

ISSN 0280-5316
ISRN LUTFD2/TFRT--5733--SE

Controller Design for a Direct Coupled Motor

Ola Svensson
Carl Windfeldt

Department of Automatic Control
Lund Institute of Technology
December 2004

Department of Automatic Control Lund Institute of Technology Box 118 SE-221 00 Lund Sweden		<i>Document name</i> MASTER THESIS	
		<i>Date of issue</i> December 2004	
		<i>Document Number</i> ISRN LUTFD2/TFRT--5733--SE	
<i>Author(s)</i> Ola Svensson and Carl Windfeldt		<i>Supervisor</i> Pontus Nordfeldt and Tore Hägglund at LTH in Lund	
		<i>Sponsoring organization</i>	
<i>Title and subtitle</i> Controller Design for a Direct Coupled Motor (Regulator design för en direktkopplad elektrisk motor)			
<i>Abstract</i> This thesis describes an effort to enhance the control capabilities of an electric motor. It is in the interest of TetraPak that the research on this motor is performed so that it, in the future, can be a part of their production systems. The thesis has been separated into three main parts, where in the first part we are trying to find a model that describes the process. The second part describes how the controllers were developed and the final part how they were implemented in real time. Our goal was to successfully identify the motor and control it within the, from TetraPak, specified demands. And if time allowed, do it all automatically.			
<i>Keywords</i>			
<i>Classification system and/or index terms (if any)</i>			
<i>Supplementary bibliographical information</i>			
<i>ISSN and key title</i> 0280-5316			<i>ISBN</i>
<i>Language</i> English	<i>Number of pages</i> 107	<i>Recipient's notes</i>	
<i>Security classification</i>			

Controller Design for a Direct Coupled Motor



Ola Svensson E-00
&
Carl Windfeldt M-99

Supervisor: Pontus Nordfeldt
Examiner: Tore Hägglund

Department of Automatic Control
Lund Institute of Technology

13th December 2004

Acknowledgements

We would especially like to thank our supervisor Pontus Nordfeldt at the department of Automatic Control, Lund Institute of Technology. He has supported us with guidance and inspiration, and has devoted a lot of time to us.

We would also like to thank Tore Hägglund at the department of Automatic Control, Lund Institute of Technology. As our examiner he has helped us structuring this master thesis and has been very helpful.

Thanks go to Anders Sundberg, Sven Hedlund and Istvan Ulvros at TetraPak R&D for supplying us with the motor. They also gave us the opportunity to visit TetraPak and see the production lines where the motor perhaps will be used. Anders, Sven and Istvan also visited us at the department half through the project and gave us constructive feedback.

We would like to thank B&R Automation who provided us with the PLC and the motor drive. In particular, we would like to thank Daniel Cederström who installed the new PLC.

Finally, we would also like to thank the proofreader Barbro Ericsson.

Contents

1	Introduction	6
1.1	Background	6
1.2	Specifications	6
1.3	Our goals	7
1.4	Report overview	7
2	Process description	9
2.1	Motor and load	9
2.2	Angular sensor	9
2.3	Motor drive and PLC	9
2.4	Time delay and sampling period	10
2.5	Disturbances acting on the process	11
2.6	Nonlinearities in the process	12
3	System identification	13
3.1	The identification procedure	13
3.2	Modeling of a motor	14
3.3	Identification methods	14
3.3.1	Relay identification	14
3.3.2	Least squares identification	15
3.3.3	Subspace identification	16
3.3.4	Recursive least squares identification	16
3.3.5	Validation	17
3.3.6	Model reduction	18
3.4	Model design	19
3.4.1	Relay identification	19
3.4.2	Least squares identification	21
3.4.3	Subspace identification	24
3.4.4	Recursive least squares identification	25
3.5	Model validations	28
3.5.1	Relay identification model	28
3.5.2	Least squares identification model	28
3.5.3	Subspace model	29
3.6	Problems	29
3.7	Summary	31
4	The control problem	33
4.1	Motivation	33
4.2	State space model	34

4.3	Controller methods	34
4.3.1	RST controller	34
4.3.2	State feedback control	35
4.3.3	Linear quadratic control	35
4.3.4	Linear quadratic gaussian control	36
4.3.5	Linear quadratic self-tuning regulator	37
4.4	Controller design	37
4.4.1	RST controller	37
4.4.2	State feedback controller	38
4.4.3	Linear quadratic controller	39
4.4.4	Linear quadratic gaussian controller	46
4.4.5	Linear quadratic self-tuning regulator	47
4.5	Problems	51
4.6	Summary	51
5	Reference following - The servo problem	53
5.1	Motivation	53
5.2	Reference following methods	53
5.2.1	Prefiltering	53
5.2.2	Iterative learning control	54
5.3	Reference following design	54
5.3.1	Prefiltering	54
5.3.2	Iterative learning control	55
5.4	Summary	58
6	Programming	60
6.1	General	60
6.2	The B&R Automation Studio environment	60
6.3	Identification methods	61
6.4	Controllers	61
7	Testing and evaluating	63
7.1	Changing loads	63
7.1.1	Procedure	63
7.1.2	Four weights attached	64
7.1.3	No weights attached	67
7.2	Increasing the current	70
7.3	Test results	72
8	Conclusion	74

9 Discussion	75
References	77
A Matlab code	78
A.1 Relay	78
A.2 ARX-model	78
A.3 Subspace identification	80
A.4 Pole placement for the RST controller	81
A.5 LQ control	81
A.6 LQG control	82
A.7 Prefilter calculations	83
A.8 Curve generating algorithm	84
A.9 ILC filter calculations	84
B Programming	86
B.1 Identification methods	86
B.1.1 Relay identification	86
B.1.2 Least squares identification	88
B.1.3 Recursive least squares identification	92
B.1.4 Resampling and computation of LQ parameters	94
B.2 Controllers	95
B.2.1 RST controller	95
B.2.2 State feedback controller	95
B.2.3 Linear Quadratic controller	96
B.2.4 Linear Quadratic Gaussian controller	98
B.2.5 Iterative Learning Control	100
C Motor specifications	103

1 Introduction

1.1 Background

This thesis is based on efforts to enhance the control of one of TetraPak's motors. TetraPak is one of the biggest corporations in the world when it comes to packaging systems. They supply hundreds of different types of packaging, from cartons to PET bottles. They also develop their own processing solutions.

In an advanced manufacturing system there are lots of different factors that have to coincide. For optimal performance there is a very high demand set on all mechanical parts of the system. That is why it is in TetraPak's interest to upgrade their production lines regularly.

The motor on which this thesis is based is a direct coupled, synchronous motor. The advantages it has to its predecessor is that the old motor is not direct coupled. A geared motor usually performs better in control applications, but is more expensive in terms of wearing damage. If it is possible to control the new motor within the same criterium as the old motor, the repairing costs could be reduced.

The problem with controlling a direct coupled motor is that the torque is directly transferred from the motor to the machinery. This means that the motor has to be very precise in its movements. If all the disturbances and dynamics are taken into consideration, it is easily realised that this is a difficult problem.

1.2 Specifications

Controlling a motor within certain specifications can be very difficult. Since the purpose of the motor is already decided, the specifications given are the same as those of the other motor, already in use. The conclusion of this is that the desired precision can be difficult to achieve with a motor with worse controlling capabilities.

With a load disturbance equal to the nominal torque of the motor, the stationary error must not be greater than 0.1 mm at the peripheral of the wheel attached to the motor. The nominal torque appears at a current of 3 A which corresponds to a pressure of about 15 kg at the peripheral of the wheel. The wheel is attached to the motor only during the testing period. Its purpose is to act as a symmetrical load on the system giving a simple model of the real process.

The same criterium holds for the reference following. The position of the wheel, at the peripheral, must not deviate from the reference signal more

than 0.1 mm at any point.

There is also a demand that the reference following can be performed quickly. In other other words, the motor must turn from point A to point B as fast as possible, but still within the specifications. The distance from point A to point B should be about a sixth of a revolution. The time used has no precise maximum limit but it should take about 0.2 s.

A distance as small as 0.1 mm is difficult to appreciate accurately without any special measuring equipment. With the resolution used in this thesis, 0.1 mm will be equal to 13 units. This will be described further in section 2.2.

1.3 Our goals

Early in the project it was clear that the specifications would be hard to reach. Since no one ever had tried to control this motor before, it might as well be impossible. Therefore our ambition became to show if it was possible to reach the specifications. Of course it was desirable to fulfill all the specifications and at the end of the project duration show a controller that did. But since it could be impossible to reach the high demands it was as desirable to show that it was impossible. To show this would be the same as showing that this motor perhaps was not the right choice for the specific application and therefore could be excluded from further testing.

1.4 Report overview

To simplify the reading of this report we have tried to make it as structured as possible. In doing so the report has been separated into six major chapters followed by the conclusions, discussion, reference list and the appendixes. The different chapters have been divided into a method description part and a design part. The first part explains the basic theory, while the second part shows the design procedure used in this thesis.

Chapter 2 describes the process and all its complications. This means that you will get an appreciation of the conditions of the motor, the PLC, the computer and the sensor dynamics. This section will also discuss different disturbances like load disturbances, measurement noise and delays.

Chapter 3 covers the system identification. To be able to control the motor in most cases you need a model of the motor to work with, both for the simulations and the actual controller calculations.

Chapter 4 deals with the control problem. The control problem is the effort to reduce the effects of unwanted disturbances. Different controllers are described and implemented so it will be possible to see which of them

has the best characteristics.

Chapter 5 deals with the servo problem. In this section, the motor is subject to a reference step. This means that the motor will rotate and stabilize at a new angle. There are certain ways of doing this fast and accurately. Some of these will be considered here.

Chapter 6 handles different tests of the chosen controllers. To get an idea of what the controllers can and cannot do, the testing is a necessary part of the complete evaluation.

Chapter 7 is the implementation part. Everything that is going to be tested in the real process has to be written in the computer language C. This means that the control theory used in the previous sections has to be discretized. For some of the algorithms this was a tricky part and sometimes very difficult since the memory and calculation time limitations set boundaries for how complex the formulas could be.

In the appendices, the C and Matlab codes used have been gathered. The C code has been divided into the separate controllers and identification methods. Every section is explained to simplify the understanding.

2 Process description

2.1 Motor and load

The main part of the process is of course the motor. It is a 3-phase synchronous motor. The maximum torque is 311 Nm and the nominal torque is 64.5 Nm. This corresponds to a peak current of 17 A and a continuous current of 3 A. If these limits are exceeded there is a risk of damaging the motor. To be sure not to damage the motor the maximum current was set to 4 A throughout the project. This of course sets a limit of how fast position reference trajectories the motor will be able to follow.

The motor is mounted on a rig. There is a big wheel connected to the motor axis, that is the load is direct coupled to the motor axis. On the peripheral of the wheel up to eight steel weights can be mounted. The purpose of these weights is to create a big moment of inertia on the load. With all weights mounted the moment of inertia on the load is about 9 kgm^2 (section C).

2.2 Angular sensor

The motor is equipped with an angular sensor. The sensor has a theoretical resolution of 16 000 000 increments per revolution. However, the actual resolution depends on the control system used. In this thesis, the maximum resolution was 3 600 000 increments per revolution (section C).

There is no sensor to measure the angular velocity with. So, to get a value of the velocity, one way is to differentiate the position. That is, to take the difference between the actual position and the previous position and divide by the sampling period.

In the beginning of the work the resolution was set to only 3 600 increments per revolution. This worked well during the identification. But in the control part it did not suffice at all. So the resolution was increased to 360 000 increments per revolution. This is why some plots of the velocity are in the range of thousands while some plots are in the range of hundreds of thousands.

From now on in the report increments per revolution are referred to as units and increments per revolution/s are referred to as units/s.

2.3 Motor drive and PLC

On the back of the rig there is a motor drive. The drive provides the motor with a 3-phase current. The value of that current is determined by a

controller. There is a PD controller implemented in the drive. But in this project, it was neither possible to access that controller, nor to implement another controller in the drive. To solve this problem there is a PLC connected to the drive. The controllers implemented in this project run from the PLC.

To be able to compute the new control current the controller in the PLC needs to read the motor position from the drive. Likewise the drive needs to read the control current from the controller in the PLC. This communication is performed over a network. The drive and the PLC are supplied by B&R Automation. B&R Automation is a multinational company in the industrial automation business. TetraPak uses their control systems.

The PLC is programmed from a computer. All code is written in C. The different controllers are implemented in a program called B&R Automation Studio and then transferred to the PLC over a network.

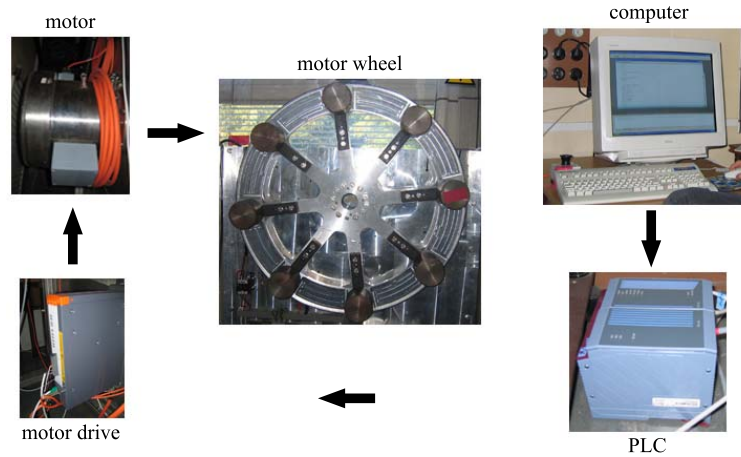


Figure 1: The programming code is downloaded from the computer to the PLC. The PLC calculates the current that the motor drive will send to the motor.

2.4 Time delay and sampling period

The fact that the controller cannot be implemented in the drive makes the control problem more difficult. A value of the current that is sent from the PLC does not affect the motor instantly. Due to the network connection between the PLC and the drive there is a time delay in the process. A time delay decreases the phase margin and therefore deteriorates the control performance.

The time delay could rather easily be measured in terms of samples by applying a step in the current and measuring how many samples it takes before the motor moves. This value is of course depending on the sampling period. At the beginning of the work the sampling period was set to 1.2 ms. Then the time delay was 6 sampling periods. But after about three months the PLC was replaced with a newer one. This led to a shorter time delay. But as the new PLC was faster it was possible to decrease the sampling period. The sampling period was set to 0.4 ms which resulted in a time delay of 6 sampling periods again. However, if TetraPak wants to use the controller that has been designed in this project it would be implemented in a B&R motor drive. The time delay would then disappear.

The change of PLC and sampling period may cause some confusion in this report. The first identification was performed with a 1.2 ms sampling period. After the change of PLC the sampling period of the control loop was changed to 0.4 ms. But it turned out to be difficult to get good identification results with such a short sampling period. Therefore the identification was placed in another loop with a 4 ms sampling period. The resulting models then had to be resampled to fit the sampling period of the control loop. This has not been a problem though, and it works excellently.

2.5 Disturbances acting on the process

There are two main types of disturbances, load disturbances and measurement noise.

It is quite difficult to foresee exactly what kind of load disturbances that are likely to occur in the process. However, Tetra Pak usually tests their controllers by applying a load disturbance as a step with a magnitude that corresponds to the nominal torque of the motor. For this motor that would be the same as adding a step load disturbance of 3 A at the input of the process. It is this load disturbance which is used in the simulations and also when simulating a load disturbance on the real process.

The measurement noise is a bit different. It is possible to get an idea of the characteristics of the noise by measuring the position of the real process. The noise does not appear to be very large when measuring the position. But the problem is that there is no velocity sensor in the motor, so the velocity cannot be measured but has to be differentiated from the position. And when differentiating, the small noise component of the position gets large in the computed velocity. As will be shown in chapter 4, the measurement noise will affect how small the position error can be without getting a too noisy control signal.

There have been made no attempts to construct a model of the measurement noise. Throughout the project the measurement noise has been considered to be white.

2.6 Nonlinearities in the process

The motor is affected by two nonlinearities. Friction is a very common nonlinearity that is hard to avoid in any application. The static friction in the motor sets the limit of how small currents that can make the motor rotate. To overcome the static friction a high initial current is needed.

The other nonlinearity makes its presence known at velocities above 400 000 units/s, or 4000 units/s in the older graphs (section 2.2). Data collected from velocities higher than this cannot be trusted.

3 System identification

3.1 The identification procedure

To efficiently control a device, a suitable model to describe it with has to be chosen. There is a wide selection of models to use depending on the number of inputs and outputs to the system and if you want to include some kind of noise model. In this thesis, four different techniques were tried.

Identifying a system usually consists of applying some kind of action to excite the unknown system and then interpret the response you get from the signal. With this motor, the action required is the current that drives the motor and the response is the position or the velocity of the wheel attached to the motor. In the construction of a model you record these data, the signal you send in and the response you get out, and apply it on a computing algorithm of choice to get a model description. Step responses and impulse responses could be used to excite the system, but they often do not give enough accuracy. This is due to the order of persistent excitation of the signal [4]. The higher the order, the more complex the model that can be identified. In this thesis there are two methods used, a relay technique based on nonlinear control [5] and identification with a PRBS signal [4].

When the data sets of the excitation signal and its response are available, it is time to decide which calculation method to use. Of course this area has a hoard of different approaches, but if you want a simple, reliable technique that is easily implemented in C, you can use least squares identification. The problem with that is that it lacks efficient noise handling and that it cannot cope with resonance peaks in a good way. If there are resonance peaks in the process, an alternative is to use subspace identification. This method is better at identifying complex dynamics. If the process parameters change with time, one option is to implement recursive least squares identification. This method updates the process parameters regularly.

When a process is described in terms of an equation, the more complex the description, the more precise the model can be. But a high model order can cause trouble when designing the controller. Therefore it is desirable to see whether it is possible to reduce the model to a lower order and still keep a high accuracy. If this is the case, a model reduction can be performed with, for example, the balanced realization technique [4].

The final step of the identification procedure is usually to validate the correctness of the model. This can be done in several different ways. In this thesis we have mostly used the cross validation simulation technique [4].

3.2 Modeling of a motor

When doing system identification of a motor you already have some pre-knowledge of the structure of the process. A typical transfer function from current to velocity of a motor consists of one pole and a gain

$$G(s) = \frac{K}{s + p}$$

This would be true if the action of the motor was transferred to the load without any dynamics in the coupling in between. But for this particular motor this is not true due to the big forces needed to rotate the mass of the load. Although the coupling is made of solid steel it will be flexible and give rise to a resonance in the transfer function from current to velocity. A resonance is the same as two complex poles, which makes it a third order system. This can be written as

$$G(s) = \frac{K}{(s + p_1)(s^2 + p_2s + p_3)}$$

The influence of this resonance is depending on the mechanics. One task of the project was to investigate how big this influence is.

3.3 Identification methods

3.3.1 Relay identification

The relay identification method is a nonlinear technique based on the describing functions method. A describing function is a way to describe a nonlinearity in a system. Since the relay is a well known nonlinearity, it is an easy task to calculate its describing function. The idea is that you get the motor to oscillate with constant amplitude and frequency. It can be done with the help of a relay with hysteresis. This state of stable oscillations is called a limit cycle. When the process reaches a limit cycle it means that both the relay and the process have the same frequency in the oscillations and this can be used to calculate the model parameters. The applicability of this method depends on the complexity of the model you wish to identify. Since it only uses one equation of equality it is only possible to have one unknown pole in the linear process. If there are more than one unknown pole you need additional techniques. The static gain will have to be determined in another way. It can, for example, be a good choice to use a step to retrieve it [5].

3.3.2 Least squares identification

Least squares estimation is based upon linear regression. Linear regression is a way to find a relationship between different variables. In this case it is the relationship between the input (current) and the output (velocity). This relationship can be expressed with the model

$$y(t) = \phi^T \theta + e(t)$$

where $y(t)$ are the observations, ϕ are the so called regressors, θ are the parameters to be estimated and $e(t)$ is the error between the observations and the linear regression model.

With a set of observations you get the matrix notation

$$Y_N = \Phi_N \theta + e$$

For a certain parameter estimation $\bar{\theta}$ you get the error vector

$$\varepsilon(\bar{\theta}) = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_N \end{pmatrix}$$

This is called the prediction error. It is easily understood that you want the prediction error to be small in order to get an accurate model. One way to achieve this is to use least squares estimation. This method minimizes the sum of the squared errors between the model and the observations, that is

$$\min V(\bar{\theta}) = \min \frac{1}{2} \varepsilon^T \varepsilon = V(\hat{\theta})$$

In other words, you want to find the parameter estimation $\hat{\theta}$ that minimizes the above expression. The solution to this problem is given by the equation

$$\hat{\theta} = (\Phi^T \Phi)^{-1} (\Phi^T Y)$$

An advantage with least squares estimation is that it is rather easy to compute the parameter estimations. All you need to do is matrix multiplication and inversion. This fact makes least squares estimation an attractive option. However, there are probably other methods, like subspace identification, that finds a model to describe a process with a resonance peak more accurately [4].

3.3.3 Subspace identification

The idea with subspace identification is to create a state space model directly from input-output data. A great advantage with subspace identification is that you do not have to know anything about the model structure, except for the order of it. Instead of using a Hankel matrix with Markov parameters you use two hankel matrices, one with the input data, U_h , and the other with the output data, Y_h .

$$U_h = \begin{pmatrix} u_k & u_{k+1} & \dots & u_{k+s-1} \\ u_{k+1} & u_{k+2} & \dots & u_{k+s} \\ \vdots & \vdots & \ddots & \vdots \\ u_{k+r-1} & u_{k+r} & \dots & u_{k+s+r-2} \end{pmatrix}$$

$$Y_h = \begin{pmatrix} y_k & y_{k+1} & \dots & y_{k+s-1} \\ y_{k+1} & y_{k+2} & \dots & y_{k+s} \\ \vdots & \vdots & \ddots & \vdots \\ y_{k+r-1} & y_{k+r} & \dots & y_{k+s+r-2} \end{pmatrix}$$

Then you introduce the two statevectors X_1 and X_2 with a relative timeshift.

$$X_1 = \begin{pmatrix} x_k & x_{k+1} & \dots & x_{k+s-1} \end{pmatrix}$$

$$X_2 = \begin{pmatrix} x_{k+j} & x_{k+j+1} & \dots & x_{k+j+s-1} \end{pmatrix}$$

Together with an extended observability matrix, C_r , and a lower triangular Toeplitz matrix, D_r , the two equalities

$$Y_{h1} = C_r X_1 + D_r U_{h1}$$

$$Y_{h2} = C_r X_2 + D_r U_{h2}$$

can be formulated. Solving these two equations for X_1 and X_2 using singular value decomposition you get an overdetermined system.

$$\begin{pmatrix} x_{k+1} & \dots & x_{k-j-1} \\ y_k & \dots & y_{k-j-2} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x_k & \dots & x_{k-j-2} \\ u_k & \dots & u_{k-j-2} \end{pmatrix}$$

From this system the A, B, C and D matrices in the state space solution can be calculated. [4]

3.3.4 Recursive least squares identification

Recursive least squares identification has many similarities to least squares identification. The purpose is the same, to minimize the sum of the squared

errors. The difference is that the computations are performed online. The parameter values are updated in each loop. In order for the computations to be sufficiently fast it is necessary to find a recursive update algorithm of the parameters. By introducing the notations

$$\begin{aligned} P(t) &= (\Phi_N^T \Phi_N)^{-1} && \text{(covariance matrix)} \\ K(t) &= P(t)\phi(t) \end{aligned}$$

and rewriting the resulting equation of the least squares identification

$$\hat{\theta} = \left(\sum_{i=1}^t \phi(i)\phi^T(i) \right)^{-1} \left(\sum_{i=1}^t \phi(i)y(i) \right) = P(t) \sum_{i=1}^t \phi(i)y(i)$$

the parameter update could be written as

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \phi^T(t)\hat{\theta}(t-1))$$

$\hat{\theta}(t)$ is now depending on the value of $\hat{\theta}(t-1)$. But $K(t)$ and $P(t)$ must also be updated accordingly. By use of the Matrix inversion lemma a complete updating algorithm for recursive least squares identification is achieved:

$$\begin{aligned} \hat{\theta} &= \hat{\theta}(t-1) + K(t)(y(t) - \phi^T(t)\hat{\theta}(t-1)) \\ K(t) &= P(t)\phi(t) = P(t-1)\phi(t)(I + \phi^T(t)P(t-1)\phi(t))^{-1} \\ P(t) &= (I - K(t)\phi^T(t))P(t-1) \end{aligned}$$

Now the whole calculation can be performed online by use of the previous values of $\hat{\theta}(t)$, $K(t)$ and $P(t)$. The recursive approach implies that initial values of $\hat{\theta}(t)$ and $P(t)$ need to be set.

