Learning control on the H-Drive

Learning Control on the *H-Drive*

J.Janssens
report nr.: DC2000.26

Id. Nr.: 420628

Supervisors:
Prof. Dr. Ir. M. Steinbuch
Dr. Ir. M.J.G. van de Molengraft

SUMMARY

The *H-drive* is a XY table system with three linear motors. A typical movement which this *H-drive* has to perform is a point-to-point movement. The goal of this project is to minimize the error made during this movement. Many terchniques can be applied to minimize this error. In this report learning control was used. Learning control is based on a fundamental property of the human behavior; to learn from one's previous mistakes. Once you have made a mistake, you learn from it. And the next time you won't make the same mistake. Learning control emulates this behavior, by continuous adaptation of the control signals until the system behaves as desired. An iterative learning controller can suppress the error made in a typical pick-and-place operation up to 250 Hz and reducing this error by a factor 7, as can be seen later on in this report. On simulation level a factor of 1500 is reached. A remarkable phenomenon that can be noticed when one looks at the difference between the learnt- and standardfeedforward, obtained with the learning controller, is that the learnt feedforward begins earlier with it's excitation than the setpoint itself. So one can conclude that, for eliminating some systematical errors, the system has be excited a bit earlier before the setpoint begins and still is a bit exciting after the setpoint has gone.

LIST OF SYMBOLS

r	reference signal
u^k	feedforward at trial *k*
y^k	output at trial *k*
e^k	error at trial *k*
k	trial number
d	reproducible disturbances
L	learning filter
Q	robustification filter
C, K	PID-controller
P, G	system
S	sensitivity function
H	open-loop transfer function
T	time
P_0	setpoint
arf, brf, crf, drf	state-space matrices for the controller
ameas, bmeas, cmeas, dmeas	state-space matrices for the system
lg	learning gain

TABLE OF CONTENTS

1 INTRODUCTION

The *H-drive* (figure 1.1) is a XY table system with three linear motors. The mechanism is used in *Philips Advanced Component Mounters (EMT)* as the basic positioning device for pick-and-place operations of components *Printed Circuit Boards (PCB's).* It is also very similar to the long stroke actuating system of modern wafer scanners. All three linear motors have encoder sensors and are direct drive, electrically commutating motors with current control. The servo systems are decoupled (single-axis) PID type with low-pass and two notches maximum. Feedforward signals are acceleration, velocity and friction feedforward. The servos typically have 30 Hz bandwidth and the mechanics show resonances above 150 Hz. Typical tracking performance is accelerations up to 80 m/s^2, and errors less than 10 μm.

