network by far. This is due to the difference in sensitivity for the curse of dimensionality.

The computation time of the LS method is much lower than of the LSSVM method. When 1000 samples are used, the training time of the least squares takes a few second, while the LSSVM takes several minutes. Even as it took more time to calculate the support vectors and their corresponding Lagrangian multiplier, the results of the LSSVM were considerable worse. If the pruning algorithm is used that is proposed in [5] the

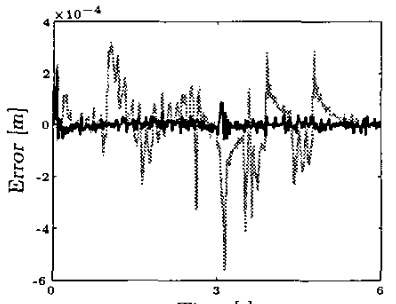

Figure 6: tracking error for the BSN (gray) and the LS (black) learning mechanism

computational time was several tens of minutes, but the found parameters gave the approximately the same performance as the least squares algorithm.

5 Conclusions

In this paper, a Least Squares learning method has been introduced that is based on support vector thinking:

- All kernels at the training samples as used in (LS)SVM are considered as potential indicator functions.
- A limited number of indicator functions is selected out of the candidates. An indicator selection procedure has been described for this.
- Learning is done recursively; as long as the performance measured over all training points is not sufficient and overfitting has not yet occurred, the number of indicators is increased in consecutive steps. Stop criteria have been given for this.

This learning mechanism is used to learn the inverse dynamics of a linear motor with Learning Feed-Forward Control. It was capable of approximating the position-dependent cogging and the position- and velocity-dependent friction. The tracking error of an unlearnt movement could be reduced considerably in simulations. The significant improvement over previous results is that this new function approximator is not sensitive for the curse of dimensionality and yet features a relatively small computational load, both during training and during application. This promises to make realisation of a self-tuning LFFC for multi-degree-of-freedom motion systems possible.

References

[1] M. Aoki. *Introduction to Optimization Techniques, Fundamentals and Applications of Nonlinear Programming.* Macmillan Series in Applied Computer Sciences. The Macmillan Company, New York, 1971.

[2] M. Brown and C. Harris. *Neurofuzzy adaptive modeling and control.* Prentice Hall International, UK, 1994.

[3] S. Chen, F.N. Cowan, and P.M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, March 1991.

[4] B.J. de Kruif and T.J.A. de Vries. On using a support vector machine in learning feed-forward control. In *Proc. Int. Conf. On Advanced Intelligent Mechatronics (AIM'01)*, pages 272–277, Como,Italy, July 2001. IEEE/ASME.

[5] B.J. de Kruif and T.J.A. de Vries. Pruning error minimization in least squares support vector machines. submitted to Transactions on Neural Networks.

[6] T.J.A. de Vries, W.J.R. Velthuis, and L.J. Idema. Application of parsimonious learning feedforward control to mechatronic systems. *IEE Proc. D: Control Theory & Applications*, 148(4):318–322, July 2001.

[7] N. R. Draper and H. Smith. *Applied Regression Analysis.* Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., New York, 3rd edition, 1998.

[8] M. H. Fun and M.T. Hagan. Recursive orthogonal least squares learning with automatic weight selection for gaussian neural networks. In *International Joint Conference on Neural Networks*, Washington, July, 1999.

[9] G.M. Funival and Jr. R.W. Wildon. Regressions by leaps and bounds. *Technometrics*, 16(4):499–511, November 1974.

[10] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987.

[11] A.J. Miller. *Subset Selection in Regression.* Chapman and Hall, London, 1990.

[12] J.G. Starrenburg, W.T.C. van Luenen, W.Oelen, and J. van Amerongen. Learning feedforward controller for a mobile robot vehicle. *Control Engineering Practice*, 14(9):1221–1230, 1996.

[13] G.W. Stewart. *Matrix Algorithms*, volume 1:Basic Decompositions. SIAM, Philidelphia, 1998.

[14] J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse approximation using least squares support vector machines. In *IEEE International Symposium on Circuits and Systems ISCAS'2000*, 2000.

[15] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 2nd edition, 2000.

[16] W.J.R. Velthuis. *Learning Feed-Forward Control, Theory, Design and Applications.* PhD thesis, University of Twente, Enschede, Febuary 2000.