In the algorithm above the process parameters are assumed to be constant. If the parameters change with time an alternative is to introduce the so called forgetting factor λ . Then, data that is n time units old are weighted by the factor λ^n . This implies that old data will not be taken into account as much as newer data. Therefore the parameter estimations will adapt faster to changes in the process. The recursive least squares identification with forgetting factor can be implemented with small modifications of the above algorithm [6].

3.3.5 Validation

The validation of a model is a necessary step if you want to verify that the model is a good approximation of the process. There are a wide variety of techniques for doing this where most of them focus on the residuals. The

residuals are vectors that describe the difference between the correct values and the approximated values. This difference is usually plotted together with some sort of restriction of how big it is allowed to be. It is a very straightforward way of observing if the model is adequate. A simple way to check the residuals is to use the command `resid` in Matlab.

The method mainly used in this thesis is cross validation simulation. In this method a comparison is made between the output from the process and the output from the model. These two data sets are plotted in the same graph so that it is easy to compare them. The output from the model is obtained by applying the same input to a simulation of the process as to the real process. This is done in the Matlab environment with the command `idsim`. For this command to work correctly, it is necessary first to calculate the correct initial values. For this purpose the command `dac2bdx` can be used.

If the validation shows that the model is not accurate enough to describe the process, the reason can be one of the following things: either the identification technique used does not work well with the process or the order of the model is not the correct one. A third reason is of course that the motor is too complex so that it is impossible to get a model good enough, but then you have to lower your demands.

3.3.6 Model reduction

The term model reduction could have a number of different interpretations. This section will cover model order reduction in a linear system, as this is what is relevant for this project. Two different methods will be described. First model reduction from a balanced realization is described, followed by the method of dominating poles.

After having created a state space model of a certain order, it would be interesting to see how significant the different states are with respect to each other. One way to do this is to transform the system to the balanced realization form. This is done by finding a transformation matrix T such that the reachability gramian P equals the observability gramian Q [4]. The new system is then described by

$$\begin{aligned} z(k+1) &= \Phi'z(k) + \Gamma'u(k) = T\Phi T^{-1}z(k) + T\Gamma u(k) \\ y(k) &= C'z(k) = CT^{-1}z(k) \end{aligned}$$

When the system is on balanced realization form, it is possible to investigate the relevance of the different states by looking at the diagonal elements of the Gramian $\Sigma = P = Q$. A small value of an element shows that the

corresponding state has a low effect on the input-output behavior. It is then natural to assume that these states could be removed without changing the accuracy of the model very much. A rule of thumb says that a state could be removed if the corresponding element is at least a factor 10 smaller than another element.

When eliminating a state i in the state space representation you set $x_i(k+1) = x_i(k)$. This means that $x_i(k)$ can be written as a function of the states that will remain and the input signal $u(k)$. Then $x_i(k)$ is replaced in the remaining states with the resulting expression. $x_i(k)$ should also be replaced in the expression of $y(k)$. This implies that there will be a direct term D in the state space system [4].

Another way to perform model reduction is the method of dominating poles. The idea is to look at the transfer function and simply eliminate the fast poles. Then the transfer function must be compensated with a factor to get the same static gain. After this, the denominator of the transfer function will only consist of the dominating poles. This method is not as satisfying as the balanced realization method, but it was used in the implementation to save time.

3.4 Model design

3.4.1 Relay identification

The equality discussed in the identification methods part (section 3.3.1), which this method depends on, is the equation $G(i\omega) = -1/N(A, \omega)$, where $G(i\omega)$ is the linear model of the motor and $N(A, \omega)$ is the describing function. The frequency, ω , was measured by counting the number of computer cycles from the moment the motor passed its zero position in the positive direction until it happened the next time. When a measurement differed less than 0.005 1/s from the previous one, that value was stored and the measurement was finished. The amplitude was measured simultaneously by just saving the highest absolute value of the position (fig. 2).

In this case the linear model of the motor is described by

$$G(i\omega) = \frac{K}{i\omega + a} e^{-i\omega L}$$

where K is the gain, a is the process pole and L is the time delay of the system. The model input is the control current and the output is a velocity. K can be given as the quotient between the amplitude of the step in the input signal and the amplitude of the response in the output signal. However, it turned out that this did not work so well. The reason was that to overcome

the static friction a quite large step was needed in the input signal. And a large step led to stationary end velocity that was above the linear region, which is velocities faster than 400 000 units/s. Therefore another method was tried. An RST controller (section 4.3.1) was used to drive the motor to a high velocity (but still in the linear region) and then step down to a lower velocity. By dividing the difference in velocity with the difference in current a value of K was achieved (fig. 3). Still it was uncertain though, if this value was correct since it differed quite a bit from the simulated results in Matlab.

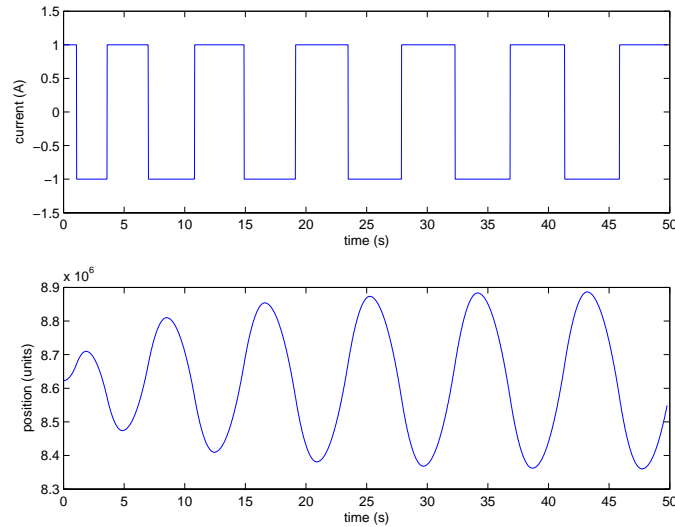


Figure 2: Identification using a relay with hysteresis.

The describing function of the relay is written as

$$1/N(A, \omega) = \frac{\pi}{4d} \sqrt{A^2 - \varepsilon^2} - i \frac{\pi \varepsilon}{4d}$$

with ε as the hysteresis, A as the amplitude of the input signal and d as the amplitude of the relay. These parameters are already known in the experiment. The equality from the beginning of this section now gives

$$\frac{K}{i\omega + a} e^{-i\omega L} = \frac{\pi}{4d} \sqrt{A^2 - \varepsilon^2} - i \frac{\pi \varepsilon}{4d}$$

with a and L as the unknown parameters. These two can be acquired by comparing the gain and the phase of the two sides.

$$a = \sqrt{(K |N(A)|)^2 - \omega^2}$$

$$L = \arctan \frac{\omega}{a} - \arctan \frac{\varepsilon}{\sqrt{A^2 - \varepsilon^2}}$$

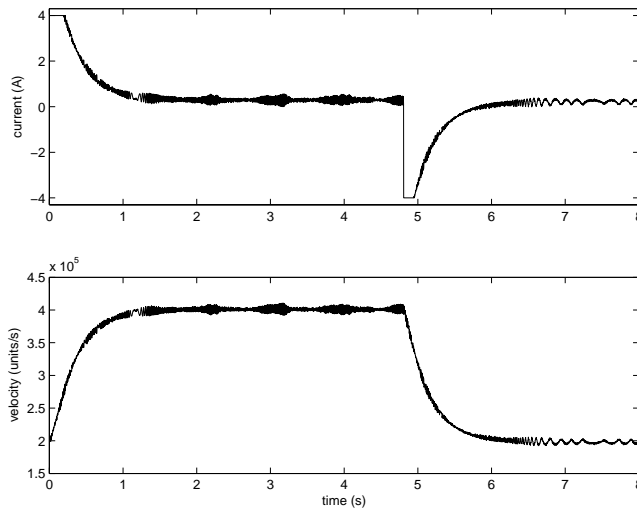


Figure 3: A step from a high velocity to a low velocity in order to determine the static gain of the system.

This gives the model of the motor. If you want to control the position instead of the velocity just add an integrator in the model.

3.4.2 Least squares identification

Least squares estimation is described in section 3.3.2. In Matlab it is performed with the command `arx`. But when using `arx` you have to decide which model order to use. Another option is to use the two commands `arxstruc` and `selstruc` to investigate the behavior of different model orders (fig. 4).

The diagram in figure 4 shows how accurate the models of different order are when applied to this particular data set. A low column means that the corresponding model order has high accuracy. It is seen that the accuracy increases with increasing model order. However, the model shall not only be able to predict this data set but all data sets from the motor. Therefore you should not pick a model with too high an order. Instead the model that is placed in the so called knee should be picked, where the accuracy is high but the model order is still not too high [4]. That model corresponds to the third column from the left, which is a model of order three. In section 3.2, it has already been predicted that model order three would be a reasonable choice.

A third order model of the process was created with the command `arx`.

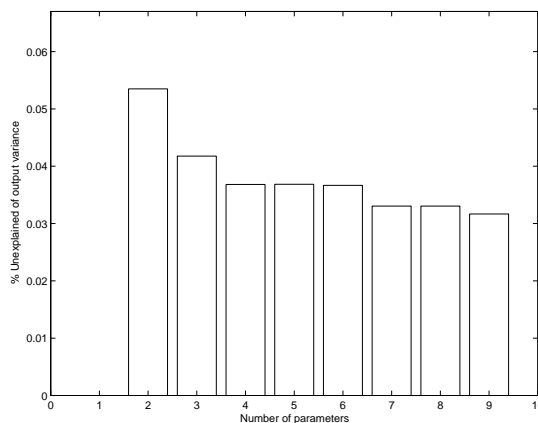


Figure 4: `Arcstruc` and `selstruc` are used for the decision of model order. The recommended model order is located in the knee.

The input-output data were collected from a loop with the sampling period 4 ms. Hence the model is also created with the sampling period 4 ms. In polynomial form, the third order model is

$$\begin{aligned}
 A(q)y(t) &= B(q)u(t) + e(t) \\
 A(q) &= 1 - 0.9458q^{-1} - 0.3929q^{-2} + 0.3391q^{-3} \\
 B(q) &= 295.1q^{-1}
 \end{aligned}$$

In the bode diagram (fig. 5), the resonance peak generated by the flexible coupling between the motor and load is visible, but not very large. It would now be interesting to see how significant the different states are. Then the system must be transformed to a balanced realization. This is done in Matlab with the command `dbalreal`. `Dbalreal` automatically computes the elements of the gramian Σ .

$$\Sigma = \begin{pmatrix} 450828 \\ 775.2 \\ 201.7 \end{pmatrix}$$

It is seen that the first state is highly dominating. The factor between state one and two is about 581. According to the arguments presented in section 3.3.6 it would be possible to reduce the system to a first order system and still keep a good model. The model reduction is performed with the command `dmodred`. The bode diagram of the resulting first order model is shown in figure 6 and its polynomial structure is

$$A(q)y(t) = B(q)u(t) + C(q)e(t)$$

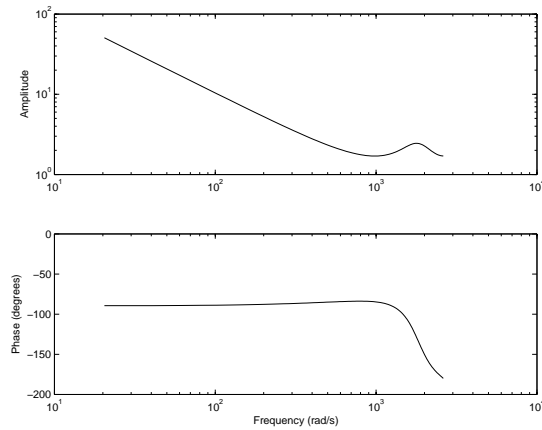


Figure 5: Bode plot of a third order least squares model. The sampling interval is 4 ms.

$$\begin{aligned}
 A(q) &= 1 - 0.99954q^{-1} \\
 B(q) &= 413.6q^{-1} \\
 C(q) &= 1 - q^{-1}
 \end{aligned}$$

The C polynomial contains a model of the noise. But this polynomial is discarded since the noise is considered to be white. The resonance peak

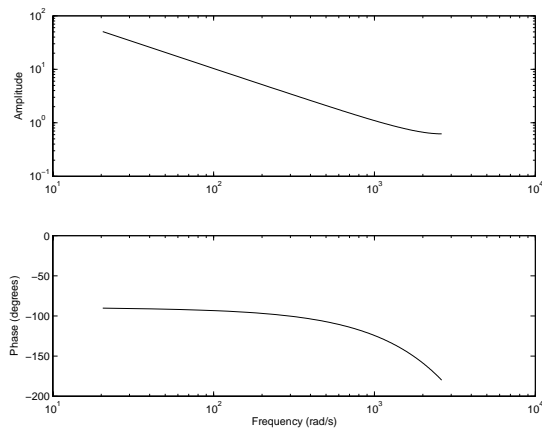


Figure 6: Bode plot of a first order least squares model. The sampling interval is 4 ms.

is no longer visible. This was expected, since the information about the resonance was contained in the states that were reduced.

In order to be able to use the model for control purposes, the model must be resampled to a 0.4 ms sampling period. This is necessary since the controllers are running in a 0.4 ms loop. Resampling is performed with the Matlab command `d2d`. The first order model with a 0.4 ms sampling period becomes

$$\begin{aligned} A(q)y(t) &= B(q)u(t) + C(q)e(t) \\ A(q) &= 1 - 0.99995q^{-1} \\ B(q) &= 41.36q^{-1} \\ C(q) &= 1 - q^{-1} \end{aligned}$$

When the least squares identification in Matlab was working and the model had been validated (section 3.5.2), the least squares identification was implemented in the PLC. The reason for this is that it is desirable for TetraPak to have an automatic identification procedure. Then the identification can be performed by anyone by a single click on a button.

The automatic least squares identification estimates a third order model. The model is then reduced to a first order model by the method of dominating poles. Finally the model is resampled to a 0.4 ms sampling period. This means that the difference between the Matlab identification and the least squares identification of the process model is the model reduction. However, it was found that the two reduction methods produced very similar results.

3.4.3 Subspace identification

The procedure in this case is a lot like the approach used in least squares identification. The data sets are retrieved by sending the PRBS signal through the motor and recording the output from the motor. The subspace model (section 3.3.3) is then obtained with the command `n4sid` in Matlab. `N4sid` gives the model on state space form.

A third order model seems like the natural choice since this was used in the least squares estimation and using the same order here will simplify comparing the different results. But there is also another reason for using a model of the third order and that is that we know the process has one real and two complex poles. The two complex poles are the cause of the resonance peak in the bode plot of the model (fig. 7). Reducing the order of this model is done in the same way as in LS estimation (section 3.4.2). In Matlab, you first make sure that the model is correctly balanced with the command `dbalreal`. Next step is to decide which order to reduce it to. The gramian Σ of the model gives a good glimpse of how relevant the different

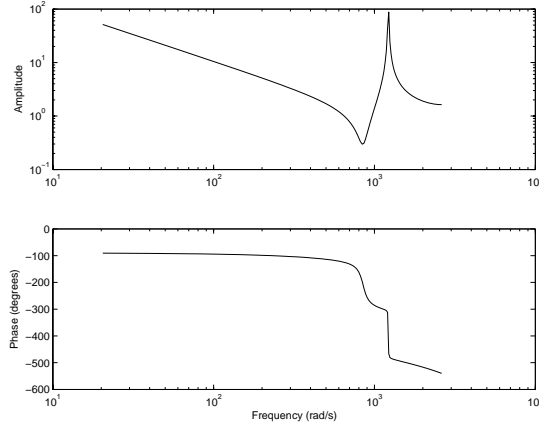


Figure 7: Bode plot of a third order subspace model.

model states are.

$$\Sigma = \begin{pmatrix} 5354.1 \\ 140.8 \\ 140.3 \end{pmatrix}$$

Since the most significant state is more than a factor ten bigger than the other states, the other states can be eliminated (3.3.6). The factor between state one and state two in this gramian is about 38 which is enough to eliminate the two smaller states. This method recommends that the third order model is reduced to a first order model. The model reduction is then achieved with the command `dmodred` in Matlab.

In the bode plot of the reduced model (fig. 8) you can see that a lot of information has been lost in the reduction.

3.4.4 Recursive least squares identification

The aim of the recursive least squares identification was to obtain a third order model of the process. The process can be described by the difference equation

$$y(k) = -a_1y(k-1) - a_2y(k-2) - a_3y(k-3) + bu(k-1)$$

Then the parameter estimation vector will be

$$\hat{\theta}(t) = \begin{pmatrix} a_1 & a_2 & a_3 & b \end{pmatrix}'$$

and the regressor vector will be

$$\phi(t) = \begin{pmatrix} -y(k-1) & -y(k-2) & -y(k-3) & u(k-1) \end{pmatrix}$$

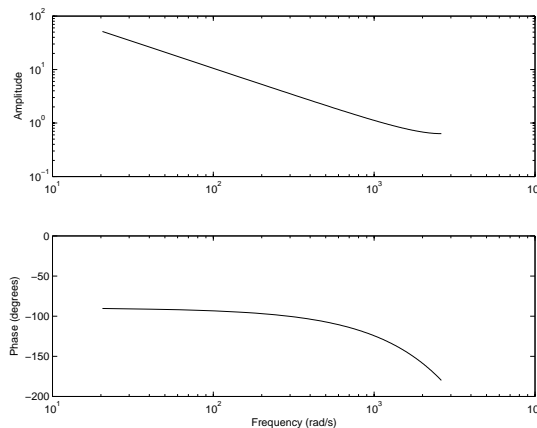


Figure 8: Bode plot of a first order subspace model.

The parameter estimations were then updated online, according to the algorithm in section 3.3.4. The sampling period of the identification loop was 4 ms. But before the RLS identification could be used, initial values of $\hat{\theta}(t)$ and $P(t)$ must be set. Since least squares estimation was already performed, it was possible to have an idea of what the reasonable initial values of $\hat{\theta}(t)$ could be. The initial values were set to be of the same magnitude as the result of the least squares identification. The initial covariance matrix was set to $10 \cdot I$.

The result of the RLS identification is highly dependent on how the process is excited. In order to test the implementation of the RLS identification, the process was excited with a PRBS signal. This is of course not the proper way to use RLS, but since this test should give about the same result as the batch version of the LS estimation, it would verify the correctness of the implementation. As seen in figure 9, the parameters converge to the desired value and are very stable. In figure 10, it is seen that the covariance matrix converges fast to low values. However, the estimation of b ($\theta[3]$) is a bit more uncertain than the other estimations. Although the tests showed that the RLS identification works fine it is difficult to validate it as a separate unit. Therefore the RLS identification will not be investigated more until it is used in conjunction with an adaptive controller in chapter 4.

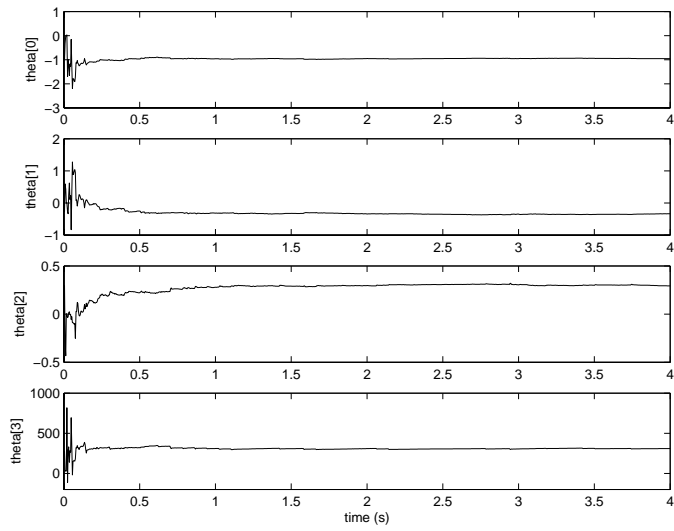


Figure 9: Parameter estimations when the RLS was excited with a PRBS signal.

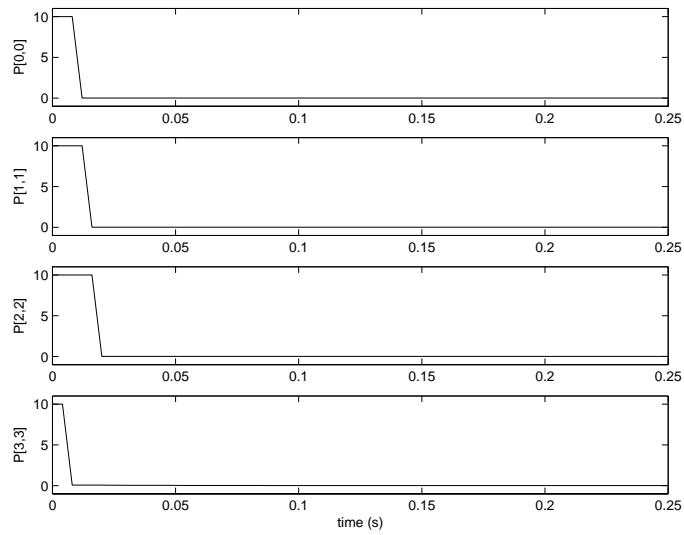


Figure 10: The value of the covariance matrix when the RLS was excited with a PRBS signal.

3.5 Model validations

3.5.1 Relay identification model

The model gained with the relay identification method is a bit uncertain. The reason for this is that the gain of the system was difficult to determine due to nonlinearities in the system. The two graphs in figures 11 and 12 show two models validated with the same data series. The difference between them is that the correctness of the models differs. This is a problem and cannot be allowed in a real environment.

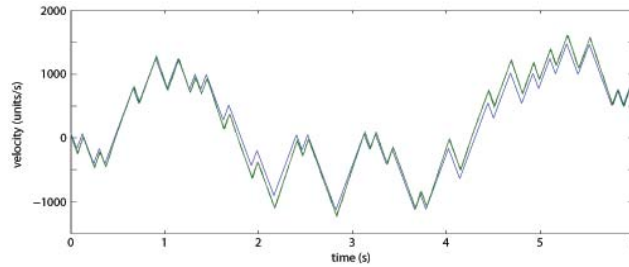


Figure 11: A first order relay model validated with a dataseries from the same experiment. The sample time is 0.4 ms.