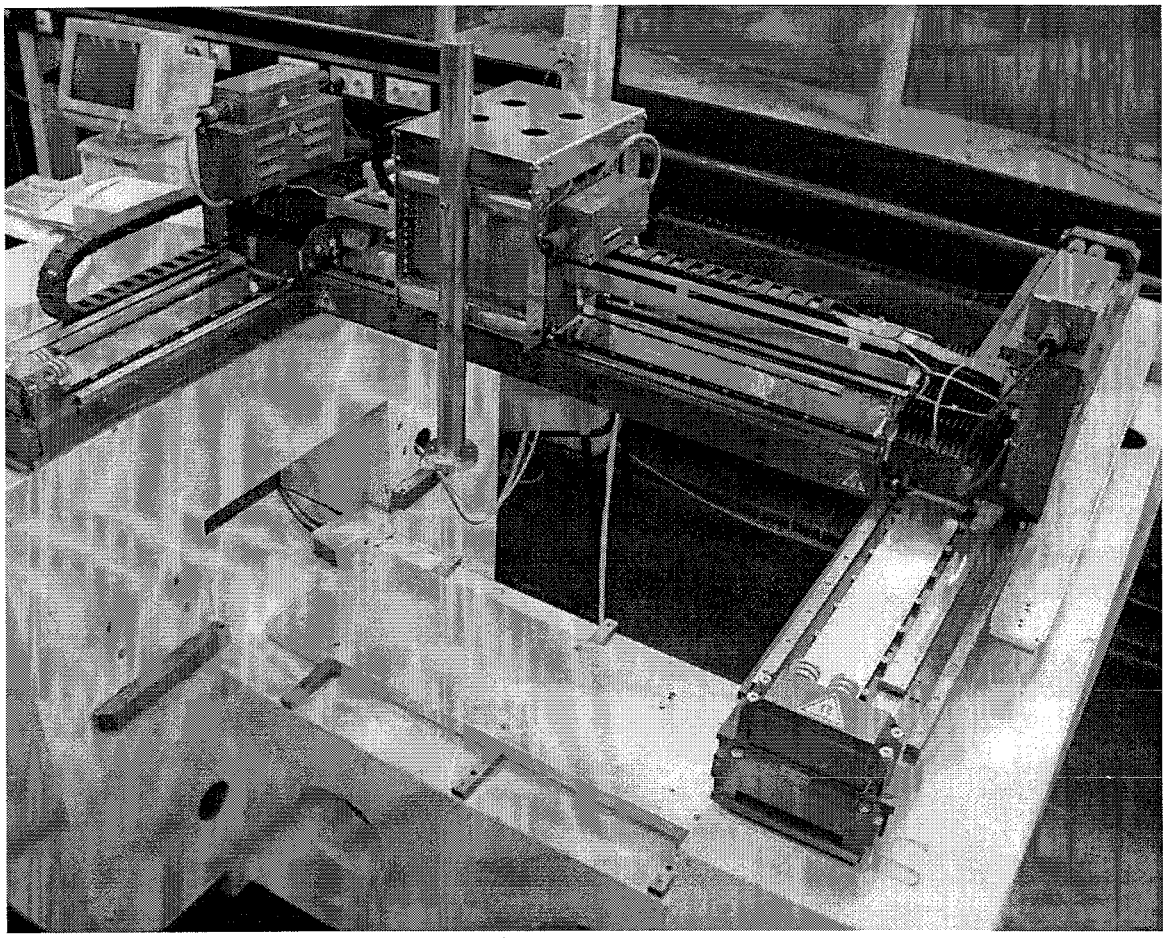


Figure 1.1

A typical movement, which the H-drive has to perform, is a point-to-point movement; the system has to be moved from one prescribed position to another within a minimum amount of time. Mechanical systems such as the H-drive exhibit parasitic dynamics: flexible modes of the system that are excited when the system is moved. This results in a behavior such as in figure 1.2

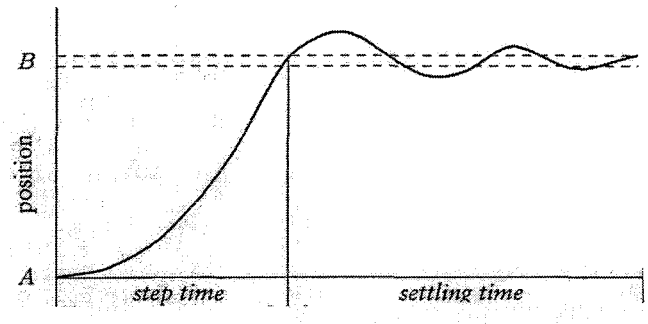


Figure 1.2

From this figure, it can be seen that when the displacement has taken place, the system oscillates in the end point B. When the accuracy constraints are strict, the time needed for these residual vibrations to converge to an acceptable amplitude, the settling time may be quite long. This settling time may even be longer than the step time. Point-to-point control tries to minimize this time.

Besides performing an accurate displacement, the control system also has to be able to track a setpoint and to cope with disturbances within a finite time span. In the past, many successful experiments with learning control have been carried out.

This method is explicitly intended to track setpoints of finite length. The main advantage of learning control is that it is well able to cope with model uncertainties. A point-to-point control must guarantee a minimal error at the end of the movement. Parasitic dynamics make this very difficult. This results in unwanted oscillations at the end of the movement and at this time it is too late to compensate. Two important limitations of learning control are:

- It can only track a known periodic setpoint, because the error is filtered offline. So one can't control an unknown stochastic signal, for example.
- It can only suppress errors up to a certain frequency, because above a certain frequency the robustification filter Q doesn't work anymore (see chapter 2.6).