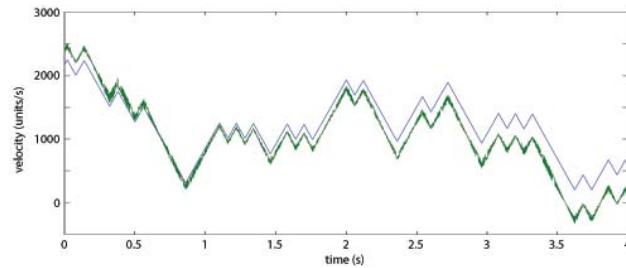


Figure 12: A first order relay model validated with the same dataseries as in figure 11. The sample time is 0.4 ms.

3.5.2 Least squares identification model

The cross validation of the least squares model shows that it is very good at following the process (fig. 13). There is very little difference in the cross validation between the third order model and the first order model. This confirms that the model reduction of the least squares estimation does not affect the accuracy of the model significantly.

One thing the least squares estimation does not manage to do so well is to model the resonances and the noise in the process. Even the third order model becomes a straight line when looking at a small area (fig. 14).

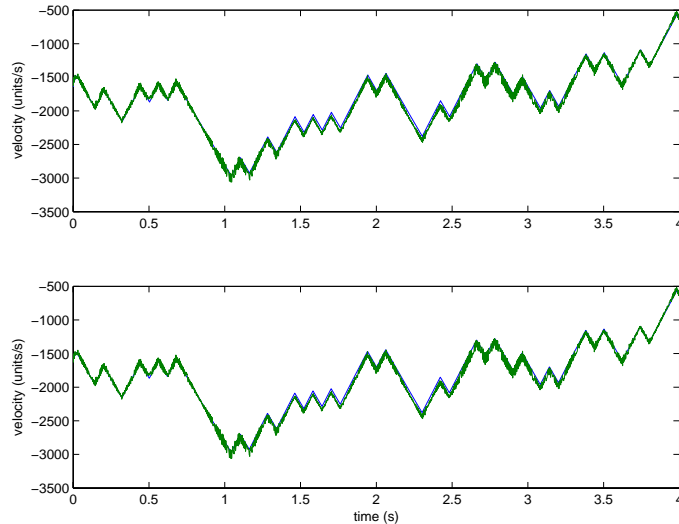


Figure 13: Cross validation between the true output from the process and the output from an arx model. The upper graph is a third order model, the lower is a first order model. The sampling period is 1.2 ms.

3.5.3 Subspace model

The cross validation shows that the the third order model and the first order model both estimate the true values with good results (fig. 15). It can also be seen that the third order model has a higher underlying frequency which the first order model has not. This is due to the complex poles in the third order model that were eliminated in the first order model.

3.6 Problems

Unknown forces are always a problem in modelling. In this project the friction forces in the motor made the work more difficult. To get a good estimate of the gain in the process using a step, it is important that the step is performed in a linear region. Because of the static friction it became unreliable to work with small currents.

Another problem was that the linear region of the motor did not stretch up to higher velocities. Above 400 000 units/s the measurements were no

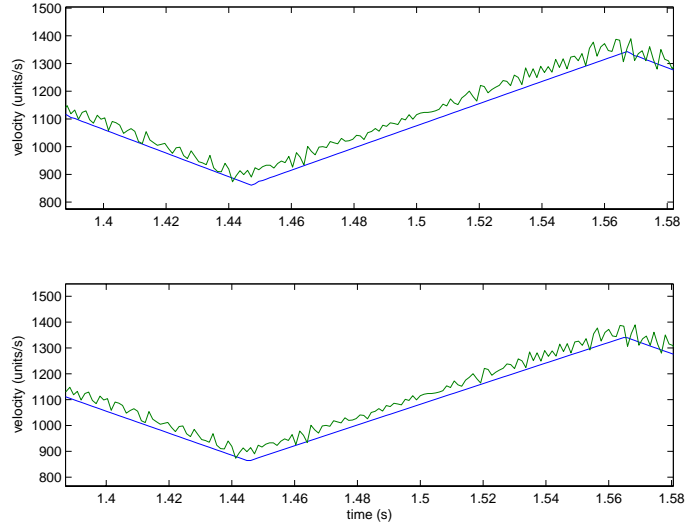


Figure 14: The same as figure 13 but a different scale. Note the poor estimation of the higher frequency oscillations.

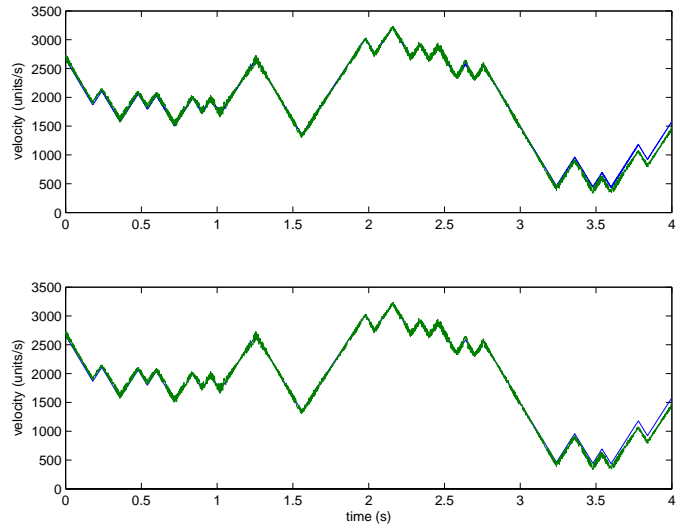


Figure 15: Crossvalidations between the true output from the process and the output simulated with the third order model (upper) and the true output from the process and the output simulated with the first order model (lower).

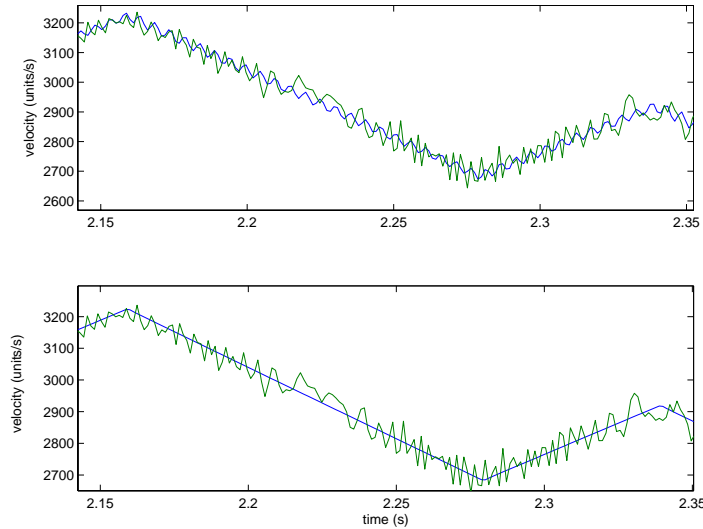


Figure 16: The same as figure 15 but a different scale. Note that the third order model can estimate the underlying frequency.

longer reliable. This made the approximated gain in the relay identification method very uncertain.

One thing rarely considered when applying a theory to a practical environment is the time delays that appear. Between the motor and the PLC drive there is a delay of about six sample periods, which equals 2.4 ms. This means that in the calculations, for every input signal the corresponding output signal comes six steps later. This leads to the fact that the phase margin is lowered which can lead to stability problems.

There was also a problem in the communication between the motor drive and the PLC. This problem first occurred when the old PLC was replaced with a new one. The problem was that the data sometimes was delayed one sample period more than usual. This led to that the values of the position and velocity were corrupted, so the results of the identification became bad. This problem was solved after consulting Tetra Pak and B&R Automation. It turned out that the PLC needed to be synchronized with the motor drive. This was done by setting a timer option. After that everything worked fine.

3.7 Summary

There are a couple of factors that should be considered when choosing the identification technique. Mainly it depends on how accurate the technique is. In this project, three of the techniques are done off line while the recursive

least squares is done on line. The continuous technique has the advantage that it can follow process variations, like wearing, during the process. This is of course a good property but sometimes unnecessary and time consuming if the process does not change with time.

The relay identification method gives very varying results in the step responses why in this case it is an unreliable method. It is necessary that the model estimations can be trusted so the controller will work correctly.

The most accurate of the techniques is the subspace identification. As seen in the validation in figure 16, it has modelled the resonance peak more precisely than the other methods. But since this accuracy is not kept in the reduced model, it will still give the same results as the least squares model.

Another obstacle is that it all has to be implemented in discrete time in the B&R Automation Studios environment. Besides the problems that a conversion to discrete time can cause, there is also a limit of how much memory and calculation time there is for the calculations. This can be a problem since the subspace model is more complex than the others. And the lesser memory space and calculation time used, the better.

RLS and LS identification are the most appropriate methods to use in this application. The difference between the two is that RLS will adapt to process changes during the run. This is, however, only a benefit if the process is exposed to variations and can otherwise reduce the accuracy of the model estimations. In this thesis, both methods have been used for control purposes. Due to the small differences between the third order models and their corresponding reduced order models, the reduced order models have been used for the control design.

4 The control problem

4.1 Motivation

In controller design a natural approach is first to design a robust controller and then adjust that controller to being able to follow a certain reference trajectory. This chapter deals with the problem of designing a robust controller. In particular, the aim is to make the controller insensitive to big load disturbances. This is necessary due to the specification that the position can deviate a maximum of 0.1 mm at the peripheral of the wheel when there is a load disturbance of the same magnitude as the nominal torque of the motor.

Throughout this chapter, load disturbances are simulated as a step at the input that goes from 3 A, to zero, to -3 A and then to zero again according to figure 17.

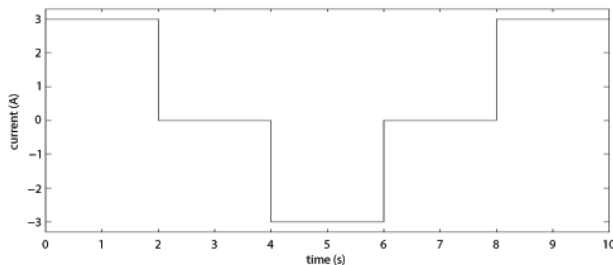


Figure 17: Simulated load disturbance.

Since it is the position of the motor that is to be controlled, all controllers but the RST controller are position controllers. The reason why the RST controller is implemented for velocity control is that it is only used as a tool in the relay identification.

All the controllers in this chapter, except for the linear quadratic self-tuning regulator, were first simulated in Matlab Simulink. If the results in Matlab Simulink were promising, the controller was implemented in the real process. As could be seen in the plots in this chapter, there is a very good correspondence between the simulations and the real process. The linear quadratic self-tuning regulator was implemented in C code directly.

All controllers are dependent on a good model. For the RST controller, the state feedback controller, the linear quadratic controller and the linear quadratic gaussian controller the model used was obtained by the batch version of least squares identification, as described in section 3.4.2. Since the least squares identification was implemented in the PLC, the controllers

acting on the real process use a model that is automatically generated in the PLC. When simulating the controllers, a corresponding Matlab model is used. The linear quadratic self tuning regulator on the other hand, uses a model given by the recursive least squares algorithm described in section 3.3.4.

4.2 State space model

In order to perform position control one must have a model of the process from current to position. However, all identification was made from current to velocity. So first a discrete state space model of the process from current to position had to be created. There are two equations needed to do this. The first one is the discrete first order transfer function from current to velocity given by the least squares identification. With the states $x_1 =$ position and $x_2 =$ velocity, the input u as the current to the motor, B and $pole$ as the gain and the discrete pole from the identification, this equation will be

$$x_2(k+1) = pole \cdot x_2(k) + Bu(k)$$

The second equation computes the velocity from the actual position and the previous position. With h as the sampling period this equation is given by

$$x_2(k) = \frac{1}{h}(x_1(k) - x_1(k-1))$$

By combining these two equations the discrete second order state space model of the process becomes

$$x(k+1) = \begin{pmatrix} 1 & h \cdot pole \\ 0 & pole \end{pmatrix} x(k) + \begin{pmatrix} B \cdot h \\ B \end{pmatrix} u(k)$$

$$y(k) = \begin{pmatrix} 1 & 0 \end{pmatrix} x(k)$$

4.3 Controller methods

4.3.1 RST controller

If you know what characteristics you want your controller to have, the RST controller can be a good choice. When designing it you choose where you want the poles and zeros of the closed loop system to be. The open loop system can be described by

$$A(q)y(k) = B(q)u(k)$$

where $A(q)$ and $B(q)$ are polynomials. The controller is described by

$$R(q)u(k) = T(q)u_c(k) - S(q)y(k)$$

where $R(q)$, $S(q)$ and $T(q)$ are polynomials of the same order. Combining these two equations and approximating will lead to the Diophantine equation

$$A(z)R(z) + B(z)S(z) = A_{cl}(z)$$

In this equation, $A_{cl}(z)$ is chosen to be the closed loop characteristic equation of choice. Solving this equation will give the controller polynomials R and S from which you can calculate the polynomial T . The schematics for a simple RST controller can be viewed in figure 18. [3]

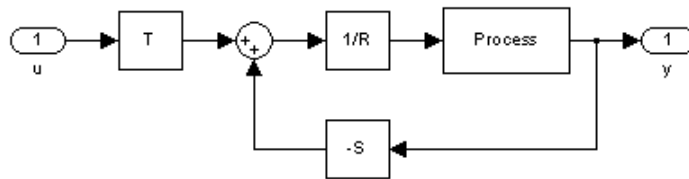


Figure 18: Schematics for a RST controller

4.3.2 State feedback control

State feedback control is a well known design method (fig. 19). Assume that the process is given on state space form

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) \\ y &= Cx(k) \end{aligned}$$

The control law is $u(k) = l_r r - Lx(k)$.

The discrete closed loop characteristic polynomial is given by $\det(zI - A)$. The idea is to determine the poles of the closed loop characteristic polynomial and then compute L according to that. l_r will then be computed so that $y = r$ in stationarity. [1]

4.3.3 Linear quadratic control

With the pole placement technique the poles can be placed anywhere, but there is still a drawback with that method. It is not possible to have unlimited control action and the different states must not grow out of control. It is desirable to be able to adjust how much control action to use and how

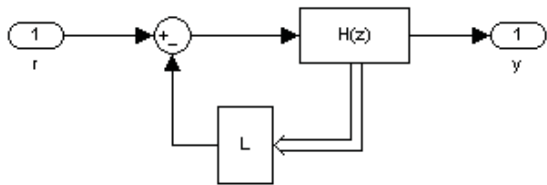


Figure 19: Schematics for a State feedback controller

large the different states are allowed to be. The LQ controller provides this option.

The structure of the LQ controller is the same as for the state feedback controller in section 4.3.2. The only difference is how the feedback vector L is computed. The LQ controller aims at minimizing the cost function

$$J = \sum (x'Qx + u'Ru + 2x'Nu)$$

The matrices Q , R and N could be seen as tuning parameters that are adjusted to achieve the desirable control performance. Q is a square matrix with the same size as the number of states in the model. For the case with one input, R is a scalar and N is a column vector with the same number of rows as the number of states. The diagonal elements of the Q matrix penalizes large deviations of the corresponding states. The other elements of the Q matrix are cross terms between the different states. The scalar R penalizes large control signals. N is usually set to 0.

For a certain choice of Q and R , L is computed by solving the Riccati equation [3].

4.3.4 Linear quadratic gaussian control

To be able to use the LQ controller, all states must be measurable. This is often not the case. Then one possibility is to use a Kalman filter to estimate the states. It is then the estimated states that are fed back through the L vector [1].

The structure of the Kalman filter is given by the equations

$$\begin{aligned} \hat{x}(k+1) &= (\Phi - KC)\hat{x}(k) + \Gamma u(k) + Ky(k) \\ \hat{y}(k) &= C\hat{x}(k) \\ u(k) &= l_r r - L\hat{x}(k) \end{aligned}$$

K is a vector that can be chosen in different ways. The LQG controller determines K so that the variances of the output and the control signal are minimized. L is determined in the same way as for the LQ controller [6].

4.3.5 Linear quadratic self-tuning regulator

The linear quadratic self-tuning regulator is an adaptive version of the LQ controller. The purpose is the same as for the LQ controller: to minimize the loss function described in section 4.3.3. Recursive least squares estimation is used to estimate the process parameters online. In order to achieve a self-tuning regulator the feedback vector L must also be computed online. This is done by solving the Riccati equation in each loop [6].

With the equation from section 4.3.3, the Riccati equation can be written as

$$\begin{aligned} S(k) &= \Phi^T S(k+1)\Phi + Q - L^T(k)(R + \Gamma^T S(k+1)\Gamma)L(k) \\ L(k) &= (R + \Gamma^T S(k+1)\Gamma)^{-1}(\Gamma^T S(k+1)\Phi + N^T) \end{aligned}$$

This equation can be solved iteratively. Another way to solve the problem is to consider the stationary case where $S(k+1) = S(k)$. Then the equation can be solved algebraically [3]. In this thesis the latter method will be used.

4.4 Controller design

4.4.1 RST controller

The simplest form of the RST controller was used. This means that the controller model was of order zero and the R , S and T polynomials were all constants. Because of the low order in the controller, there is only one available pole that can be placed. This will naturally make it easier to find the best choice of placement. But it will also limit the flexibility of the controller.

A pole placed on the real axis in $-3 - 5j$ gives the wanted qualities, a fast response to load disturbances and stability. A too fast response affects the robustness negatively and makes the controller unstable. This is because the controller will try to eliminate small loads with a big gain to get a fast response. But instead it gives a big overshoot and the gain will work in the other direction. This eventually leads to instability.

Note that this controller controls the velocity and not the position of the motor. The reason for this is that it is mainly used for controlling the step sequence in relay identification (section 3.3.1). A step in velocity can be viewed in figure 20.

A pole placed in -3 gives the following $R(q)$, $S(q)$ and $T(q)$.

$$\begin{aligned} R &= 1 \\ S &= 0.00003010395452284263 \\ T &= 0.00003090275477639738 \end{aligned}$$

The schematics for the zero order RST controller can be viewed in figure

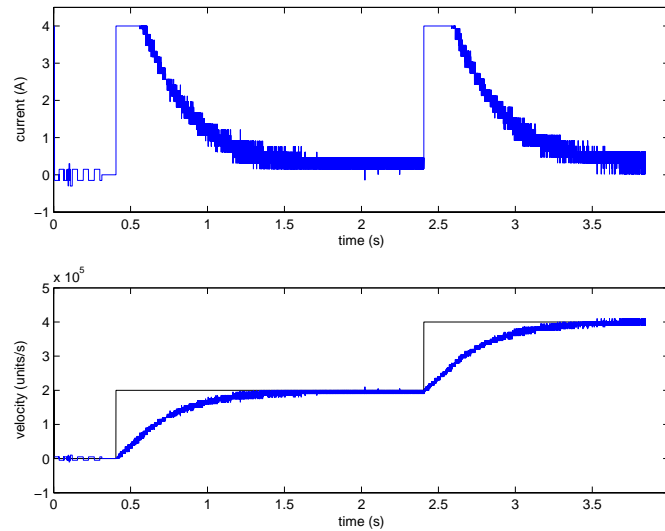


Figure 20: A step in velocity with the RST controller in the real process. The pole is in -3.

18.

4.4.2 State feedback controller

The aim of the state feedback controller was to perform position control. The process model used for simulations is described in section 4.2. For the state feedback controller the model is based on least squares identification. A Matlab Simulink model of the controller was designed according to figure 21. In the simulations the process model was obtained in Matlab, but when the controller was tested on the real process, the model was obtained by the automatic least squares algorithm implemented in the PLC. As described in figure 17, the load disturbance is simulated with a step that goes from zero to 3 A, back to zero and then to -3 A.

The controller considers the velocity to be measurable, although it is not measurable in the process but differentiated from the position as described in section 2.5.

The discrete closed loop characteristic polynomial, $\det(zI - A)$, is

$$z^2 + z(Bl_2 - pole - 1 + B \cdot h \cdot l_1) + pole - B \cdot l_2$$

After having determined the poles of the closed loop polynomial, p_1 and p_2 , l_1 and l_2 can be computed by comparing the coefficients. The controller

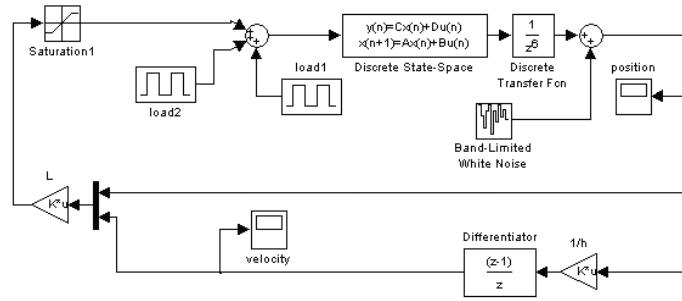


Figure 21: Simulation schematics for a state feedback controller

was tested with different sets of discrete poles. In figure 22, both poles are placed in -0.995 on the real axis. The result is very poor. Due to the bad results from the testings, this method was abandoned rather quickly.

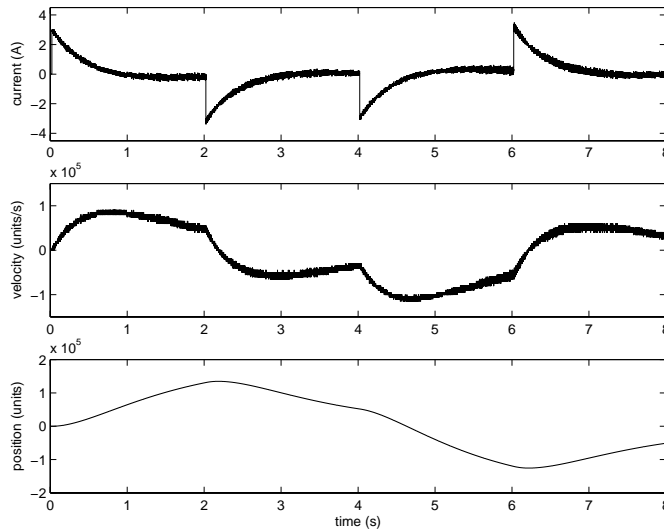


Figure 22: Both poles are placed in -0.995 on the real axis.

4.4.3 Linear quadratic controller

Since the tests with pole placement had failed, the LQ controller was tested instead. All efforts were concentrated on fulfilling the specification of load disturbance rejection, that is to keep the position within the limits of ± 13 units, as described in section 1.2. For the LQ controller the model is based on least squares identification. In the simulations the process model was ob-

tained in Matlab, but when the controller was tested on the real process, the model was obtained by the automatic least squares algorithm implemented in the PLC.

The structure of the LQ controller is the same as in the pole placement case. But for the LQ controller the feedback vector L was computed with the Matlab command `dlqr`. To use `dlqr` you must specify the weighting matrices Q , R and N in the cost function as described in section 4.3.3.

The tuning of these parameters was a rather time-consuming task. Some conclusions could be drawn at an early stage, though. The crossterms in the Q matrix did not seem to have any effect, so they were set to zero. The first element in the diagonal of the Q matrix would have to be bigger than the second term since the main goal was to keep the position error small. The R term needed to be several factors bigger than the other terms. This is natural since the values of position and velocity are a lot bigger than the values of the current. N was set to zero.