In order to maintain zero settling error, the learning controller has to actuate the system during the settling time. This means that, although the error is zero, the system is not at rest when a consecutive step is made. The learning controller is still suppressing the residual vibrations.

To obtain short cycle times, the setpoint's energy content has to stretch up to high frequencies. Consequently, the system's high-frequent flexible modes are excited which can be seen in the error. As mentioned, only the lower frequent components of servo errors can be compensated by a learning controller. This frequency problem results in longer settling times.

2 ITERATIVE LEARNING CONTROL

2.1 THEORY

Learning control is based on a fundamental property of the human behavior; to learn from one's previous mistakes. Once you have made a mistake, you learn from it. And the next time you won't make the same mistake. Learning control emulates this behavior, by continuous adaptation of the control signals until the system behaves as desired. Consider the following system. (Figure 2.1.1)

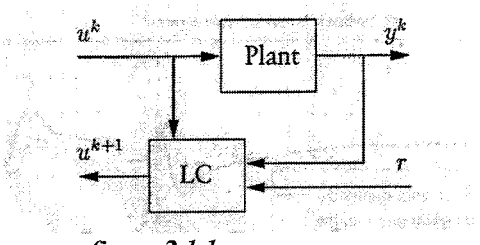


figure 2.1.1

Tracking of the signal r is required. First a signal u^0 is generated, that is believed to be able to generate an output similar to r. The error, y^0-r, is measured. On the basis of this error and the old feedforward u^0 the learning controller (LC) computes a new feedforward u^1, which hopefully results in a better output y^1. This process can be repeated until the error has converged to zero (or to a value specified by the user). (The convergence criterion is explained later on). The result of this is a signal, which is the exact response of the inverse of the plant to the given r without having to invert the plant model. The inversion takes place iteratively, based on the input and output signals.

The main issue in this is the concept of repeatability. The error made must be systematic. This means that non-periodic and stochastic effects cannot be learnt, because they are different at each trial.

2.2 IMPLEMENTATION

The learning controller is an add-on to the closed-loop control system as you can see in figure 2.2.1.

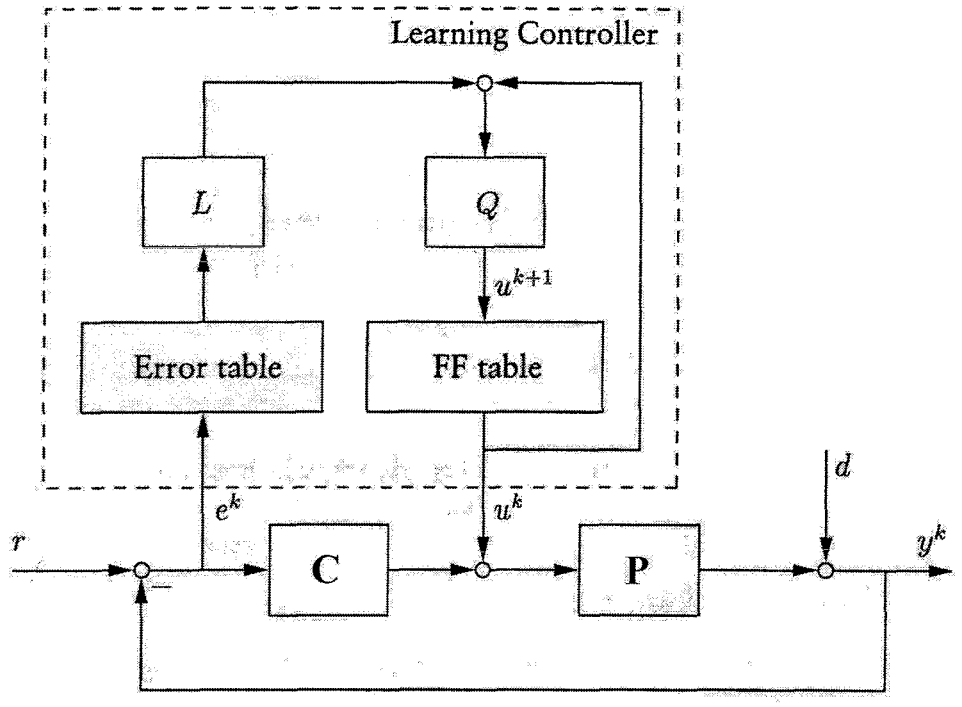


figure 2.2.1

In this figure the following symbols occur:

r	reference signal (setpoint)
y^k	output
d	reproducible disturbances
e^k	error at trial k
u^k	feedforward at trial k
u^{k+1}	feedforward at trial $k+1$
L	learning filter
Q	robustification filter
C	PID-controller
P	system