When these things were considered the tuning was very much a trade-off between small errors and a well behaved control signal. It is possible to get very small errors in the position and the velocity. But the control signal will then be very noisy, since the control signal is penalized to little in proportion to the position and velocity. When the control signal is not penalized enough the measurement noise will reach through the controller and affect the control signal. This is not acceptable since it will sound terrible and wear out the motor. For this reason the measurement noise in the process sets a limit for how small the position and velocity errors can be. If the measurement noise could be reduced it would be possible to get smaller errors. However, in this project the strategy used was to find a tuning that gave small position and velocity errors but still kept a nice control signal. The tuning ended with the following parameters

$$Q = \begin{bmatrix} 50000 & 0 \\ 0 & 5 \end{bmatrix}$$

$$R = 100000000$$

With these parameters, the current, velocity, and position look as in figure 23. The reference value of the position is zero, but the process is affected by a load disturbance as described in section 4.1. A problem that has already been discussed is that the noise has a big influence on the velocity signal. In figure 23 it is seen that it also affects the current. This implies that the current will switch signs all the time even though the motor is standing still, which is unacceptable. However, it was found that by filtering the velocity

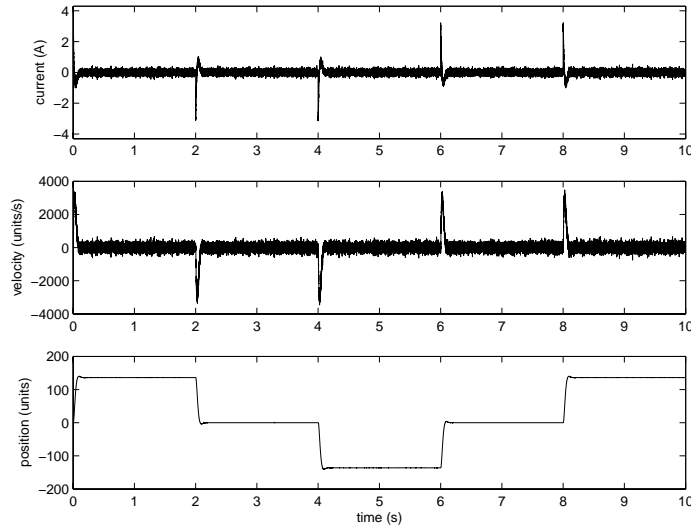


Figure 23: Current, velocity, and position of the simulated LQ Controller.

with a low pass filter, the noise could be heavily reduced. This is shown in figure 24.

As seen in the position graph, the lowest of the plots in figure 24, the position does not at all stay within the limit of ± 13 units. There is a peak and then the position stabilizes at about 135 units under a load disturbance. However, as described above it is not possible to tune the controller further without getting a too noisy control signal. Instead a lag filter was added to the controller that reduced the stationary error.

The purpose of a lag filter is to reduce the stationary error by increasing the low frequency gain. The price you pay is that the phase margin decreases.

A lag filter has the structure

$$\frac{s + a}{s + a/M}$$

A rule of thumb says that if a is chosen to be $0.1 \cdot \omega_c$, where ω_c is the cut-off frequency of the open loop system, then the phase margin will decrease with less than 6° . The value of M determines how much the stationary gain will be increased [1].

To get ω_c a bode diagram of the open loop system was drawn (fig. 25). It is seen that ω_c is 80.1 rad/s, and the phase margin is 20° . The lag filter was chosen to be

$$\frac{s + 8.01}{s + 8.01/50}$$

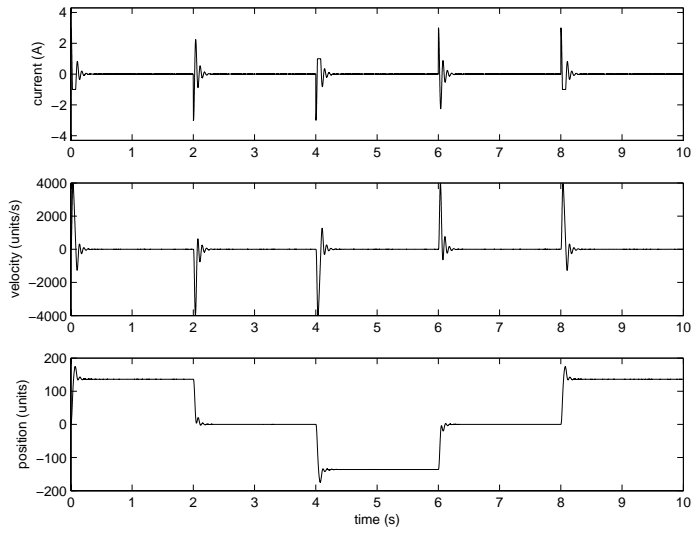


Figure 24: The velocity is filtered with a low pass filter in the simulation.

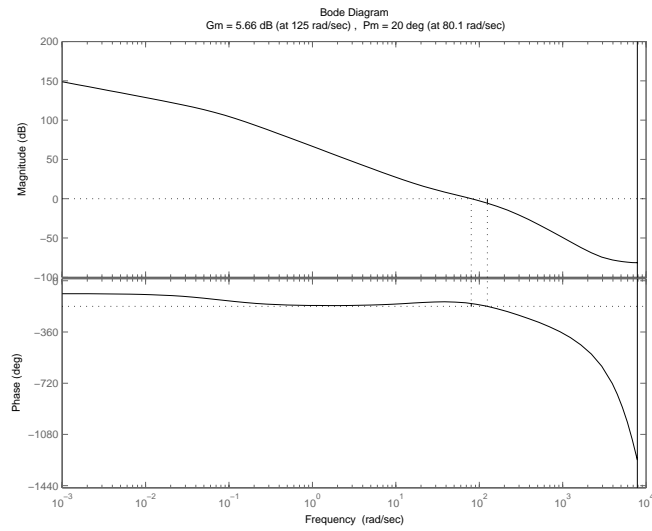


Figure 25: Bode diagram without a lag filter.

By adding the lag filter to the controller a new bode diagram could be drawn (fig. 26). The phase margin is now 14.4° , thus it has decreased with 5.6° .

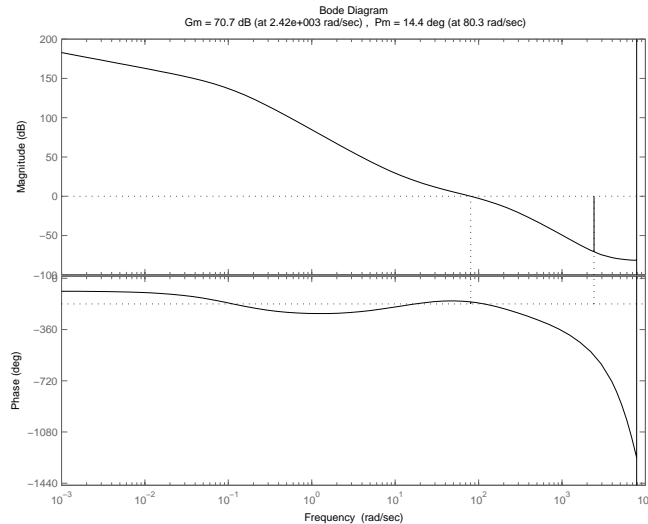


Figure 26: Bode diagram with a lag filter.

The stationary gain has increased.

With the lag filter, the current, velocity and position instead look as figure 27. The peak of the position error is the same as before, but the error decreases fast to a level within the limits of the specification. It takes at most 0.33 seconds to reach within the limits. The extended controller now looks like in figure 28.

The LQ controller with a low pass filter and a lag filter was implemented on the real process. Before this could be done the lag filter had to be discretized. This was done with the command `c2d` in Matlab. The result when the real process is affected by the same load disturbance as in the simulations is seen in figure 29. The correspondence between the simulation and the real process is in other words very good.

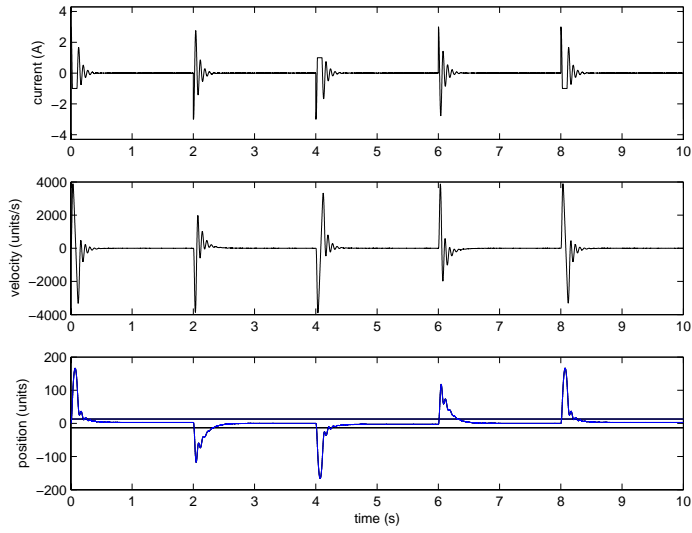


Figure 27: Current, velocity and position when a lagfilter is added to the simulation. For the position, the limits are plotted as well.

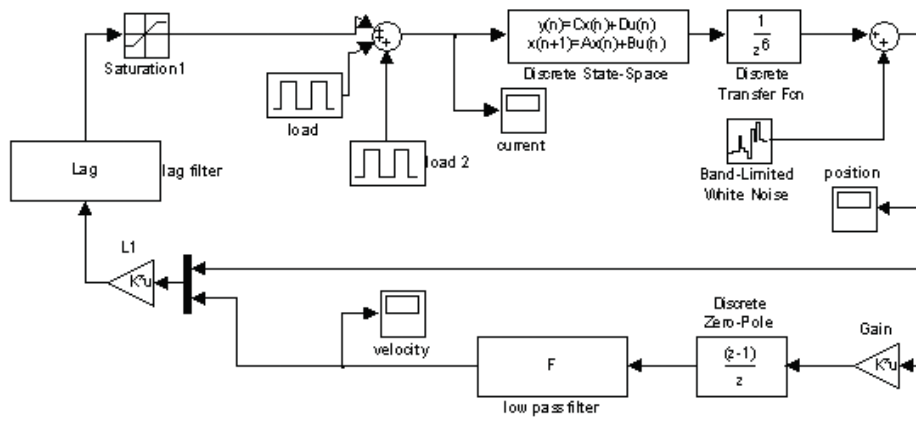


Figure 28: Schematics for the LQ controller

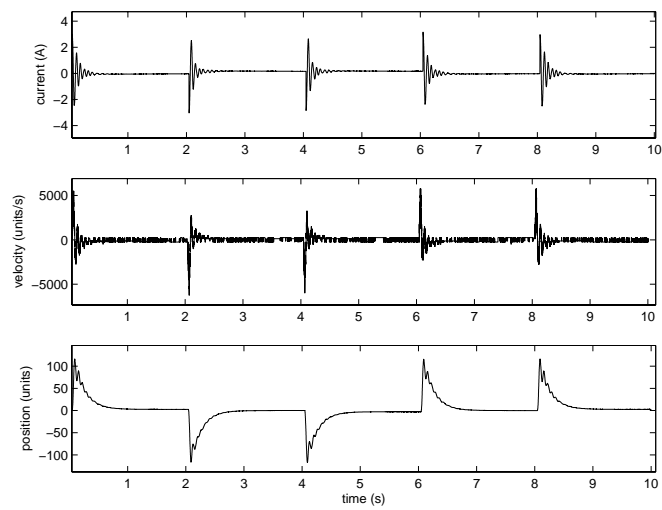


Figure 29: The real process affected by a load disturbance of 3 A. The LQ controller is complemented with a lag filter.

4.4.4 Linear quadratic gaussian controller

To be able to tune the controller further you must somehow reduce the effects of the measurement noise. One way to do this is to use a Kalman filter, as described in section 4.3.4. A Kalman filter can be added to the LQ controller as in figure 30. For the LQG controller the model is based on least squares identification. In the simulations the process model was obtained in Matlab, but when the controller was tested on the real process, the model was obtained by the automatic least squares algorithm implemented in the PLC. The K vector is computed in Matlab with the command `kalman`.

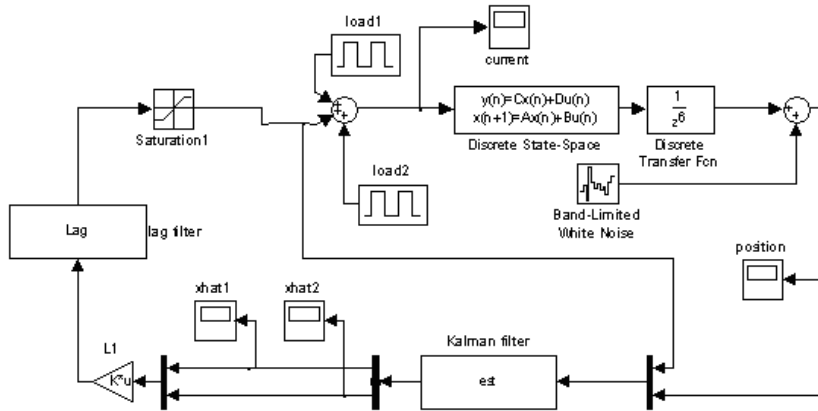


Figure 30: Schematics for the LQG controller

When using `kalman` the variance of the process noise Q_n and the variance of the measurement noise R_n must be specified. But since these values are not given, Q_n and R_n could be seen as tuning parameters.

The L vector is computed with the command `dlqr`, the same as for the LQ controller. Thus, for the LQG controller there are four tuning parameters, Q , R , Q_n and R_n . The principle is the same as when tuning the LQ controller: increase the parameter which corresponds to the variable you want to penalize more.

Quite soon it was found that there was a big problem with the estimations. The position estimation \hat{x}_1 worked fine, the effect of the measurement noise was reduced. But the Kalman filter did not manage to estimate the velocity correctly, due to the big load disturbance. Instead of the expected peak in velocity as the load disturbance changed, the estimation kept a high level as long as the disturbance was active. Due to this problem it was not possible to get as good a performance with this controller as with the LQ controller. An idea was to feed back the differentiated and filtered velocity

as before, and only use the Kalman filter to estimate the position (fig. 32). But this did not enhance the performance compared to the LQ controller. Therefore it was decided to abandon the LQG controller.

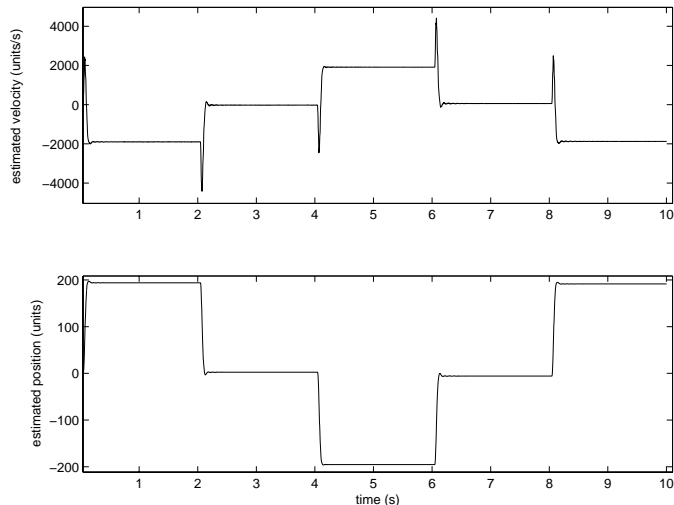


Figure 31: The velocity estimation \hat{x}_2 at the top and the position estimation \hat{x}_1 below.

4.4.5 Linear quadratic self-tuning regulator

As opposed to the other controllers in this chapter, the LQ STR was implemented directly in C code. No simulations were performed. For the LQ STR the model is based on recursive least squares identification, which is implemented in the PLC.

The main effort of the linear quadratic self-tuning regulator was to implement a Riccati equation solver. To make the problem a bit less time-consuming the resulting model of the RLS identification was reduced to a first order model. This was done by the method of dominating poles. The arguments that a model reduction can be performed are the same as for the ordinary least squares identification. When this was done the structure of the process model from current to position was

$$\begin{aligned} x_1(k+1) &= x_1(k) + h \cdot pole \cdot x_2(k) + B \cdot h \cdot u(k) \\ x_2(k+1) &= pole \cdot x_2(k) + B \cdot u(k) \\ y(k) &= x_1(k) \end{aligned}$$

where B and pole are the reduced parameters from the RLS identification

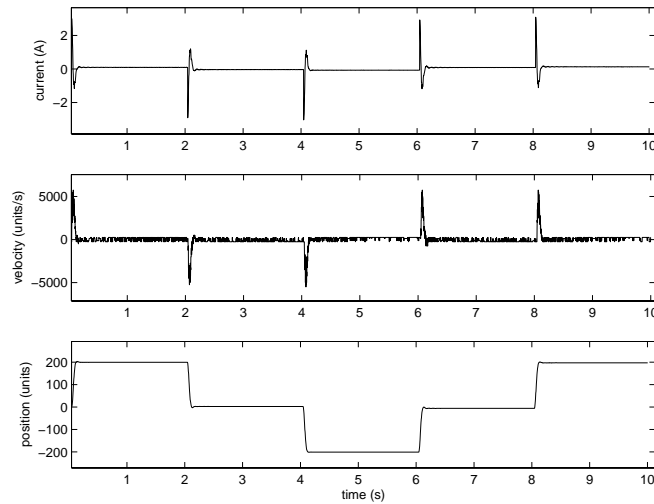


Figure 32: LQG controller with a load disturbance. The velocity is differentiated from the position and the Kalman filter is only used to estimate the position.

in section 3.4.4.

The Riccati equation to be solved is described in section 4.3.5. However, the Riccati solver implemented here presupposes some conditions to be fulfilled. Firstly, the stationary case where $S(k+1) = S(k)$ is considered. Secondly, the Riccati solver can only manage models of order two. Thirdly, the Q matrix must be symmetric. This is also the case when using `dlqr` in Matlab. A symmetric Q matrix implies that also S will be symmetric. The last condition is that N must be zero. This is the default choice of `dlqr`.

Assume that these conditions are fulfilled. Then S can be solved algebraically by inserting the expression of L into the expression of S . After that, L can be computed by using the obtained value of S .

The implementation was verified by giving the same input to `dlqr` in Matlab. Since the result was the same, the implementation was considered to be correct. By combining the RLS identification and Riccati solver with the LQ controller in section 4.4.3 a complete Linear Quadratic Self-Tuning Regulator was obtained.

The next step was to use the LQ STR to control the process. Since the RLS and Riccati equation solver operates in a 4 ms loop while the control is performed in a 0.4 ms loop, the resulting model from the RLS first had to be resampled to 0.4 ms before the new L was computed.

The LQ STR was tested in the same way as the ordinary LQ controller,

that is a load disturbance that changes from 3 to 0 to -3 A was added at the input of the process.

First the performance of the RLS identification was investigated. The parameter estimations are plotted in figure 33. The first load disturbance occurs after about 1 second. After that the parameters are very stable. But the value of the parameter estimations are different in this case compared to the parameters obtained by ordinary least squares identification. This is due to the different excitation of the process in the two cases. However, for an adaptive controller the most important quality is not the actual value of the parameter estimation, but the control performance. Figure 34 shows

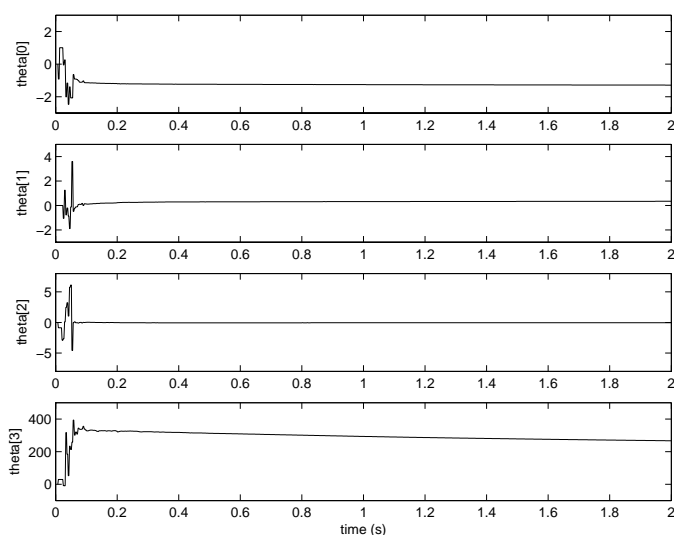


Figure 33: Parameter estimations when the process was excited with a load disturbance.

the diagonal elements of the covariance matrix. The values are converging fast to a value close to zero which indicates that the parameter estimations are accurate.

Finally, the control performance was investigated. The current, velocity, and position of the process are plotted in figure 35. The result is very similar to that of the ordinary LQ controller. But here, the peak of the position error is a little bit smaller. In figure 36 it is seen that the controller parameters, $lq1$ and $lq2$, are converging very fast and are very stable. The reduced parameters B and $pole$ are also plotted in figure 36.

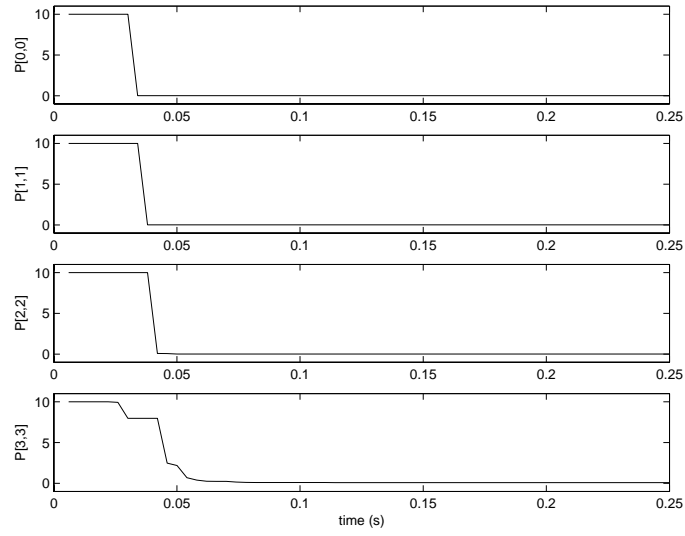


Figure 34: Covariance matrix when the process was excited with a load disturbance.

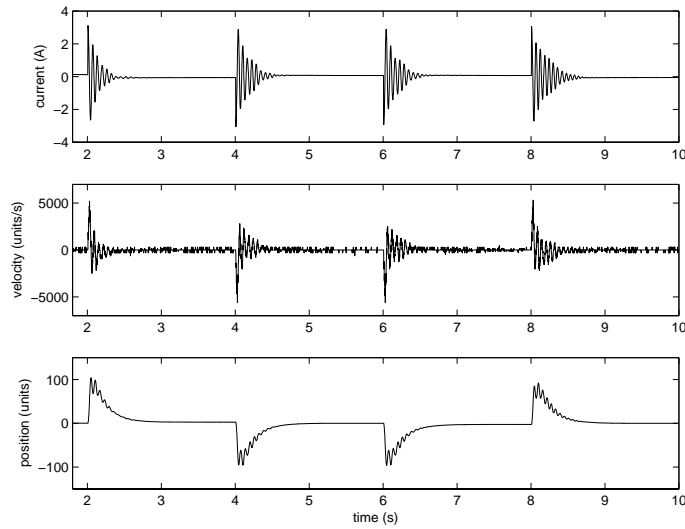


Figure 35: Current, velocity, and position error when the process was excited with a load disturbance.