In the beginning you start with a feedforward u^0 (known as the standard feedforward). This standard feedforward generates an output y^0. The error e^0 is therefore r- y^0. This error can now be filtered through a learning filter L and a robustification filter Q (these filters are discussed in detail in sections 2.5 and 2.6) according to:

$$u^{k+1} = Q(u^k + Le^k) \qquad (1)$$

in which u^{k+1} is the new feedforward that has to be given to the system. This procedure has to be repeated until the error e^k has converged to its minimum value. Ideally, the learning controller now has eliminated all periodic, reproducible components of the disturbance d. Perfect tracking can only be achieved when $u^{k+1} = u^k$. This can only be the case when, according to (1), Q=1 and e^k=0. This would mean that the disturbance d is completely reproducible.

2.3 CONVERGENCE

If r, d and x^0 (initial state) are the same at each trial, the error at the kth trial can be written as:

$$\begin{aligned} E^k(s) &= R(s) - Y^k(s) \\ &= R(s) - \left(P(s)\cdot C(s)\cdot E^k(s) + P(s)\cdot U^k(s) + D(s)\right) \\ &= S(s)\cdot\left(R(s) - D(s)\right) - P(s)\cdot S(s)\cdot U^k(s) \end{aligned}$$

with $E^k(s) = L\{e^k(t)\}, U^k(s) = L\{u^k(t)\}, D(s) = L\{d(t)\}, Y^k(s) = L\{y^k(t)\}$,
$Y^k(s) = \left(P(s)\cdot C(s)\cdot E^k(s) + P(s)\cdot U^k(s) + D(s)\right)$

and

$$S(s)\left(= \frac{1}{1 + P(s)\cdot C(s)}\right)$$

Now at trial k, the learning process can be described by the following two equations:

$$E^k(s) = S(s)\cdot\left(R(s) - D(s)\right) - P(s)\cdot S(s)\cdot U^k(s) \tag{2}$$

$$U^{k+1}(s) = Q(s)\cdot\left(U^k(s) + L(s)\cdot E^k(s)\right) \tag{3}$$

The signal $U^{k+1}(s)$ is injected into the system at trial $k+1$. The resulting error in the $k+1$th trial is then given by (according to (2)):

$$E^{k+1}(s) = S(s)\cdot\left(R(s) - D(s)\right) - P(s)\cdot S(s)\cdot U^{k+1}(s) \tag{4}$$

When equation (3) is substituted in (4) the following equation forms:

$$E^{k+1}(s) = S(s)\cdot\left(R(s) - D(s)\right) - Q(s)\cdot P(s)\cdot S(s)\cdot\left(L(s)\cdot E^k(s) + U^k(s)\right) \tag{5}$$

Since the disturbance $D(s)$ and the setpoint $R(s)$ are assumed to be the same at each trial, one can substitute (2) in (5), which yields to:

$$E^{k+1}(s) = \left(1 - Q(s)\right)\cdot S(s)\cdot\left(R(s) - D(s)\right) - Q(s)\cdot\left(1 - L(s)\cdot P(s)\cdot S(s)\right)\cdot E^k(s) \tag{6}$$

This can be generalized to:

$$\begin{aligned} E^k(s) = {} & Q(s)\cdot\left(1 - L(s)\cdot P(s)\cdot S(s)\right)^k \cdot E^0(s) \\ & + \sum_{j=0}^{k-1}\left(Q(s)\cdot\left(1 - L(s)\cdot P(s)\cdot S(s)\right)\right)^j \cdot\left(1 - Q(s)\right)\cdot S(s)\cdot\left(R(s) - D(s)\right) \end{aligned}$$

In this equation $E^0(s)$ is the error of the first trial. Convergence is proven when this sum exists for $k\to\infty$. This means that this equation has to result in a coinstant final value for the error if $k\to\infty$.