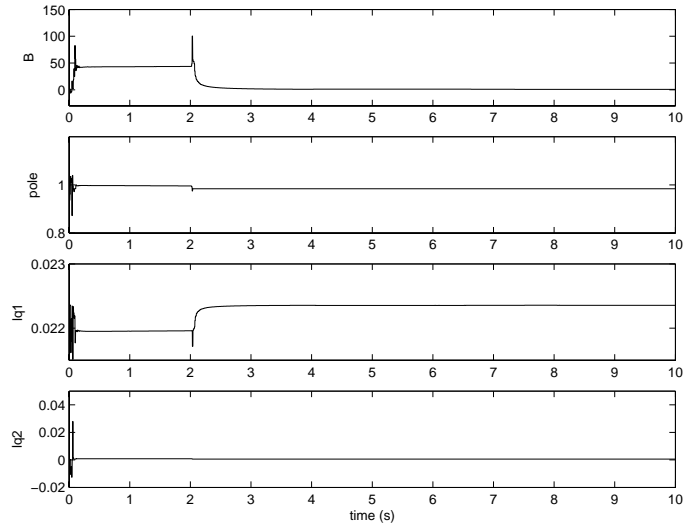


Figure 36: Reduced and resampled parameter estimations of the RLS identification, and controller parameters when the process was excited with a load disturbance.

4.5 Problems

A problem that occurred when the LQG controller was simulated was that the kalman filter did not manage to estimate the velocity when a load disturbance was acting on the process. No solution was found to this problem, why the efforts to control with an LQG controller were cancelled.

4.6 Summary

Four different controllers have been investigated in terms of their capability to reject load disturbances. An RST controller for velocity control has also been discussed, but since this controller was only used in the relay identification it will not be investigated further.

For the state feedback controller based on pole placement, it was very difficult to know where to place the poles to get the desired result. Therefore no more efforts will be made to improve this controller.

There were great expectations on the LQG controller since it has the ability to reduce measurement noise. However, due to the big load disturbance the LQG controller did not manage to estimate the velocity in a proper way. So the LQG controller was also abandoned.

Then the LQ controller and the LQ STR remained. The two controllers gave very similar, and good, results when being subjected to load distur-

bances. Therefore it was decided to go on working with both these controllers and investigate how they managed to follow a certain reference trajectory.

5 Reference following - The servo problem

5.1 Motivation

In control problems, the main interest is to keep the application as stable as possible when exposed to different disturbances. In this case the main interest is to make the application follow a reference signal as accurately and fast as possible. One of the main ways to approach this problem is to add a filter at the input of the closed loop system. The purpose of this is to clean the signal from unwanted frequencies and adapt the reference input to the rest of the system.

There is also another approach used in this project. This is to form the reference signal curve into a desirable shape. For example, a curve with rounded edges will give a smoother reference following than a curve with sharp edges. This is important since the curve with the sharp edges can perhaps give a fast response but will probably give rise to peak currents that will either be saturated or destroy the motor. Normally this is what is done by the prefilter, but if you know how large the reference step is and how fast it should be, the reference curve shaping can be a good alternative, or a complement.

As mentioned in section 1.2, the specification is to keep the error smaller than 13 units throughout the reference step. This specification implies that the reference signal must be easy to follow. This is why a carefully shaped reference curve is a good choice.

5.2 Reference following methods

5.2.1 Prefiltering

A prefilter can have different purposes. The usual one is to eliminate high frequencies to avoid aliasing. Aliasing means that frequencies above the Nyquist frequency will fold and become a disturbance. The Nyquist frequency is half the sampling frequency. In this project, however, the main objective of the prefilter is to act as a complement to the closed loop system. This includes eliminating high frequencies but also to transform the transfer function for the closed loop system. To achieve this you add and extract poles and zeros so the bode diagram of the closed loop system gets the desired shape. This usually means to ensure that the gain of the system is one as high up in the frequency band as possible [3].

5.2.2 Iterative learning control

Iterative learning control makes the process follow a specific reference signal as accurately as possible [8]. This is done by sending the reference signal to the input of the closed loop system and then reading the output from the closed loop system. The two signals are then compared and a signal u_k is added to the input signal so the output will get the same shape as the input signal (fig. 37). To get a good following of the reference signal the

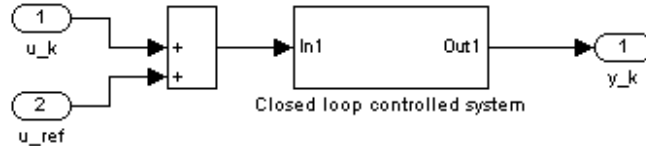


Figure 37: ILC schematics in theory.

procedure has to be repeated a couple of times so that u_k gets the optimal shape. The optimal shape is usually not acquired until after a couple of iterations. When it is acquired, the shape ($u_k + ref. signal$) is stored. It can then be used in the real application.

The downside with this controller is that it is not as flexible as the prefilter. Since u_k is optimized for a special shape of the reference signal you need to produce a unique u_k for every reference signal [8].

5.3 Reference following design

5.3.1 Prefiltering

To be able to use a fast reference signal and to have an exact and predictable response from the motor, it is important to have a good transfer function from the input to the output of the closed loop system. This means that any deviation from the amplification 1 in the bode diagram should be avoided since this will amplify or reduce the reference signal at that frequency. Looking at the pole zero map for the closed loop system, a judgement can be made of which poles and zeros that should be cancelled to achieve the wanted bode diagram.

After the cancellations a low pass filter on the form

$$F_{lowpass} = \frac{1}{s/40 + 1}$$

was implemented. This gave a bode curve that never became larger than 1 at any point. To move the slope up in the frequency band a lead filter can

be used. This filter increases the gain at high frequencies without changing the phase remarkably. The structure of a lead filter is

$$F_{lead} = \frac{s + b}{s + bN}$$

and, with the correct values, it can give the wanted shape of the curve. In this project, however, the lead filter did not change the curve noticeably. Because of the lack of quality results from prefiltering, the efforts were cancelled and the focus was aimed at iterative learning control instead.

5.3.2 Iterative learning control

The basic idea schematics of ILC is shown in figure 37. To the reference signal u_k is added which will give the desired output. The complete schematics used in this thesis is found in figure 38. u_k is updated in the following way

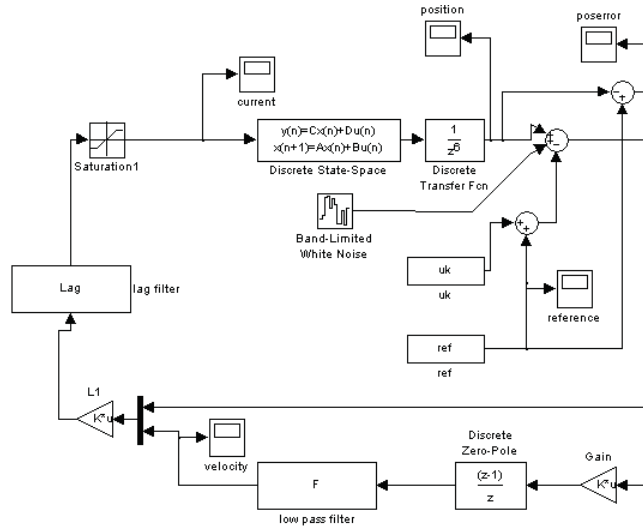


Figure 38: The ILC schematics used in this project.

$$\begin{aligned} u_k(t) &= Q(q)[u_{k-1}(t) + L(q)e_{k-1}(t)] \\ e_k(t) &= y_k(t) - u_{ref}(t) \\ y_k(t) &= G_{cl}(q)(u_{ref}(t) + u_k(t)) \end{aligned}$$

where $Q(q)$ and $L(q)$ are filters. $Q(q)$ is usually a lowpass filter with a cut-off frequency in the same area as the cut-off frequency of the process. $L(q)$ is a kind of compensation for the delay in the system. The idea is to eliminate

the error $e_k(t)$ in the output with the input signal u_k . This is why it is necessary for this method to iterate a couple of times. Otherwise u_k will not be able to foresee the error $e_k(t)$. When the iterations have led to a satisfying result, the vector u_k is stored. The optimal input to the closed loop system is now the sum of the reference curve and u_k .

This method is more or less dependent on the quality of the reference signal. The goal is usually to have as fast a reference signal as possible. The fastest way is just to use a step and try to get the output to follow it. In this case however, this was not possible because of the high currents that it would cause. Too high currents would damage the motor. To solve this problem, that is, to have low currents but still a fast reference signal, the input signal to the motor (not the closed loop system) was shaped so it would resemble a squarewave period. This would give a good result with low currents. It was achieved by sending a squarewave through the motor in the open loop system. A difficulty with this was to match the relative lengths between the positive and negative signals in the squarewave period. It is necessary for the output to be stationary. The reference signal looked for could then be recorded from the position output of the motor. The result is that the controller will try to shape the input to the motor in the closed loop system as a squarewave signal, so that the output from the motor will have the same shape as the input signal to the closed loop system, which is the reference signal.

What should be considered is that the output from the closed loop will not follow the input immediately. So the current will not have the squarewave shape until after a couple of iterations. So if the device is sensitive of high currents some kind of limitation, like a saturation, should be implemented. The motor in this project already has a saturation for safety reasons and because of the damage that high currents can cause.

There are two graphs that show the function of the ILC combined with the LQ controller from section 4.4.3. The first one, figure 39, illustrates many iterations done with a small step. Notice how the error becomes smaller after a couple of iterations and that the current resembles a squarewave. The second graph, figure 40, shows the error and current after several iterations. The error is not at any point more than 7 units. This is a good result since the specifications in section 1.2 hold that it must not be over 13 units. Also notice that the current clearly has the squarewave shape that was desired.

Graph 41 shows ILC combined with LQSTR. This combination did not work so well because the two methods both update regularly and ILC depends on the fact that the conditions are the same throughout the iteration process.

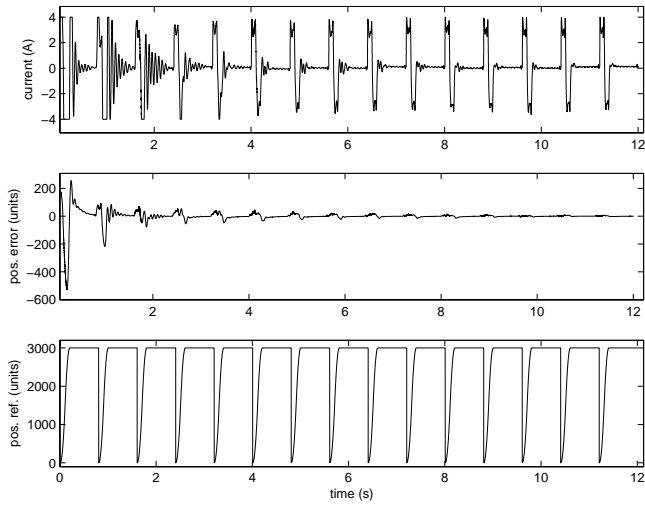


Figure 39: Several ILC iterations of a reference step of 3 000 units in 0.2 s. ILC is combined with an LQ controller. The top graph shows the current, the middle graph shows the error and the lower graph shows the reference signal.

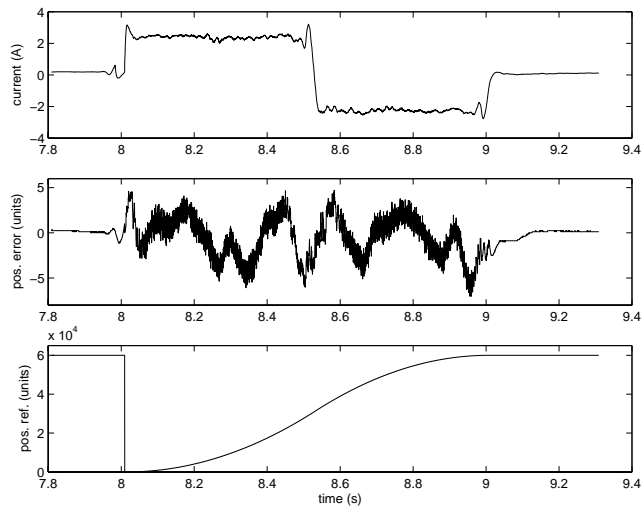


Figure 40: The 38th iteration after a step of 60 000 units (1/6 revolution) in 1 s. ILC is combined with an LQ controller. The top graph shows the current, the middle graph shows the error and the lower graph shows the reference signal.

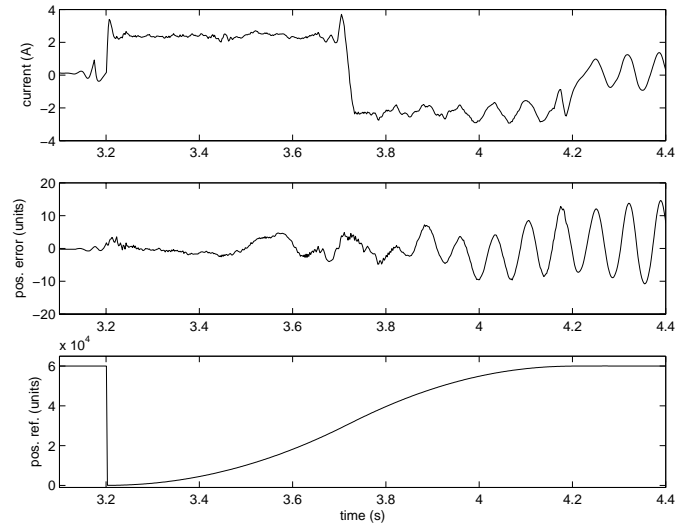


Figure 41: A step of 60 000 units (1/6 revolution) in 1 s. ILC is combined with an LQ STR. The top graph shows the current, the middle graph shows the error and the lower graph shows the reference signal.

5.4 Summary

Looking at the facts from the servo problem results, it is easy to draw the conclusion that ILC can improve the reference following considerably. Compare the two figures 42 and 43. The error is reduced from about 240 units ($\sim 1.8mm$) to 7 units ($\sim 0.05mm$) when doing a step of 60000 units (1/6 revolution). It must be remembered though that ILC optimizes one movement and that this means that different ILC optimizations have to be iterated and stored if you want a more flexible application.

The prefiltering did not achieve the same results as ILC but can, in another application, be the better method. The motor used in this project will, in its environment, have a uniform movement scheme which is why ILC will prove to be the better method to use in this application. But if the motor had to be more flexible, a prefilter would probably be a better choice.

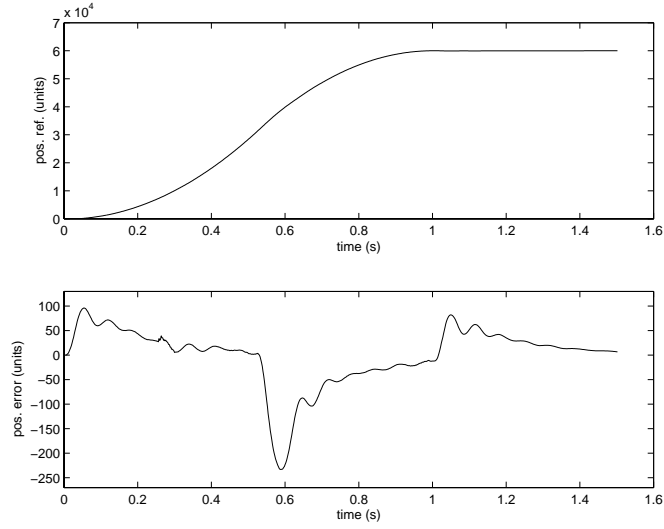


Figure 42: A step of 60 000 units using an LQ controller. The upper graph shows the reference step, the lower shows the position error.

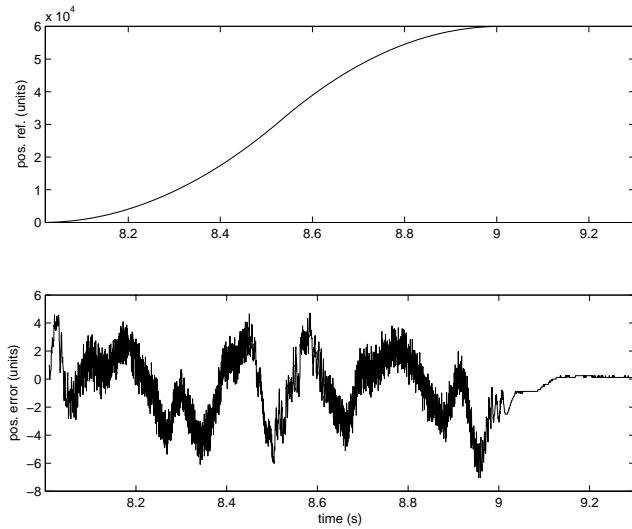


Figure 43: A step of 60 000 units using an LQ controller with an ILC optimized reference signal. The upper graph shows the reference step, the lower shows the position error.

6 Programming

6.1 General

All the identification methods and controllers described in the previous sections had to be rewritten to work in the computer. The programming language used, C, is a very common programming language in machine applications. Transforming the algorithms into C means that they have to be discretized. This is the same as to prepare them to work in a discrete environment where all tasks are performed only once every sampling period instead of continuously.

Every separate controller or identification method is implemented as a case. This structures the code and makes it simple to choose what method you want to use. You can choose a case by setting one of the flags `ident_mode` and `ctrl_mode`. The program is executed by setting one of the boolean variables `identon` or `ctrlon` to true.

Every case is divided into two major parts, Calculate output and Update states. In Calculate output, all the tasks that have to be done before the control signal is sent are done. Update states handles all the other tasks.

At the start of every Calculate output, there is an initiating sequence called `init_`. The purpose of this is to set certain parameters that are only to be set once every time a case is used, but need to be reset if you want to apply the case again without doing a warm startup (section 6.2).

The time delay also has to be taken into account. An example of this can be seen in the LS case (section B.1.2). The variables are stored the same number of cycles as the delay. This way the input can be compared with its coherent output.

6.2 The B&R Automation Studio environment

B&R Automation Studio is a PC program that provides an interface to the PLC from where the controllers and the identification are run. B&R Automation Studios supports several programming languages, such as Structured Text, Ladder and C. In this thesis, all code is written in C.

In B&R Automation Studios the program of the PLC connected to the computer is visualized as a tree structure, with the CPU on top. It is possible to add so called cyclic objects to the program. The cyclic objects are divided into two parts. The first part is an initialization part, which only runs once when the program is started. The second part is a cyclic part which runs as a loop with a specified sampling period.

Several cyclic objects can be added to the program. Then the object

with the highest sampling rate will get the highest priority. So called data objects can also be added to the program. The purpose of the data objects is simply to store data.

There is a watch window for each cyclic object where the current value of the variables used can be shown. Another option is to trace variables. Then the values of the variables chosen are stored in each loop. The trace program then displays the variables as a graph over time. This is how the plots of the real process are obtained in this project.

The trace can be performed in two different ways. Either you trace from the PLC, or you trace directly from the motor drive. But since the controllers are placed in the PLC it is natural to trace from the same place. When tracing from the motor drive the data will not match the process that is to be controlled, since there is a time delay.

To reset the program parameters, the measuring equipment and the motor, a warm or a cold restart is required. It is easily initiated from the program but is an undesirable, time-consuming process.

6.3 Identification methods

The identification is done with a sampling period of 4 ms. This is fast enough to perform the identification satisfyingly. The flag `ident_mode` chooses which identification technique to use. There are three different identification techniques implemented: relay identification, LS identification and RLS identification.

```
#define IDENT 1 (LS)
#define IDENTIFY 2 (Relay)
#define RLS 3
```

The first two are only done once but RLS identification runs at the same time as the controller, updating the process and controller parameters. The implementations can be found in appendix B.1.

6.4 Controllers

All controllers implemented in this project are placed in a cyclic object called `control_loop`. The sampling period of the control loop is 0.4 ms. What controller to be used is determined by the flag `ctrl_mode` through a switch statement. The possible choices are

```
#define STEP_SEQUENCE 1
#define RST_CTRL 2
```

```
#define STATE_FEEDBACK 3
#define LQ_CONTROL 4
#define ILC 5
#define LQG_CONTROL 6
```

The functionality of the different controllers is described in section B.2.

7 Testing and evaluating

7.1 Changing loads

7.1.1 Procedure

The object of these tests was to test the ability of the controllers to adapt to different process conditions. From the original eight weights attached to the motor, first four weights were removed and then all weights were removed.

Both the load disturbance rejection and the reference following have been tested. The load disturbance rejection was tested with the same load disturbance as in chapter 4 in all the cases. When the reference following was tested, a reference signal with a step of 60 000 units in 1 s was used (chapter 5).

ILC has also been tested for the different loads, but only combined with the LQ controller. When the LQ controller was used and the load was changed a new identification had to be performed before the LQ controller would work properly. The LQ STR on the other hand, was used without an initial identification of the process.

When the LQ controller was used and the load was changed, it was found that the tuning of the LQ parameters was no longer correct. The control signal was noisier and there were big overshoots in the position. This showed that the tuning of the weighting matrices in section 4.3.3 depended on the present load. To solve this problem the weighting of the input signal, R , was adapted to the new circumstances according to the following formula

$$R_{new\ load} = R_{8\ weights} \cdot \left(\frac{B_{new\ load}}{B_{8\ weights}} \right)^2$$

This conversion works since the input signal enters the process through the B matrix. When B is changing, R has to be changed as well to maintain the same relation. Q does not need to be changed since the pole does not move noticeably. However, this is a very process-dependent solution and should be used carefully. For example, if an adaptive controller like LQ STR is used, R should not be updated regularly. This is because an adaptive controller adapts the process model to load disturbances, which would result in big changes in R .

7.1.2 Four weights attached

The identification of this process resulted in a different process model. The new model from current to velocity became

$$H(z) = \frac{69.94}{z - 0.999928}$$

It can be compared to the original reduced order model from section 3.4.2.

$$H(z) = \frac{41.36}{z - 0.99995}$$

Figures 44 to 53 show the test results:

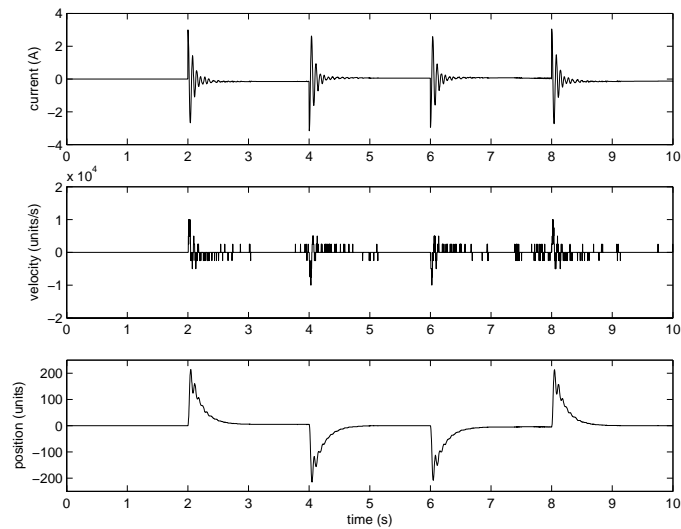


Figure 44: The load is decreased, only 4 weights are attached to the wheel. The process is excited with a load disturbance and the LQ controller is used. The upper graph shows the current, the middle graph shows the velocity and the lower one shows the position.