Since:

$$U^{k+1}(s) =$$
$$Q(s)\cdot U^{k}(s) + Q(s)\cdot L(s)\cdot E^{k}(s) =$$
$$Q(s)\cdot U^{k}(s) + Q(s)\cdot L(s)\cdot \left(S(s)\cdot (R(s) - D(s)) + P(s)\cdot S(s)\cdot U^{k}(s)\right) =$$
$$f\left(U^{k}(s)\right)$$

(which means that $U^{k+1}(s)$ is a function of $U^{k}(s)$)
the next inequality holds:

$$\begin{aligned}\left\| f(x_1) - f(x_2)\right\| &= \left\| Q(s)\cdot (x_1 - x_2) - Q(s)\cdot L(s)\cdot P(s)\cdot S(s)\cdot (x_1 - x_2)\right\| \\ &\le \left\| Q(s) - Q(s)\cdot L(s)\cdot P(s)\cdot S(s)\right\|_i \cdot \left\| x_1 - x_2\right\|\end{aligned}$$

This is satisfied when

$$\left\| Q(s)\cdot (1 - L(s)\cdot P(s)\cdot S(s))\right\|_i < 1 \tag{7}$$

where $\left\| \bullet \right\|_i$ is some induced norm (usually i is 2)
For a converged system u does not change over one trial. As a consequence of this neither does the error. Now with equation (6) one can write:

$$\overline{E}(s) = \lim_{k\to\infty} E_k(s) = \frac{(1 - Q(s))\cdot S(s)}{1 - Q(s)\cdot (1 - L(s)\cdot P(s)\cdot S(s))}\cdot (R(s) - D(s)) \tag{8}$$

Which is the value of the converged error.

2.4 FREQUENCY DOMAIN ANALYSIS

Equation (6) from paragraph 2.3 can also be written in the frequency domain as:

$$|Q(\omega)(1-L(\omega)P(\omega)S(\omega))|<1 \quad \forall \quad \omega \in (-\infty,\infty)$$

Since the expressions for Q, L and P are complex, this criterion states that the complex vector Q(1-LPS) must remain within the unit circle for the learning process to converge. This circle will be referred to as the region of convergence (ROC). First assume Q=1. The closer the vector |1-LPS| is to the origin, the faster convergence will be. When however for a given frequency the inverse of the process sensitivity, which is L, is of poor quality, the vector will point outward the ROC. For this frequency, convergence is not guaranteed and oscillations might occur. In figure 2.4.1 one can see a frequency where $L \neq (PS)^{-1}$.

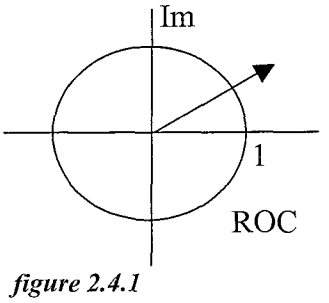


figure 2.4.1

If this vector, (1-LPS), would lie within the unit cycle, no robustification filter Q was needed. However in the case of figure 2.4.1, a robustification filter Q has to be introduced. The effect of Q can be illustrated by writing the convergence criterion as follows,

$$|1-L(\omega)P(\omega)S(\omega)|<\left|\frac{1}{Q(\omega)}\right| \quad \forall \quad \omega \in (-\infty,\infty)$$

This implies that the vector *1-LPS* has to lie within a bigger cycle than the unit cycle by choosing |Q| smaller than 1. Now convergence can be guaranteed at those frequencies at which the inversion is of poor quality. In essence, Q increases the diameter of the ROC from 1 to |1/Q|. The effect of this is shown in figure 2.4.2