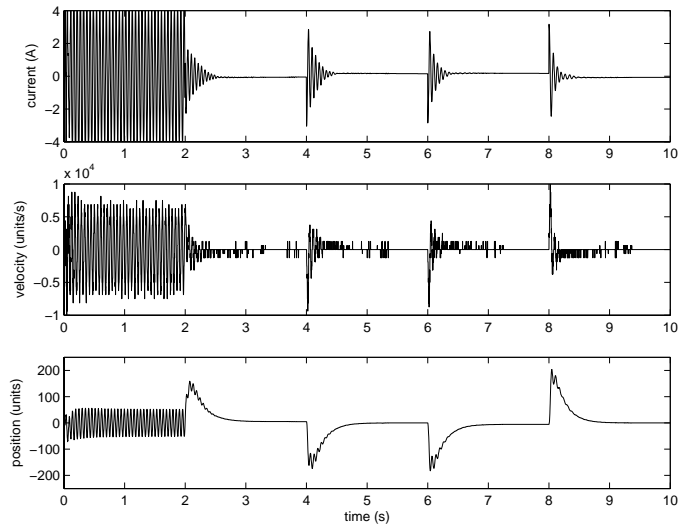


Figure 45: The load is decreased, only 4 weights are attached to the wheel. The process is excited with a load disturbance and the LQ STR is used. The upper graph shows the current, the middle graph shows the velocity and the lower one shows the position.

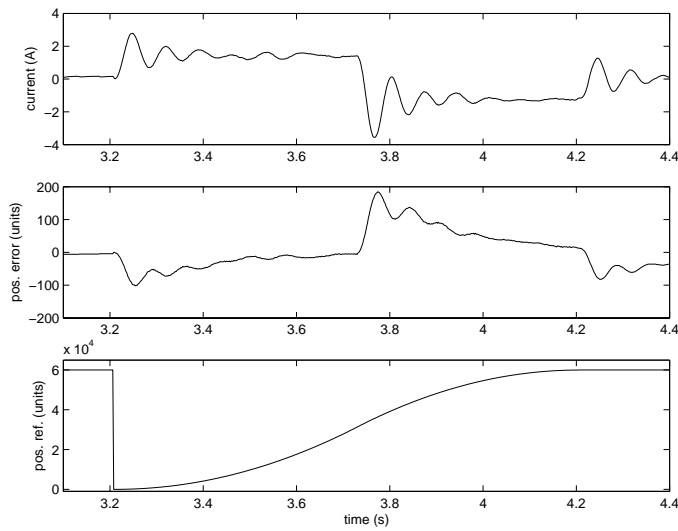


Figure 46: 4 weights are attached to the wheel. A step of 60 000 units is taken in 1 s with the LQ controller. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

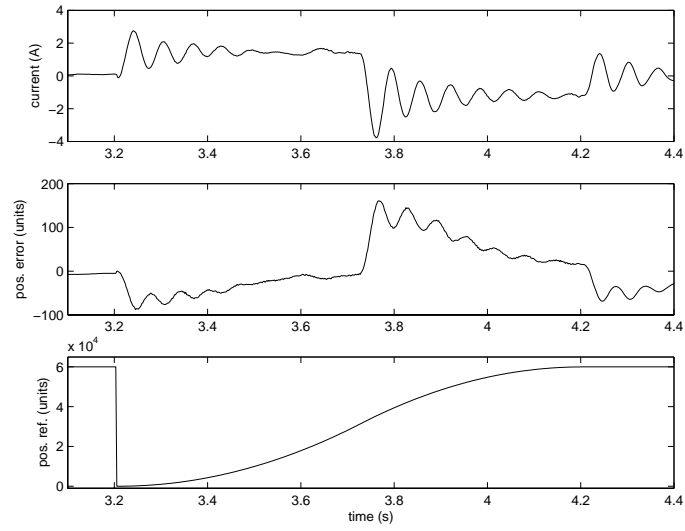


Figure 47: 4 weights are attached to the wheel. A step of 60 000 units is taken in 1 s with the LQ STR. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

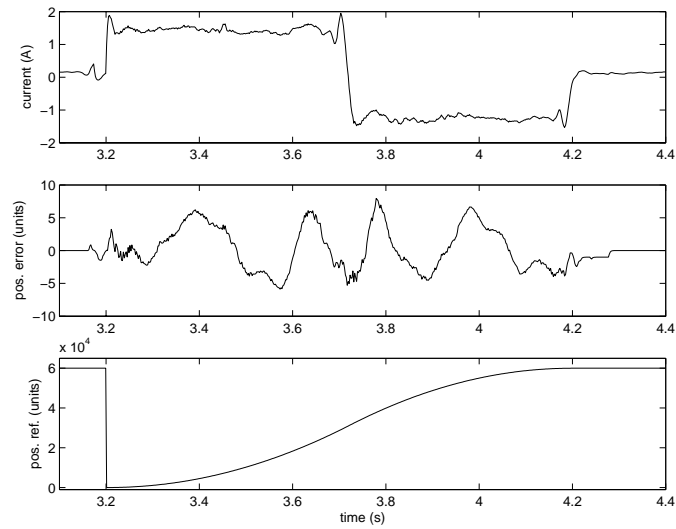


Figure 48: 4 weights are attached to the wheel. A step of 60 000 units is taken in 1 s with the LQ controller combined with ILC. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

7.1.3 No weights attached

The new model from current to velocity became

$$H(z) = \frac{296.12}{z - 0.999763}$$

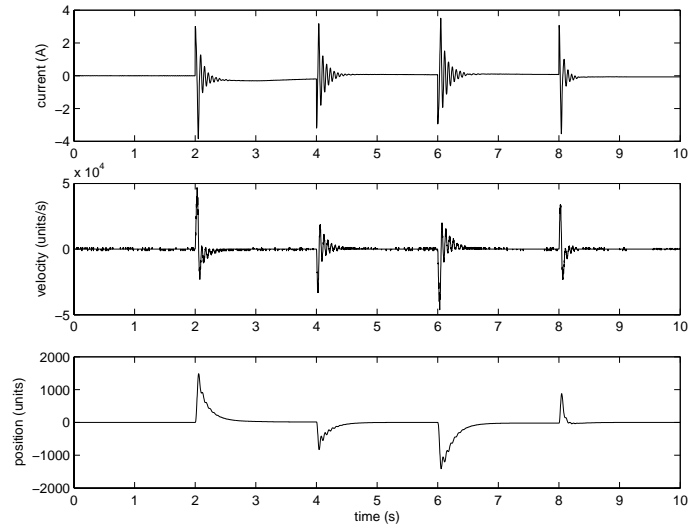


Figure 49: No weights are attached to the wheel. The process is excited with a load disturbance and the LQ controller is used. The upper graph shows the current, the middle graph shows the velocity and the lower one shows the position.

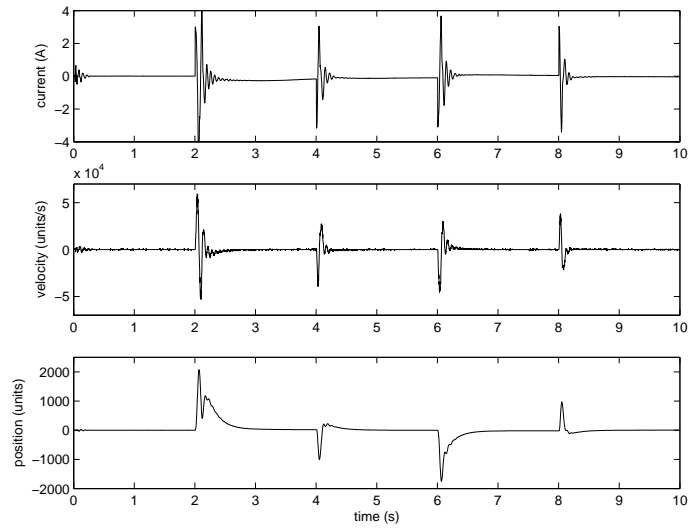


Figure 50: No weights are attached to the wheel. The process is excited with a load disturbance and the LQ STR is used. The upper graph shows the current, the middle graph shows the velocity and the lower one shows the position.

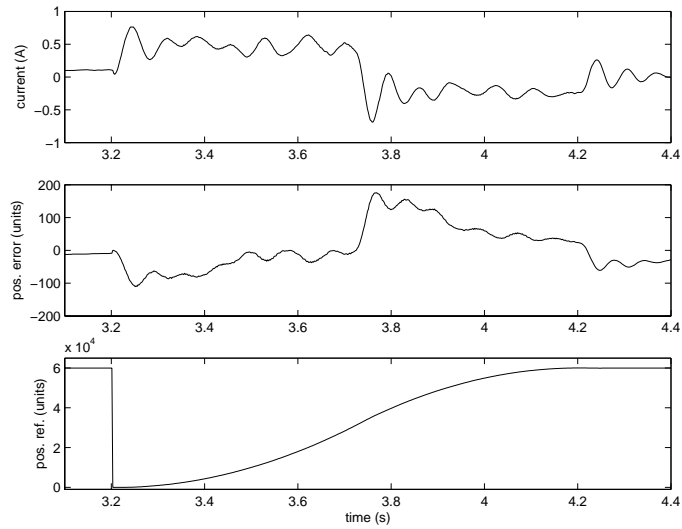


Figure 51: No weights are attached to the wheel. A step of 60 000 units is taken in 1 s with the LQ controller. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

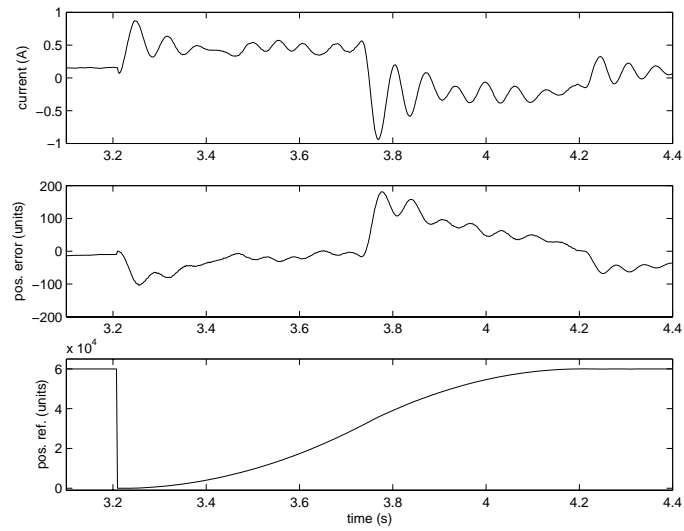


Figure 52: No weights are attached to the wheel. A step of 60 000 units is taken in 1 s with the LQ STR. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

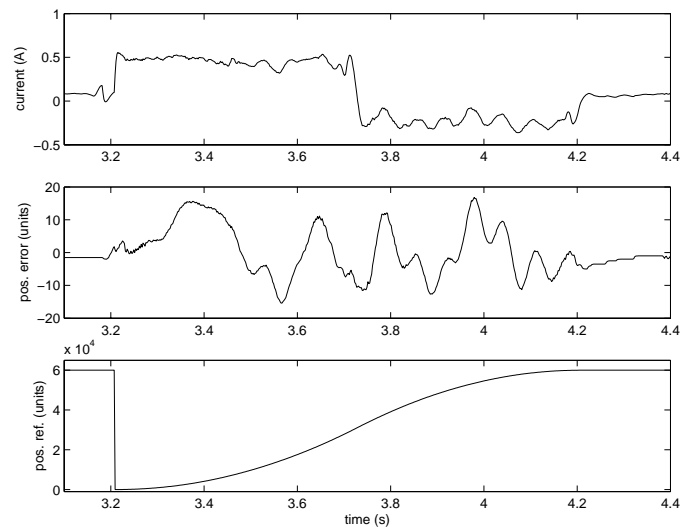


Figure 53: No weights are attached to the wheel. A step of 60 000 units is taken in 1 s with the LQ controller combined with ILC. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

7.2 Increasing the current

Tests were also performed with higher currents. The saturation limit was set to 8 A instead of 4 A. The benefits with this is that the reference step can be made faster and larger and the control application can control bigger load disturbances. It is also a way to show that even if the specifications were not reached to its full extent in the previous chapters, it can be reached with the controller in this project as long as it is supplied with enough power.

All these tests were performed with full load, that is, all weights attached to the wheel.

There were two tests with a reference step performed in the real process, one with as a large step as possible in 0.5 s (fig. 54), and the second with a three times larger step in 1 s (fig. 55).

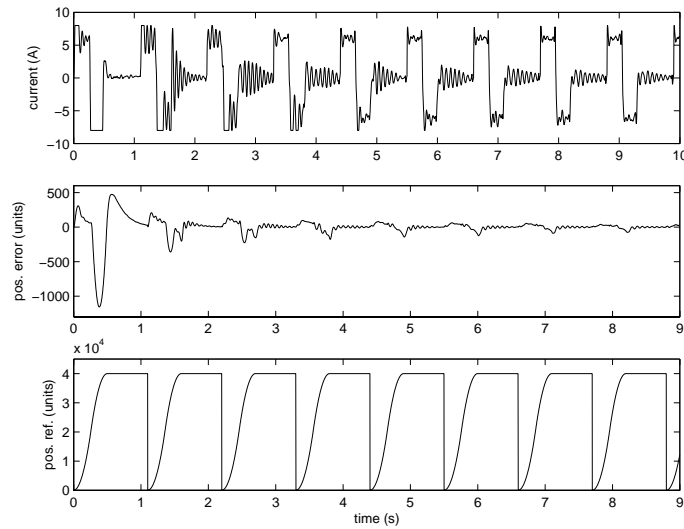


Figure 54: A step of a 1/9:th revolution in 0.5 s with the LQ controller combined with ILC. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

In section 2.1 it is mentioned that the highest peak current allowed in the motor is 17 A. Therefore, a test was performed of how fast the reference step of 1/6:th of a revolution could be made. Figure 56 shows a simulation of the fastest successful attempt with the reference step in 0.42 s.

In the load disturbance test, the LQ controller was excited with a load disturbance twice as big as in the previous tests, that is 6 A. The load disturbance graph in figure 57 shows that the stationary error of the position is doubled as the load disturbance is doubled. This was expected and is not

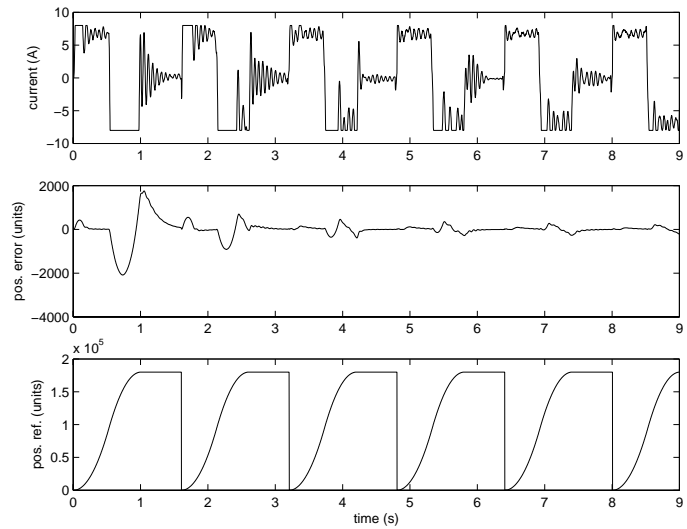


Figure 55: A step of a half revolution in 1 s with the LQ controller combined with ILC. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

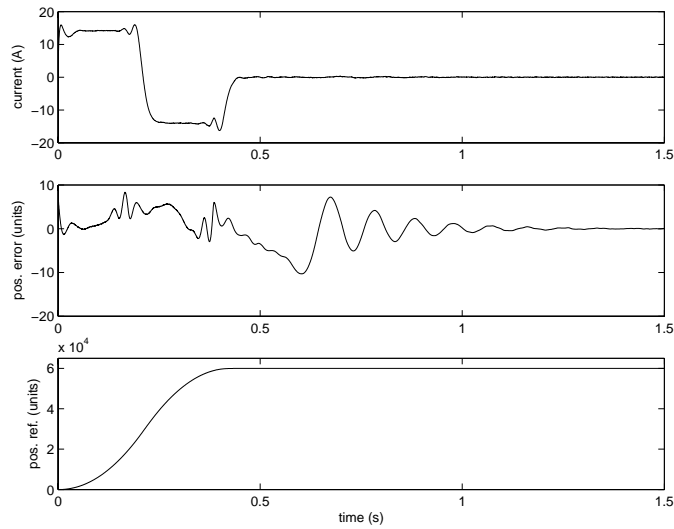


Figure 56: Simulation of a step of 1/6:th revolution (60 000 units) in 0.42 s with the LQ controller combined with ILC. The current limit is set to 17 A. The upper graph shows the current, the middle graph shows the position error and the lower one shows the position reference.

a big problem.

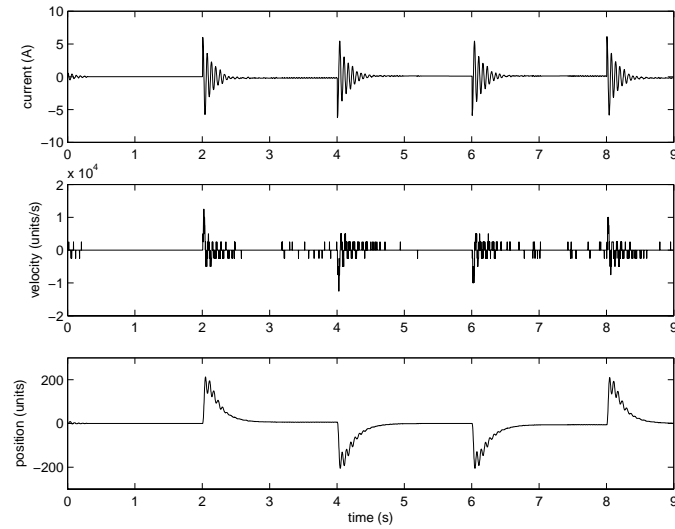


Figure 57: The process is excited with a load disturbance of 6 A. The upper graph shows the current, the middle graph shows the velocity and the lower one shows the position.

7.3 Test results

A big disappointment with the tests described in this chapter was that the LQ STR did not manage to adapt to changing loads. This is due to the fact that the weighting matrices in the controller have to be changed when the load is changed. But the weighting matrices cannot be updated in the LQ STR, because then the weightings would change when a load disturbance occurs and destroys the control performance.

Because of this problem the LQ STR is no advantage to the ordinary LQ controller. The fact that ILC, which is needed to keep the position error small enough, works best in combination with the LQ controller also makes the choice easier.

The attempts with different currents showed that it is possible to get a faster step. The limitation for this is the motor. The highest allowed peak current, according to the specifications sheet C, is 17 A. The simulations showed that using a current limit of 17 A instead of 4 A could reduce the step time from 1 s to 0.42 s.

The tests with higher currents also showed that the position error when ILC was performed was equally small for different currents. The position

error will stay small for fast reference trajectories as long as the current is not saturated.

The final result of these tests was that the LQ controller combined with ILC would be the best choice of the different controllers implemented in this project. It was also realized that it is not the controller that limits the time it takes to make a reference step, but the limits in power.

8 Conclusion

Several different identification methods and controllers have been investigated during the project. This chapter presents the identification method and controller that have the best performance and that will fulfill TetraPak's needs.

For the identification the batch version of the LS identification has been chosen. The LS identification is described in detail in sections 3.3.2 and 3.4.2. A third order model from current to velocity is estimated. The process is excited by a PRBS signal during the identification. The model is then reduced to a first order model by use of the method of dominating poles.

Since it is the position that is to be controlled, a second order state space model is created with x_1 as the position and x_2 as the velocity. This is described in section 4.2. The whole identification procedure is performed automatically, and takes about 30 seconds.

For the control of the process the LQ controller combined with ILC has been chosen. The LQ controller is described in sections 4.3.3 and 4.4.3, and ILC is described in sections 5.2.2 and 5.3.2.

The LQ controller manages to keep the stationary error within the limits of ± 0.1 mm when the process is affected by a load disturbance that corresponds to the nominal torque of the motor. Since a Ricatti solver is implemented, the LQ controller is ready to be used directly after the identification.

ILC is needed since the position error must not be bigger than ± 0.1 mm at any point during a reference step. For a reference step of 1/6:th revolution in 1 second ILC fulfills the specification. When using ILC the desired motion must be iterated several times. The procedure takes about 30 seconds. But once that is done the resulting signal can be stored and used over and over again. It is also possible to store signals for several different motions and then just switch between these when the process setup is changed.

We think that this is a good solution for TetraPak. The solution works very well when the same motion is performed several times, and when major changes in the process, like change of load, is known by the operator. The operator can then adjust the controller to cope with the new circumstances by a single click of a button. Other unknown process changes could be handled as load disturbances by the LQ controller.

9 Discussion

In this project, the most common identification methods and controllers have been used. The reason for this is that there has been no previous work done on this very process before. Concentrating on many controller solutions gives an idea of what controllers that might be preferable. But it does not give a more developed description of the specific controller of choice and how to benefit from it.

During this project, controllers that were considered to be less fitting were discarded so that efforts could be concentrated elsewhere. This has led to the fact that the LS model and the LQ controller combined with ILC have been more developed than for example the relay identification technique.

Before implementing the solutions given in this thesis, the following must be considered:

- If it is possible to measure the velocity directly, this should be done. The measuring disturbances of the motor are amplified by the differentiation of the position. If these could be reduced, it would be possible, for example, to reduce the effect of load disturbances by tuning the weighting matrices in the LQ controller differently (section 4.3.3).
- The time delay used in this project has been estimated to 6 sample periods. If this implementation of identification method and controller would be used, the time delay should be reinvestigated. The best thing to do would be to implement it all in the motor drive and in that way eliminate the time delay completely.
- No model of the measurement noise has been identified in this project. The noise has been considered to be white. If further work on this motor will be done, a suggestion would be to create a model of the noise, and then try to reduce the effects of it. This would probably lead to the fact that the LQ controller could be tuned to smaller errors.

There are a few things that should be considered especially when implementing the least squares identification, namely:

- The sampling time for the identification, 4 ms, could be optimized. The sampling time was changed from 1.2 ms to 4 ms because of synchronization disturbances that corrupted the process model. These disturbances were later eliminated. However, as opposed to the controller, the identification does not have to be as fast as possible to perform its best.

- The reduction algorithm used, the method of dominating poles, is not one of the most accepted technique. But considering the characteristics of the poles in the third order model, this technique is very satisfying. Combining this with the fact that the amount of C code required for dominating poles is a lot less than for the other techniques, it is realized that for this specific process, the method of dominating poles is a good choice.

When implementing the LQ controller for industrial use, there are also a couple of things that should be considered. First, the possibility of measuring the velocity mentioned above must be regarded. This would mean that the feedback could consist of two states instead of one real state and one differentiated one. Then, the lowpass filter on the velocity could perhaps be removed, which would increase the performance.

The other thing is the tuning of the weighting matrices (section 4.3.3). Whether or not the velocity could be measured, an improved identification of Q and R could result in a reduction of the effects of load disturbances. However, the investigations behind this thesis imply that the reduction of the effects would only be in the region of a few percent.

The L and Q filters used with ILC might perhaps also need an update if ILC would be used. If the time delay changes, the L used here will no longer be valid. Also if the process is changed, or a filter is replaced or removed, L would need to be updated. The same holds for Q . If the cut-off frequency for the closed loop system is changed, Q should be updated according to section 5.2.2.

If further work will be done on this motor, this thesis will hopefully be consulted. The choices of identification technique and controller made here are only to be considered a recommendation as to what solution that should be implemented. However, if another technique should be used, the information in this thesis can still be of help.

References

- [1] Tore Hägglund
Reglerteknik AK, Föreläsningar, Department of Automatic Control, Lund University, Lund, Sweden, 2000.
- [2] Karl J. Åström, Tore Hägglund
Automatic Tuning of PID Controllers, Instrument Society of America, 1988.
- [3] Karl J. Åström, Björn Wittenmark
Computer Controlled Systems, third edition. Prentice Hall Inc., New Jersey, 1997.
- [4] Rolf Johansson
System Modeling and Identification. Prentice Hall Inc., Engelwood Cliffs, New Jersey, 1993.
- [5] Jean-Jaques E. Slotine, Weiping Li
Applied Nonlinear Control. Prentice Hall Inc., Upper Saddle River, New Jersey, 1991.
- [6] Karl J. Åström, Björn Wittenmark
Adaptive Control, second edition. Addison Wesley Publication Company, 1995.
- [7] Pontus Nordfeldt
Regulator Design for a Flexible Servo, Master Thesis, ISSN 0280-5316, Department of Automatic Control, Lund University, Lund, Sweden, 2003.
- [8] H. Hjalmarsson, S. Gunnarsson, M. Gevers, O. Lequin
Iterative Feedback Tuning: Theory and Applications, ISSN 0272-1708, IEEE Control Systems, pp. 26-41, 1998.

A Matlab code

A.1 Relay

```
G = tf([32000],[1/0.0309 1]);
G.outputdelay=0.04;
H = c2d(G,0.0012,'tustin');

sys=ss(H);
A=sys.a;
B=sys.b;
C=sys.c;
D=sys.d;
sys=idss(sys);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Data series for validation %%%%%%%%%%
dirtrace2
u50_3=matrix2(:,4);
y50_3=matrix2(:,2);
zv_3=[y50_3 u50_3];
uv_3 = zv_3(:,2);
yv_3 = zv_3(:,1);

% Estimating initial values
[ye3,x03,B3,D3] = dac2bdx(A,C,uv_3,yv_3);
B=0;
D=0;
ye3=0;

% Cross validation
ye3=idsim(uv_3,sys,x03);

figure(3)
plot(time50_3,[ye3,zv_3(:,1)])
xlabel('time (s)')
ylabel('velocity (units/s)')
```

A.2 ARX-model

```
data50
u50=matrix50(:,2);
y50=matrix50(:,4);
y50=y50/1000;
time50=matrix50(:,1);
zi=[y50 u50];
th=arx(zi,[3 1 1],[],0.0012);

% Creating a third order model
[den num]=th2poly(th);
```

```

[A,B,C,D]=tf2ss(num,den);
sys=ss(A,B,C,D,0.0012);
sys=idss(sys);

% Creating a reduced order model
[Ab,Bb,Cb,M,T] = dbalreal(A,B,C);
Db = 0;
[Ared,Bred,Cred,Dred] = dmodred(Ab,Bb,Cb,Db,2:3);
Dred = 0;
sys2=ss(Ared,Bred,Cred,Dred,0.0012);
sys2=idss(sys2);
threduced = idpoly(sys2);

figure(1)
bodeplot(th,'red')
figure(9)
bodeplot(threduced,'blue')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Data series for validation %%%%%%%%%%%%%%%
data50_3
u50_3=matrix50_3(:,2);
y50_3=matrix50_3(:,4);
y50_3=y50_3/1000;
time50_3=matrix50_3(:,1);
zv_3=[y50_3 u50_3];

th2=arx(zv_3,[3 1 1],[],0.0012);
NN = [1 1 1;2 1 1;2 2 1;3 1 1;3 2 1;3 3 1;4 3 1;4 4 1;5 4 1];
V = arxstruc(zi,zv_3,NN);
selstruc(V);
uv_3 = zv_3(:,2);
yv_3 = zv_3(:,1);

% Estimating initial values
[ye3,x03,B3,D3] = dac2bdx(A,C,uv_3,yv_3);
B=0;
D=0;
ye3=0;

% Cross validation
ye3=idsim(uv_3,sys,x03);

% Estimating initial values for the reduced model
[ye3red,x03red,B3red,D3red] = dac2bdx(Ared,Cred,uv_3,yv_3);
B3red=0;
D3red=0;
ye3red=0;

% Cross validation with the reduced model

```

```

ye3red=idsim(uv_3,sys2,x03red);

figure(13)
subplot(2,1,1); plot(time50_3,[ye3,zv_3(:,1)])
subplot(2,1,2); plot(time50_3,[ye3red,zv_3(:,1)])

```

A.3 Subspace identification

```

data50
u50=matrix50(:,2);
y50=matrix50(:,4);
y50=y50/1000;
time50=matrix50(:,1);
zi=[y50 u50];
G = n4sid([y50 u50],3,'Focus','Stability');
[num,den] = tfdata(G,0.0012);
[A,B,C,D]=tf2ss(num,den);
sys = ss(A,B,C,D,0.0012);
sys = idss(sys);

% Creating a reduced order model
[Ab,Bb,Cb,M,T] = dbalreal(A,B,C);
Db = 0;
[Ared,Bred,Cred,Dred] = dmodred(Ab,Bb,Cb,Db,2:3);
Dred = 0;
sys2=ss(Ared,Bred,Cred,Dred,0.0012);
sys2=idss(sys2);
threduced = idpoly(sys2);

figure(1)
bodeplot(sys,'red')
figure(9)
bodeplot(threduced,'blue')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Data series for validation %%%%%%%%%%
data50_3
u50_3=matrix50_3(:,2);
y50_3=matrix50_3(:,4);
y50_3=y50_3/1000;
time50_3=matrix50_3(:,1);
zv_3=[y50_3 u50_3];
uv_3 = zv_3(:,2);
yv_3 = zv_3(:,1);

% Estimating initial values
[ye3,x03,B3,D3] = dac2bdx(A,C,uv_3,yv_3);
B=0;
D=0;
ye3=0;

```

```

% Cross validation
ye3=idsim(uv_3,sys,x03);

% Estimating initial values for the reduced model
[ye3red,x03red,B3red,D3red] = dac2bdx(Ared,Cred,uv_3,yv_3);
B3red=0;
D3red=0;
ye3red=0;

% Cross validation with the reduced model
ye3red=idsim(uv_3,sys2,x03red);

figure(3)
subplot(2,1,1); plot(time50_3,[ye3,zv_3(:,1)])
subplot(2,1,2); plot(time50_3,[ye3red,zv_3(:,1)])

```

A.4 Pole placement for the RST controller

```

pol = 0.999878;
gfB = 372.165;
h = 0.004;
s = tf('s');
G = tf(1/(1+s/3));
H = tf(c2d(G,h))
[num den] = tfdata(H,h)
p1 = -den(2)
z = tf('z');
[R,S,T] = rstfd(1,gfB,[1 -pol],1,[1 -p1],1)

```

A.5 LQ control

```

gfB = 38.8082;
pole = 0.999969;
h = 0.0004;

G = tf(gfB,[1 -pole],h);
sysc=d2c(G,'matched');
Gc = d2c(G);
Ad = [1 h*pole ; 0 pole];
Bd = [gfB*h ; gfB];
Cd = [1 0];
Dd = 0;
sys=ss(Ad,Bd,Cd,Dd,h);

Qd = [50000 0;0 5];
Rd = 100000000;
[Ld,Sd,Ed] = dlqr(Ad,Bd,Qd,Rd,0);

```

```

s = tf('s');
D = tf([1 -1],[h 0],h);
F_delay = tf([1],[1 0 0 0 0 0],h);
sys = sys*F_delay;
F=1/(s/200+1)^2;
F=c2d(F,h);

sys2 = (Ld(1)+Ld(2)*F*D);
figure(10)
bode(sys*sys2/(1+sys*sys2))
figure(1)
margin((sys2*sys));

[Gm,Pm,Wcg,Wcp]=margin((sys2*sys));
a = 0.1*Wcp;
M = 50;
Fret = tf([1 a],[1 a/M]);
Fretd = c2d(Fret,h);
Fretdss = ss(Fretd);
sys2 = Fretd*sys2;

figure(2)
bode((sys*sys2)/(1+sys*sys2))
grid on
figure(11)
margin(sys*sys2)

```

A.6 LQG control

```

gfB = 38.8082;
pole = 0.999969;
h = 0.0004;

G = tf(gfB,[1 -pole],h);
Gc = d2c(G);
Ad = [1 h*pole ; 0 pole];
Bd = [gfB*h ; gfB];
Cd = [1 0];
Dd = 0;
sys=ss(Ad,Bd,Cd,Dd,h);

Qd = [50000 0;0 5];
Rd = 100000000;
[Ld,Sd,Ed] = dlqr(Ad,Bd,Qd,Rd,0);

Qn = 0.001;
Rn = 0.002;
Nn = 0;

```

```

[KEST,K,P] = kalman(sys,Qn,Rn,Nn);
kest = ss(Ad-K*Cd,[Bd K],eye(2),zeros(2,2),h);

s = tf('s');
D = tf([1 -1],[h 0],h);
AA = Ad-Bd*Ld-K*Cd;
sys2 = ss(AA,K,-Ld,0,h);

figure(6)
[Gm,Pm,Wcg,Wcp]=margin((-sys2*sys));
a = 0.1*Wcp;
M = 50;
Fret = tf([1 a],[1 a/M])
Fretd = c2d(Fret,h);
Fretdss = ss(Fretd)

```

A.7 Prefilter calculations

```

Flowpass = tf(1,[1/40 1])
Flowpass = c2d(Flowpass,h);
Flowpass = ss(Flowpass);

Flead = tf([1/a 1],[1/b 1]);
Flead = c2d(Flead,h);
Flead = ss(Flead);

Gc = sys*sys2/(1+sys*sys2);

figure(1)
bode(Gc)
grid on
figure(3)
pzmap(Gc)

ff = 1/(s/b+1);
ffs = c2d(ff,h)
ffs=ffs*ffs;

[a,b]=tfdata(ffs,'s')
qpart = 0.99383680248198*0.99383680248198
+ 0.03380477264556*0.03380477264556;
ppart = -sqrt(qpart)*2*cos(atan(0.03380477264556/0.99383680248198));
notch = tf(a(3)*[1 ppart qpart],b,h);
notch = (1/freqresp(notch,0))*notch;
notch = ss(notch);

PrefF = Flowpass*Flead*notch;
figure(5)
pzmap(PrefF*Gc)

```



```

figure(15)
bode(PrefF*Gc)
grid on

```

A.8 Curve generating algorithm

```

load 'C:\Carl01a_exjobb\ref_lowpass.mat' ref_lowpass;
reft = ref_lowpass;
h=0.0004;

ampfactor = 45000/3654;
timefactor = 1.5;
simtime = timefactor;

reft21 = (0.2004:h:1)';
reft22 = ones(1,2000)'*reft(500,2);
reft2 = [reft ; reft21 reft22];

reft2 = [reft2(:,1)*timefactor reft2(:,2)*ampfactor];
s = tf('s');
lowp = tf([1],[1/200 1]);
lowp = c2d(lowp,h);
[num,den] = tfdata(lowp,h);
reft2 = filter(num,den,reft2);

```

A.9 ILC filter calculations

```

simtime = 1.2;
reft_2;
sinefreq = 3;
nl=42;
Tf=131;
s=tf('s');
h=0.0004;
z=tf('z',h);
slutfelet=300;
nli=40:1:44;
Tfi=logspace(1,2,5);
%for ii=1:length(nli)
    %nl=nli(ii);
    nl=55;
    % for kk=1:length(Tfi)
        %Tf=Tfi(kk);
        Tf=300;
%Tf=1000;

Q=1/(s/Tf+1)^4;
Q=c2d(Q,h);
L=z^nl*0.1;

```

```

[aL,bL]=tfdata(L,'z');
[aQ,bQ]=tfdata(Q,'z');

n = 0;
uk=0;
time = 0;
for i=1:(simtime/h+2)
    uk(i) = 0;
    time(i) = n;
    n=n+h;
end

uk = [time', uk'];
clear e
for i=1:20
    rn=round(100*rand(1));

    sim('newLQmodeldisbackup')
    felet=max(abs(e(:,3)));
    uk = [uk(:,1) uk(:,3)];
    figure(1)
    plot(e)
    grid on
    title('e')
    figure(2)
    plot(uk,'+')
    grid on
    title('uk')

    nn=length(e(:,3));
    e(1:nn-nl+1,3)=e(nl:nn,3);
    e(nn-nl+1:nn,3)=e(nn-nl,3)*ones(nl,1);

    er=e(:,3);
    uk(:,2) = uk(:,2)+0.2*er;
end

felet;
if felet<slutfelet
    slutfelet=felet
    finalTf=Tf;
    finalnl=nl
end

```

B Programming

B.1 Identification methods

B.1.1 Relay identification

The relay technique is written so the relay identification is directly followed by the step sequence. The relay part is active as long as the period length is trying to find its equilibrium. In the script this is as long as $(\text{oldomega}-\text{omega})>0.005$). When this is no longer true the boolean variable `done` changes to true so the step sequence can be initiated. The relay hysteresis is called `eps` and when the `position` variable crosses it, the variable `relaysign` changes the direction of the current. The RST controller is written as $\text{current} = (\text{vref} \cdot T - S \cdot y) / R$ and is used to control the step sequence. The simplicity of the controller is due to the low order of the model (section 4.3.1). 24 005 cycles after the initiation of the step sequence, the identification procedure is done.

```
/****** Calculate output *****/
case IDENTIFY:
if(init_identify == true)
{
    i = 0;
    oldcurrent = 0;
    zeroposition=position;
    omega = 2;
    oldomega = 3;
    init_identify = false;
}
if((oldomega-omega)>0.005 && (done == false))
{
    if(position<(zeroposition-eps) && relaysign==-1)
        relaysign=1;
    if(position>(zeroposition+eps) && relaysign==1)
        relaysign=-1;
    current = relaysign*d;
}
else
{
    done = true;
    current = (vref*T-S*y)/R;
    if (i>24005)
    {
        current = 0;
        identify_done=true;
    }
}
}
```

```
break;
```

The amplitude of the relay oscillations is described in the variable `deviation`. This value is stored in `maxdev` every time it is larger than its previous value. The `omega` used in the Calculate output is calculated from `period` which is given by `stepnbr*h`. `stepnbr` is the number of cycles from the previous sign shift of deviation from negative to positive and `h` is the sampling time.

```
/****** Update states *****/
case IDENTIFY:

/****** Relay *****/
if(done == false)
{
    deviation = position-zeroposition;
    if(deviation<0)
        deviation = -1*deviation;
    if(deviation>maxdev)
        maxdev=deviation;
    stepnbr = stepnbr + 1;
    if((oldposition-zeroposition)*(position-zeroposition)<0
    && (position-zeroposition)>0)
    {
        period = stepnbr*h;
        stepnbr = 0;
        if(period>maxperiod){
            maxperiod = period;
            oldomega = omega;
            omega = 2*Pi/maxperiod;
        }
    }
}

/****** Step response *****/
else
{
    i++;
    if (i>12000)
        vref = 0.5*speed;
    if(i>=8000 && i<=10000)
        temp1 = temp1 + current;
    if(i>=22000 && i<=24000)
        temp2 = temp2 + current;
}
oldcurrent = current;

/****** Calculations *****/
if(identify_done)
```

```

{
  K = vref/(temp1/(10000-8000+1)-temp2/(24000-22000+1));
  q = sqrt((-Pi/(4*d)*sqrt(maxdev*maxdev-eps*eps))*(-Pi/(4*d)
*sqrt(maxdev*maxdev-eps*eps))+((Pi*eps)/(4*d))*((Pi*eps)/(4*d)));
  phase = Pi+atan(((Pi*eps)/(4*d))/((4*d)*sqrt(maxdev*maxdev-eps*eps)));
  if(phase<0)
  phase=phase+2*Pi;
  T1 = omega/sqrt((K/(q*omega))*(K/(q*omega))-1);
  L=(phase-(Pi+atan(T/omega)))/omega;
  identify_done = false;
  done = false;
  ctrlon = false;
}
break;

```

In the calculations part, the model is obtained with formulas that can be found in [5].

B.1.2 Least squares identification

The least squares identification has been divided into three parts. The Calculate output part and Update states part are the same as for the rest, whereas the third part, final calculations, takes place after the system has been excited. This is because it needs the gathered sums from the Update state part to do its calculations.

The first part, `if(init_ident == true)`, is just for setting the variable to zero at the beginning of the identification procedure. The excitation signal, a PRBS signal, is set at the end of this part.

```

/***** Calculate output *****/
case IDENT:
if(init_ident == true)
{
  y1y1 = 0;
  y1y2 = 0;
  y1y3 = 0;
  y1u7 = 0;
  y2y2 = 0;
  y2y3 = 0;
  y2u7 = 0;
  y3y3 = 0;
  y3u7 = 0;
  u7u7 = 0;
  y1y = 0;
  y2y = 0;
  y3y = 0;
  yu7 = 0;
  yone=0;

```

```

    ytwo=0;
    ythree=0;
    uone=0;
    ident4ms = false;
    i = 0;
    zeroposition=position;
    oldposition=zeroposition;
    init_ident = false;
}
current = 2*prbs_ampl*(prbs_reg[15] - 0.5);
break;

```

In the Update states section, the PRBS signal is updated. After that, sums of the different multiplications used in the least squares are gathered. Finally, the two boolean variables, `ident4ms` and `alldone`, are set to true. `Ident4ms` tells the final calculations part that the identification has been performed with a sampling period of 4 ms. This is because the controllers work with a sampling period of 0.4 ms so the calculated model has to be resampled to work with the controllers.

`Alldone` declares that the excitation part is over and all that remains is to calculate the model.

```

/***** Update states *****/
case IDENT:
prbs_c = prbs_c + 1;
if (prbs_c >= prbs_period)
{
    prbs_c = 0;
    tmpi = prbs_reg[15] + prbs_reg[14] + prbs_reg[12] + prbs_reg[3];
    for(lp1 = 14; lp1 >= 0; lp1--)
        prbs_reg[lp1 + 1] = prbs_reg[lp1];
    prbs_reg[0] = tmpi % 2;
}
ydiv = y/100;
i++;
if((i>200) && (i<=2500))
{
    y3y3 = y3y3 + ythree*ythree;
    y2y2 = y2y2 + ytwo*ytwo;
    y2y3 = y2y3 + ytwo*ythree;
    y1y1 = y1y1 + yone*yone;
    y1y2 = y1y2 + yone*ytwo;
    y1y3 = y1y3 + yone*ythree;
    y1u7 = y1u7 + yone*uone;
    y2u7 = y2u7 + ytwo*uone;
    y3u7 = y3u7 + ythree*uone;
    u7u7 = u7u7 + uone*uone;
    y1y = y1y + yone*ydiv;
}

```

```

    y2y = y2y + ytwo*ydiv;
    y3y = y3y + ythree*ydiv;
    yu7 = yu7 + uone*ydiv;
}
if(i==2701)
{
    ident4ms = true;
    alldone = true;
}
ythree=ytwo;
ytwo=yone;
yone=ydiv;
uone=current;
break;

```

When `alldone` is set to true, this part is initiated. The variables `A1`, `A2`, `A3` and `B` are the model parameters in the third order model. The next step is to reduce the model order. This is done by iterations to find the dominating pole (section 3.3.6). Dividing the denominator of the third order model with its derivative will give a negative or a positive value depending on its relationship to the pole. Using this, you can find the pole by iterating. The code lines after the if-algorithm are necessary to keep the stationary gain of the process after the reduction.

The next part is for putting the model on state space form. Some of the controllers need it to be on this form.

```

/***** Final calculations *****/
if(alldone){
    j++;
    if(j==1)
        A1 = -(y1y*y2y2*y3y3*u7u7-y1y*y2y2*y3u7*y3u7-y1y*y2y3*y2y3*u7u7+2*y1y*y2y3
*y2u7*y3u7-y1y*y2u7*y2u7*y3y3-y2y*y1y2*y3y3*u7u7+y2y*y1y2*y3u7*y3u7+y2y*y2y3
*y1y3*u7u7-y2y*y2y3*y1u7*y3u7-y2y*y2u7*y1y3*y3u7+y2y*y2u7*y1u7*y3y3+y3y*y1y2
*y2y3*u7u7-y3y*y1y2*y2u7*y3u7-y3y*y2y2*y1y3*u7u7+y3y*y2y2*y1u7*y3u7+y3y*y1y3
*y2u7*y2u7-y3y*y2u7*y1u7*y2y3-yu7*y1y2*y2y3*y3u7+yu7*y1y2*y2u7*y3y3+yu7*y2y2
*y1y3*y3u7-yu7*y2y2*y1u7*y3y3-yu7*y2y3*y1y3*y2u7+yu7*y1u7*y2y3*y2y3)
/(y1y1*y2y2*y3y3*u7u7-y1y1*y2y2*y3u7*y3u7-y1y1*y2y3*y2y3*u7u7+2*y1y1*y2y3
*y2u7*y3u7-y1y1*y2u7*y2u7*y3y3-y1y2*y1y2*y3y3*u7u7+y1y2*y1y2*y3u7*y3u7+2
*y1y2*y2y3*y1y3*u7u7-2*y1y2*y2y3*y1u7*y3u7-2*y1y2*y2u7*y1y3*y3u7+2*y1y2*y2u7
*y1u7*y3y3-y2y2*y1y3*y1y3*u7u7+2*y1y3*y2y2*y1u7*y3u7+y1y3*y1y3*y2u7*y2u7-2
*y1y3*y1u7*y2u7*y2y3-y1u7*y1u7*y2y2*y3y3+y1u7*y1u7*y2y3*y2y3);
    else if(j==3)
        A2 = (y1y*y1y2*y3y3*u7u7-y1y*y1y2*y3u7*y3u7-y1y*y2y3*y1y3*u7u7+y1y*y2y3
*y1u7*y3u7+y1y*y2u7*y1y3*y3u7-y1y*y2u7*y1u7*y3y3-y2y*y1y1*y3y3*u7u7+y2y*y1y1
*y3u7*y3u7+y2y*y1y3*y1y3*u7u7-2*y2y*y1y3*y1u7*y3u7+y2y*y1u7*y1u7*y3y3+y3y
*y1y1*y2y3*u7u7-y3y*y1y1*y2u7*y3u7-y3y*y1y3*y1y2*u7u7+y3y*y1y3*y1u7*y2u7+y3y
*y1u7*y1y2*y3u7-y3y*y1u7*y1u7*y2y3-yu7*y1y1*y2y3*y3u7+yu7*y1y1*y2u7*y3y3+yu7
*y1y3*y1y2*y3u7-yu7*y1y3*y1y3*y2u7-yu7*y1u7*y1y2*y3y3+yu7*y1u7*y1y3*y2y3)