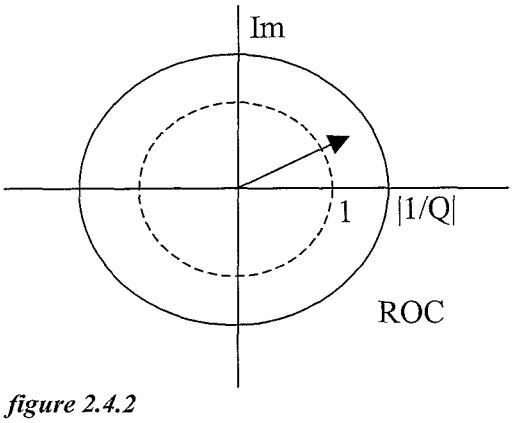


figure 2.4.2

Now the following conclusions can be drawn with respect to the design of the robustification filter Q:

- For low frequencies, L approximates $(PS)^{-1}$ quite well, so choose $Q \approx 1$
- For high frequencies, the radius of the ROC has to be made greater, so choose $Q \ll 1$

When we take a look again on the formula for the converged error, one can see that for low frequencies, where Q is 1, the error $\bar{E}(s)$ becomes zero. For high frequencies, where Q becomes zero, the error will converge to *S(s)(R(s)-D(s))*, which is the same error as without learning. For intermediate frequencies, the error reduction is given by full evaluation of this *error-formula.*

$$\bar{E}(s) = \lim_{k\to\infty} E_k(s) = \frac{(1-Q(s))S(s)}{1-Q(s)(1-L(s)P(s)S(s))}(R(s)-D(s))$$

With respect to this, the cut off frequency of the robustification filter Q, is known as the *'learning bandwidth'*.

2.5 DESIGN OF THE LEARNING FILTER

To achieve fast convergence as well as maximum error reduction, the learning filter L has to approximate the inverse of the process sensitivity PS as closely as possible. The problem is that the discretised PS has zeroes outside the unit circle. Since zeroes become poles after inversion, this implies that the inverse of PS has poles outside the unit cycle, which yields an unstable inverse.
To obtain a stable learning filter, the *zero pole error tracking controller* algorithm (*zpetc.m in DIET Toolbox in Matlab*) was used to split PS into an invertible and a non-invertible part. The algorithm inverts all stable zeroes, yielding stable poles, and cancels the phase shift induced by the unstable zeroes. The phase lag induced by the non-invertible part is cancelled. This cancellation is a forward shift, i.e. a non-causal operation. Since the learning process is performed offline, this non-causality forms no problem: the filtering is done in between learning trials on data sequences that are known beforehand. First, the causal part of the filtering is done. The phase advance is given in number of samples (*phd* in *zpetc.m*). Therefore, shifting the filtered signal backward in time by this number of *phd's* yields the desired non-causal effect.
To check the quality of the inverted model L, a Bode plot is made of L times PS. This should have amplitude of 1 and zero phase for all frequencies. If everything went well this inversion is reasonable until a certain frequency. From this frequency on a robustification filter Q is needed.

2.6 DESIGN OF THE ROBUSTIFICATION FILTER

Since the learning filter is not able to assure convergence throughout the frequency interval of interest, a robustification filter is needed. To compare the effect of different *Q* filters, several filters were examined. They are all low pass Butterworthfilters-type filters with different order and frequencies. The pass band of these filters is equal to one. Since the filtering is performed off-line, zero-phase filtering can be done with the m-file *filtfilt.m* (see *signal Toolbox in matlab 5.3*). This algorithm filters the given sequence and filters it again but backwards. Phase delay is thus cancelled exactly. In figure 2.6.1 the vector *Q(1-LPS)* is plotted, together with the unit cycle. The importance of a *Q* filter is clearly visible in figure 2.6.1a. Here *Q* has been chosen to be equal to 1. As you can see the vector completely leaves the unit cycle (compare with figure 2.4.1). In 2.6.1b *Q(1-LPS)* is plotted where *Q* has a cut off frequency of 50Hz, which implies that all modeled flexible modes of the system lie above this frequency. This figure shows some similarity with figure 2.4.2. The difference lies in the fact that in figure 2.4.2 the vector *1-LPS* was plotted and the radius of the ROC-cycle was enlarged with the help of *Q*. In figure 2.6.1, on the other hand, the vector *Q(1-LPS)* is plotted against the unit cycle. But from both figures the same conclusions can be drawn. In figure 2.6.1d *Q* has a cut off frequency of 500Hz, which is able to cope with all modeled flexible modes. The learning algorithm is stable with a 500Hz *Q* filter, but within a very small margin. When one doesn't work on simulation level, but with a real machine the best *Q* filter to use is one where the region of convergence lies around the circle with radius 0.5. This will be called the robustness cycle. This is shown in figure 2.6.1c.

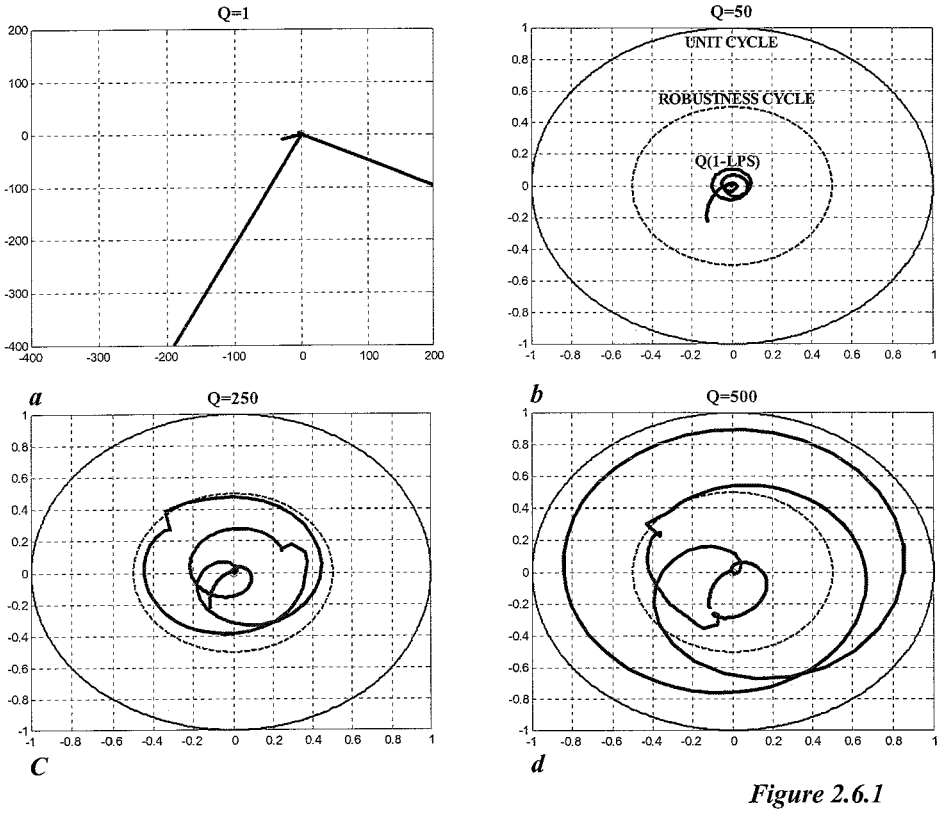


Figure 2.6.1

3 THE PROCEDURE

In this chapter a procedure for designing a correct learning and robustification filter is presented. The sensitivity function and the controller of the system can easily be measured. The sensitivity S is defined according to:

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

This can be transformed to:

$$1 + P(s) \cdot C(s) = S(s)^{-1}$$

$$P(s) \cdot C(s) = S(s)^{-1} - 1$$

$$P(s) = \frac{S(s)^{-1} - 1}{C(s)}$$

Thus when one knows the sensitivity function and the controller of a certain system, the system P(s) can be obtained. When S(s) and C(s) are measured one always encounters measurement noise. Therefore not the entire measurement has to be taken into account. The check the data, a look at the coherence has to be taken. Since P(s) has to be fitted once it has been calculated, this coherence shows the lower and upper bound for this fit. As can be seen later on, a lower bound of approximately 10 Hz and an upper bound of approximately 1000 Hz were used. In figure 3.1a the measured sensitivity function and the measured controller are displayed. In figure 3.1b the corresponding coherence is plotted. As one can see, the lower bound for reliable results for the sensitivity function lays around 20 Hz. Also an upper bound of ± 1000 Hz can be seen in the coherence plot of the sensitivity function. (The coherence of the controller measurement was a lot better, so the sensitivity measurement is the restricting factor).