```

```

/(y1y1*y2y2*y3y3*u7u7-y1y1*y2y2*y3u7*y3u7-y1y1*y2y3*y2y3*u7u7+2*y1y1*y2y3
*y2u7*y3u7-y1y1*y2u7*y2u7*y3y3-y1y2*y1y2*y3y3*u7u7+y1y2*y1y2*y3u7*y3u7+2
*y1y2*y2y3*y1y3*u7u7-2*y1y2*y2y3*y1u7*y3u7-2*y1y2*y2u7*y1y3*y3u7+2*y1y2*y2u7
*y1u7*y3y3-y2y2*y1y3*y1y3*u7u7+2*y1y3*y2y2*y1u7*y3u7+y1y3*y1y3*y2u7*y2u7-2
*y1y3*y1u7*y2u7*y2y3-y1u7*y1u7*y2y2*y3y3+y1u7*y1u7*y2y3*y2y3);
else if(j==5)
A3 =-(y1y*y1y2*y2y3*u7u7-y1y*y1y2*y2u7*y3u7-y1y*y2y2*y1y3*u7u7+y1y*y2y2
*y1u7*y3u7+y1y*y1y3*y2u7*y2u7-y1y*y2u7*y1u7*y2y3-y2y*y1y1*y2y3*u7u7+y2y*y1y1
*y2u7*y3u7+y2y*y1y3*y1y2*u7u7-y2y*y1y3*y1u7*y2u7-y2y*y1u7*y1y2*y3u7+y2y*y1u7
*y1u7*y2y3+y3y*y1y1*y2y2*u7u7-y3y*y1y1*y2u7*y2u7-y3y*y1y2*y1y2*u7u7+2*y3y
*y1y2*y1u7*y2u7-y3y*y1u7*y1u7*y2y2-yu7*y1y1*y2y2*y3u7+yu7*y1y1*y2u7*y2y3+yu7
*y1y2*y1y2*y3u7-yu7*y1y2*y1y3*y2u7-yu7*y1u7*y1y2*y2y3+yu7*y1u7*y1y3*y2y2)
/(y1y1*y2y2*y3y3*u7u7-y1y1*y2y2*y3u7*y3u7-y1y1*y2y3*y2y3*u7u7+2*y1y1*y2y3
*y2u7*y3u7-y1y1*y2u7*y2u7*y3y3-y1y2*y1y2*y3y3*u7u7+y1y2*y1y2*y3u7*y3u7+2
*y1y2*y2y3*y1y3*u7u7-2*y1y2*y2y3*y1u7*y3u7-2*y1y2*y2u7*y1y3*y3u7+2*y1y2*y2u7
*y1u7*y3y3-y2y2*y1y3*y1y3*u7u7+2*y1y3*y2y2*y1u7*y3u7+y1y3*y1y3*y2u7*y2u7-2
*y1y3*y1u7*y2u7*y2y3-y1u7*y1u7*y2y2*y3y3+y1u7*y1u7*y2y3*y2y3);
else if(j==7)
{
B = (-y1y*y1y2*y2y3*y3u7+y1y*y1y2*y2u7*y3y3+y1y*y2y2*y1y3*y3u7-y1y*y2y2
*y1u7*y3y3-y1y*y2y3*y1y3*y2u7+y1y*y1u7*y2y3*y2y3+y2y*y1y1*y2y3*y3u7-y2y*y1y1
*y2u7*y3y3-y2y*y1y3*y1y2*y3u7+y2y*y1y3*y1y3*y2u7+y2y*y1u7*y1y2*y3y3-y2y*y1u7
*y1y3*y2y3-y3y*y1y1*y2y2*y3u7+y3y*y1y1*y2u7*y2y3+y3y*y1y2*y1y2*y3u7-y3y*y1y2
*y1y3*y2u7-y3y*y1u7*y1y2*y2y3+y3y*y1u7*y1y3*y2y2+yu7*y1y1*y2y2*y3y3-yu7*y1y1
*y2y3*y2y3-yu7*y1y2*y1y2*y3y3+2*yu7*y1y2*y1y3*y2y3-yu7*y1y3*y1y3*y2y2)/(y1y1
*y2y2*y3y3*u7u7-y1y1*y2y2*y3u7*y3u7-y1y1*y2y3*y2y3*u7u7+2*y1y1*y2y3*y2u7
*y3u7-y1y1*y2u7*y2u7*y3y3-y1y2*y1y2*y3y3*u7u7+y1y2*y1y2*y3u7*y3u7+2*y1y2
*y2y3*y1y3*u7u7-2*y1y2*y2y3*y1u7*y3u7-2*y1y2*y2u7*y1y3*y3u7+2*y1y2*y2u7*y1u7
*y3y3-y2y2*y1y3*y1y3*u7u7+2*y1y3*y2y2*y1u7*y3u7+y1y3*y1y3*y2u7*y2u7-2*y1y3
*y2u7*y1u7*y2y3-y2y2*y1u7*y1u7*y3y3+y1u7*y1u7*y2y3*y2y3);
B = B*100;
}
else if(j==9)
{
z = 1;
for (i=1;i<100;i++)
{
a = z*z*z + A1*z*z + A2*z + A3;
deriv = 3*z*z + 2*A1*z + A2;
diff = a/deriv;
if (diff > 0.01)
diff = 0.01;
if (diff < -0.01)
diff = -0.01;
z = z - diff;
}
pole = z;
gainfactor = (1-pole)/(1 + A1 + A2 + A3);
gfB = gainfactor*B;
}

```



```

    Ad11 = 1.0;
    Ad12 = h*pole;
    Ad21 = 0.0;
    Ad22 = pole;
    Bd1 = gfB*h;
    Bd2 = gfB;
    Cd1 = 1;
    Cd2 = 0;
    Dd = 0;
}
else if(j==11)
{
    alldone = false;
    identon = false;
    j=0;
}
}
}

```

B.1.3 Recursive least squares identification

The RLS identification computes `theta`, `P_rls` and `K_rls` according to the algorithm in section 3.3.4. For some reason, the B&R Automation Studio does not handle multidimensional arrays in the standard way. Therefore a 4×4 matrix is expressed as a 16×1 vector. In Calculate output, initial values of `theta` and `P_rls` are being set. This is done by the flag `init_rls` which is only true in the first loop.

```

/***** Calculate output *****/
case RLS:
if(init_rls)
{
    theta[0] = -0.917926;
    theta[1] = -0.002529517;
    theta[2] = -0.871672;
    theta[3] = 29.2793;
    lambda = 1;
    /*Covariance matrix*/
    for(row=0; row<=15; row++)
    P_rls[row] = 0;
    for(rls_cnt=0; rls_cnt<=15; rls_cnt=rls_cnt+5)
    P_rls[rls_cnt] = 10;
    zeroposition=position;
    init_rls = false;
}
break;

```

In the Update states part, first all values that are computed by accumulation must be set to zero. After that `K_rls` and `theta` are computed. `P_rls` is computed last, since the Covariance matrix used in `K_rls` shall be delayed one step.

```

/***** Update states *****/
case RLS:
phi[0] = -yone;
phi[1] = -ytwo;
phi[2] = -ythree;
phi[3] = uone;

/***** Setting variables to zero *****/
for(rls_cnt=0; rls_cnt<=3; rls_cnt++){
    K_rls[rls_cnt] = 0;
    K_rls_temp[rls_cnt] = 0;
}
theta_temp = 0;
K_rls_temp2 = 0;

/***** Computation of K_rls *****/
for(col=0; col<=3; col++){
    for(rls_cnt=0; rls_cnt<=3; rls_cnt++){
        K_rls_temp[col] = K_rls_temp[col] + phi[rls_cnt]*P_rls[col + 4*rls_cnt];
    }
}
for(rls_cnt=0; rls_cnt<=3; rls_cnt++)
K_rls_temp2 = K_rls_temp2 + K_rls_temp[rls_cnt]*phi[rls_cnt];
for(row=0; row<=3; row++){
    for(col=0; col<=3; col++){
        K_rls[row] = K_rls[row] + P_rls[row*4 + col]*phi[col];
    }
    K_rls[row] = K_rls[row]/(lambda + K_rls_temp2);
}

/***** Computation of theta *****/
for(rls_cnt=0; rls_cnt<=3; rls_cnt++)
theta_temp = theta_temp + phi[rls_cnt]*theta[rls_cnt];
for(row=0; row<=3; row++)
theta[row] = theta[row] + K_rls[row]*(y-theta_temp);

/***** Computation of P_rls *****/
for(row=0; row<=3; row++){
    for(col=0; col<=3; col++){
        P_rls_temp[row*4 + col] = eye[row*4 + col]-K_rls[row]*phi[col];
        oldP_rls[row*4 + col] = P_rls[row*4 + col];
        P_rls[row*4 + col] = 0;
    }
}
for(row=0; row<=3; row++){
    for(col=0; col<=3; col++){
        for(rls_cnt=0; rls_cnt<=3; rls_cnt++){
            P_rls[row*4 + col] = P_rls[row*4 + col] + P_rls_temp[row*4 + rls_cnt]
*oldP_rls[col + rls_cnt*4];

```

```

    }
    P_rls[row*4 + col] = P_rls[row*4 + col]/lambda;
  }
}
ythree=ytwo;
ytwo=yone;
yone=y;
uone=current;
break;

```

B.1.4 Resampling and computation of LQ parameters

This piece of code is needed since the identification is performed in a 4 ms loop, while the control is performed in a 0.4 ms loop. The algorithm resamples a first order model from sampling period `h_ident` to sampling period `h`. `gfB` is the new numerator of the model and the denominator will be `z - pole`.

```

gfB4ms = gfB;
pole4ms = pole;
pole = exp(log(pole4ms)*(h/h_ident));
gfB = gfB*((1-pole)/(1-pole4ms));

```

The code below computes the L vector for a second order model of the process from current to position. It is presupposed that the model is given as described in section 4.4.2. The Q and R matrices must be given, and Q must be symmetric. First `s12`, `s11` and `s22` are computed by use of some temporary variables. After that the L vector `lq1` and `lq2` can be computed.

```

gfB_2 = gfB*gfB;
pole_2 = pole*pole;
h_2 = h*h;
temp_sqrt = sqrt(q11*(4*q22*gfB_2+4*gfB_2*h*q12+ 4*Ru+4*Ru*pole_2
+ 8*Ru*pole+gfB_2*q11*h_2));

s12_temp2 = 4*gfB_2*gfB*q11*h*q22+4*gfB_2*gfB*q12*q11*h_2+ 4*gfB
*q11*h*Ru + 4*gfB*q11*h*pole_2*Ru +8*gfB*q11*h*pole*Ru +gfB_2*gfB
*q11*q11*h_2*h;

s12_temp1 = gfB_2*q11*h_2*temp_sqrt+2*gfB_2*h*q12*temp_sqrt + 2
*q22*gfB_2*temp_sqrt+2*Ru*temp_sqrt+2*pole_2*Ru*temp_sqrt-4*pole
*Ru*temp_sqrt+s12_temp2;

s12 = 0.25*(gfB*q11*h+4*gfB*q12+temp_sqrt-1.41421356237310
*sqrt(q11*s12_temp1/temp_sqrt))/gfB;
s11 = ((q12*gfB_2*s12-gfB_2*s12*s12+pole*q11*Ru)
/(gfB_2*h*(s12-q12));

s22 = (-q11*Ru-q11*gfB_2*h_2*s11-2*q11*gfB_2*h*s12+gfB_2*h_2*s11
*s11 + 2*gfB_2*h*s11*s12+gfB_2*s12*s12)/(q11*gfB_2);

```

```

den = 1/(Ru + gfB_2*(h_2*s11 +h*(s12 +s12) +s22));
lq1 = (gfB*(h*s11+s12))*den;
lq2 = (gfB*pole*(h_2*s11 +2*h*s12 +s22))*den;

```

B.2 Controllers

B.2.1 RST controller

This is the simplest type of controller used in this project. It is also used in the relay identification for controlling the step sequence.

The Calculate output only consists of the control formula for the RST controller (section 4.3.1). In Update states the reference speed (**vref**) is set.

```

/***** Calculate output *****/
case RST_CTRL:
current = (vref*T-S*y)/R;
break;

/***** Update states *****/
case RST_CTRL:
i++;
if (i>=0 && i<1000)
    vref = 0;
if (i>=1000 && i<6000)
    vref = 0.5*speed;
if (i>=6000 && i<12000)
    vref = 1*speed;
if (i>12000)
    i=0;
break;

```

B.2.2 State feedback controller

In the Calculate output part of state feedback control the current can be computed directly by use of the known variables **posref**, which is the position reference and **relpos**, which is the position relative to the value of the position when the program started and the velocity **y**.

```

/***** Calculate output *****/
case STATE_FEEDBACK:
if(init_sf == true)
{
    zeroposition=position;
    posref = 0;
    refcounter = 0;
}
relpos = position-zeroposition;
current = l1*posref - l1*relpos - l2*y;
break;

```

In the first loop the L vector is determined by use of the specified discrete poles of the closed loop characteristic polynomial, `pole1` and `pole2`. The position reference is also updated.

```

/***** Update states *****/
case STATE_FEEDBACK:
if(init_sf == true)
{
    if (pole1[0]==pole2[0])
    {
        part1 = pole1[0];
        part2 = pole1[1];
        l1 = (part1*part1-2*part1+1+part2*part2)/(gfB*h);
        l2 = (pole+1-2*part1)/(gfB)-h*l1;
    }
    if ((pole1[1]==pole2[1]) && (pole1[0] != pole2[0]))
    {
        part1 = (pole1[0]+pole2[0])/2;
        part2 = (pole1[1]-pole2[1])/2;

        l1 = (part1*part1-2*part1+1-part2*part2)/(gfB*h);
        l2 = (pole+1-2*part1)/(gfB)-h*l1;
    }
    init_sf = false;
}

/***** Reference generator *****/
refcounter++;
if(refcounter==10000)
    posref = 120000;
if(refcounter==20000)
{
    posref = 0;
    refcounter = 0;
}
break;

```

B.2.3 Linear Quadratic controller

First the position error `poserror` and the velocity error `yerror` are computed. Then the velocity error is filtered through a low pass filter. The current is computed by use of these values and the values of `lq1` and `lq2`. Either the L vector is given once, or if the flag `r1s_on` is set to true, the L vector will be updated online. Then this code will instead be an LQ STR. The current is filtered either by a low pass filter or by lag filter. The reason why the lag filter cannot be used at all times is that it will drive the system unstable if the position error or the velocity error are too big. Bumpless transfer of the switching is implemented.

```

/***** Calculate output *****/

```

```

case LQ_CONTROL:
if(init_lq == true)
{
    zeroposition=position;
    load = 0;
    loadcounter = 0;
    refcounter = 0;
    init_lq = false;
}
poserror = position - zeroposition - 0;
yerror = (poserror - oldposerror)/h;

/***** Filtered velocity *****/
yf = 0.92311634638664*yfold + 0.07688365361336*yf1;
yf1 = 0.92311634638664*yf1 + 0.07688365361336*yerrorold;

/***** Control application *****/
current = - lq1*poserror - lq2*yf;

/***** Low Pass filter *****/
fcurrent = (70*h*current+oldfcurrent)/(70*h+1);

/***** Lag filter *****/
fretcurrent = oldfretcurrent*0.99995038011943+ current -
oldcurrent*0.99751900597172;
oldcurrent = current;

/**** Choosing Lag filter or Low passs filter ****/
if(abs(poserror)<5000 && abs(yerror)<20000)
{
    fcurrent = fretcurrent; /* Bumpless transfer */
    current = fretcurrent + load;
}
else
{
    fretcurrent = fcurrent; /* Bumpless transfer */
    current = fcurrent + load;
}
break;

```

The only things done in the Update states part is to generate a load disturbance and to update some parameters.

```

/***** Update states *****/
case LQ_CONTROL:
/***** Disturbance generator *****/
loadcounter++;
if(loadcounter==5000)
t' load = 3;
if(loadcounter==10000)

```

```

    load = 0;
if(loadcounter==15000)
    load = -3;
if(loadcounter==20000){
    load = 0;
    loadcounter = 0;
}
relpos = position-zeroposition;
oldfcurrent = fcurrent;
oldfretcurrent = fretcurrent;
yold = y;
yfold = yf;
yerrorold = yerror;
oldposerror = poserror;
if(rls_on)
{
    ident_mode = 3;
    identon = true;
}
else
    identon = false;
break;

```

B.2.4 Linear Quadratic Gaussian controller

The structure of the LQG controller is very similar to the LQ controller. But now the current is computed by use of the error state estimations. The current must also be saturated in order for the Kalman filter to get the same information as the real process (except for the load disturbance).

```

/***** Calculate output *****/
case LQG_CONTROL:
if(init_lqg == true)
{
    zeroposition=position;
    xhat1 = 0;
    xhat2 = y;
    load = 0;
    loadcounter = 0;
    refcounter = 0;
    goingup = true;
    sintime = 0;
    init_lqg = false;
}

/***** poserror, yerror *****/
poserror = position - zeroposition -0;
yerror = (poserror - oldposerror)/h;
yf = 0.92311634638664*yfold + 0.07688365361336*yf1;

```

```

yf1 = 0.92311634638664*yf1 + 0.07688365361336*yerrorold;
current = - Ld1*xhat1 - Ld2*xhat2;
fcurent = (70*h*current+oldfcurent)/(70*h+1);

if(abs(poserror)<5000 && abs(yerror)<20000)
{
    fcurent = fretcurrent; /* Bumpless transfer */
    satfcurent = current;
    if (satfcurent >= safelimit)
        satfcurent=safelimit;
    if (satfcurent <= -safelimit)
        satfcurent=-safelimit;
    current = current + load;
}
else
{
    fretcurrent = fcurent; /* Bumpless transfer */
    satfcurent = fcurent;
    if (satfcurent >= safelimit)
        satfcurent=safelimit;
    if (satfcurent <= -safelimit)
        satfcurent=-safelimit;
    current = fcurent + load;
}
break;

```

A load disturbance is generated and the error state estimations are updated according to the equations in section 4.3.4.

```

/***** Update states *****/
case LQG_CONTROL:
/***** Disturbance generator *****/
loadcounter++;
if(loadcounter==5000)
    load = 3;
if(loadcounter==10000)
    load = 0;
if(loadcounter==15000)
    load = -3;
if(loadcounter==20000){
    load = 0;
    loadcounter = 0;
}
relpos = position-zeroposition;
oldxhat1 = xhat1;
oldxhat2 = xhat2;

/***** Kalman filter update *****/
xhat1 = (Ad11-Kalmanvector1*Cd1)*oldxhat1 + (Ad12-Kalmanvector1*Cd2)
*oldxhat2 + Bd1*satfcurent + Kalmanvector1*poserror;

```



```

xhat2 = (Ad21-Kalmanvector2*Cd1)*oldxhat1 + (Ad22-Kalmanvector2*Cd2)
*oldxhat2 + Bd2*satfcurrent + Kalmanvector2*poserror;
oldfcurrent = fcurrent;
oldfretcurrent = fretcurrent;
yold = y;
yfold = yf;
yerrorold = yerror;
oldposerror = poserror;
break;

```

B.2.5 Iterative Learning Control

In ILC the first task was to obtain a good reference signal. This signal was found with the step sequence code below.

```

/***** Calculate output *****/
case STEP_SEQUENCE:
if(init_step == true)
{
    zeroposition=position;
    init_step = false;
}
i++;
relpos = position-zeroposition;
if (i>=0 && i<1300)
current = 4;
if (i>=1300 && i<2530)
current = -4;
if (i>=2500)
current = 0;
break;

```

When a suitable reference step had been found, the ILC algorithm below returns a fitting u_k after a couple of iterations.

In the reference generator, the reference signal obtained above is retrieved from the vector `refvector` which is stored in another file. It can then be chosen how big the step will be by multiplying it with the term `step/stepfactor`, where `step` is the wanted amplitude and `stepfactor` is the reference amplitude. The error between the reference signal and the output from the motor (`poserror`) is then calculated as well as the error velocity (`yerror`). These two signals are used in the feedback connection. The filtered `poserror` is used in the update law for u_k in Update states.

In the control application, an LQ controller is used (section 4.3.3). Consideration is taken to the saturation limit of the system. Also notice that `yerror` is low pass filtered.

```

/***** Calculate output *****/
case ILC:

```

```

if(init_ilc == true)
{
    zeroposition=position;
    oldposition=zeroposition;
    nbrofsamples = 0;
    init_ilc = false;
}
relpos = position-zeroposition;

/***** Reference Generator *****/
if(nbrofsamples<=(2500/steptimefactor-1))
{
    reference = refvector[nbrofsamples*steptimefactor];
    reference = (reference*step)/(stepfactor);
}
reference_uk = reference + uk[nbrofsamples];
realposerror = reference - relpos;
poserror = 0.81873075307798*oldposerror + 0.18126924692202*realposerror;
yerror = (relpos-reference_uk - (oldrelpos-oldreference_uk))/h;

/***** Control application *****/
yf = 0.92311634638664*yfold + 0.07688365361336*yf1;
yf1 = 0.92311634638664*yf1 + 0.07688365361336*yerrorold;
current = - lq1*(relpos-reference_uk) - lq2*yf;
satfcurrent = fretcurrent;
if (satfcurrent >= safelimit)
satfcurrent = safelimit;
if (satfcurrent <= -safelimit)
satfcurrent = -safelimit;
current = fretcurrent;
break;

```

In Update states, u_k is updated only if a new step sequence is initiated. The second if-algorithm is for filling the remaining positions in u_k with the final value of the step. When the number of ILC iterations, `nbrofilc`, is greater than the predetermined number of iterations, `ilc_cycles`, the u_k vector is stored and the loop ends.

```

/***** Update states *****/
case ILC:
if(nbrofsamples<=length_uk && nbrofsamples>=timeshift)
{
    uk[nbrofsamples-timeshift] = (uk[nbrofsamples-timeshift] + 0.2*poserror);
}
oldfretcurrent = fretcurrent;
yold = y;
yfold = yf;
yerrorold = yerror;

```

```

oldrelpos = relpos;
oldreference_uk = reference_uk;
oldposerror = poserror;
nbrofsamples++;
if((nbrofsamples >= (2500/steptimefactor+1500)) && (nbrofilc <= ilc_cycles))
{
    for (i=1;i<timeshift;i++)
    {
        uk[nbrofsamples-timeshift+i]=uk[nbrofsamples-timeshift];
    }
    init_ilc = true;
    nbrofilc++;
}
if(nbrofilc>ilc_cycles)
    ctrlon = 0;

/***** LQSTR implementation *****/
if(rls_on)
{
    ident_mode = 3;
    identon = true;
}
else
    identon = false;
break;

```

The last part, RLS implementation, is if you want to use the recursive update law for LQ parameters. This alternative is chosen by setting the boolean variable `rls_on` to true.

C Motor specifications

ETEL Torque motor TMA 0210 100 3TA

	DESIGN CONSTANT	UNIT	VALUE	water cooling
Tp	Peak torque	Nm	311	
Tu	Ultimate torque	Nm	362	
Tc130	Continuous torque (coil at 130)	Nm	85.3	203
Tc80	Continuous torque (coil at 80)		Nm	64.5
Pp	Peak power dissipation (at 20)		W	4899
Pc130	Cont. power dissipation (coil at 130)	W	406	2078
Pc80	Cont. power dissipation (coil at 80)	W	191	
Km	Constant motor (at 200C)	Nm/ W	5.38	
Te	Electrical time constant	ms	3.53	
Rth130	Thermal resistance (coil at 130)	K/W	0.271	0.053
Rth80	Thermal resistance (coil at 80)	K/W	0.315	
2p	Number of poles		44	
J	Rotor inertia	kgm^2	3.09E-2	
m	Motor weight	kg	16.8	
Th	Hysteresis torque	Nm	1.23	
Td	Detent torque	Nm	5.0	
n	Maximum recommended speed	rpm	270	
Water cooling:				
Tw	Water temperature difference	C	5	
Fw	Water flow	l/min	6.0	
Pi	Input pressure	bar	0.6	
	WINDING CONSTANT	UNIT	VALUE	
Kt	Torque constant	Nm/Arms	22.2	
Ku	Back EMF constant [peak value]	V/rad/s	18.0	
R130	Electrical resistance at 130 (*)	Ohm	16.2	
R80	Electrical resistance at 80 (*)	Ohm	14.0	
R20	Electrical resistance at 20 (*)	Ohm	11.3	
L1	Electrical inductance	mH	40.0	
Ip	Peak current	Arms	17.0	
Iu	Ultimate current	Arms	21.2	
Ic130	Continuous current at 130	Arms	4.08	
Ic80	Continuous current at 80	Arms	3.01	
lcw130	Continuous current with water cooling	Arms	18 .5	

Note: All data $\pm 10\%$ (*) : Terminal to terminal