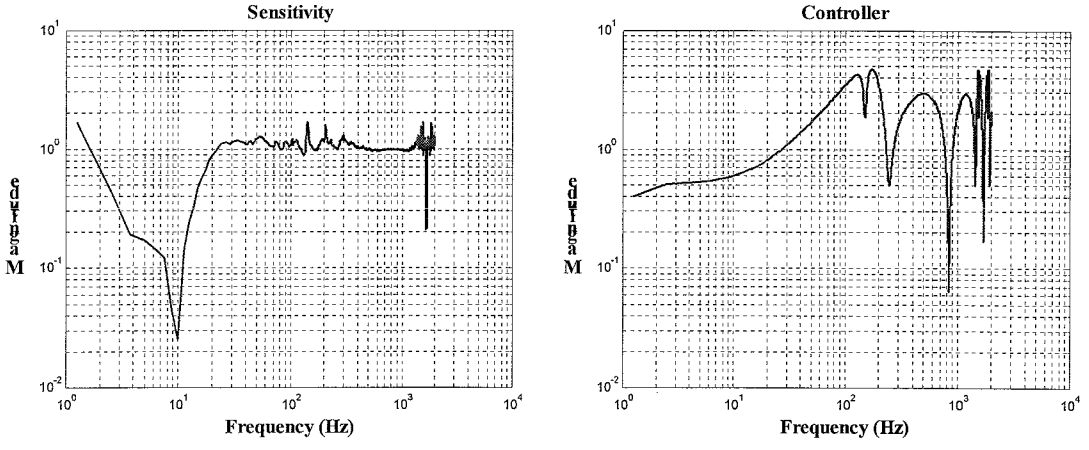


figure 3.1a

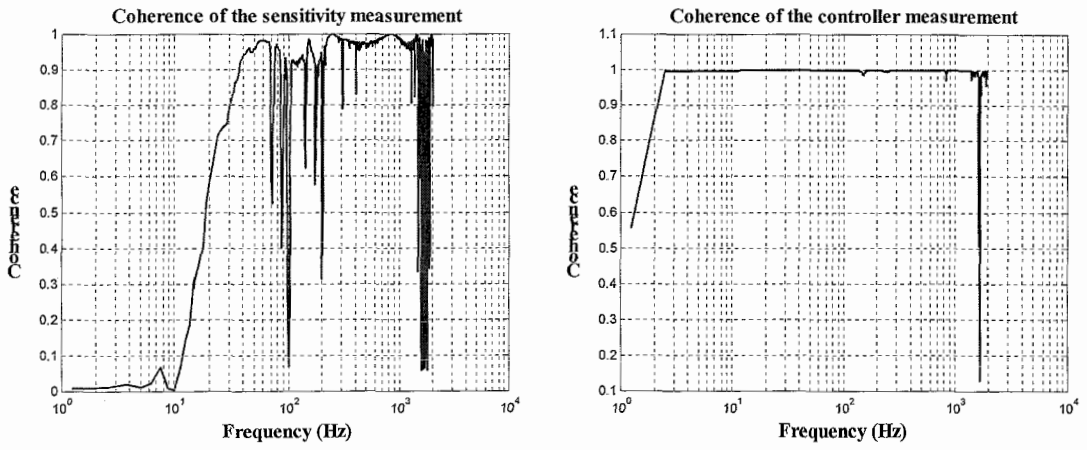


figure 3.1b

When one takes a look at the obtained results for the sensitivity function, it can be concluded that with the used boundaries the result is the same as one would expect. For low frequencies a slope of +2, but for high frequencies zero slope and magnitude one. Also there are some resonances visible between the 100 and 300 Hz. A closer look at the controller tells us that the controller has:

- A notch at about 150 Hz
- A second notch at about 250 Hz
- A low pass filter at about 500 Hz
- A differentiator at about 10 Hz

When this has been done, the system P can be calculated and the following bode diagram can be drawn (figure 3.2).

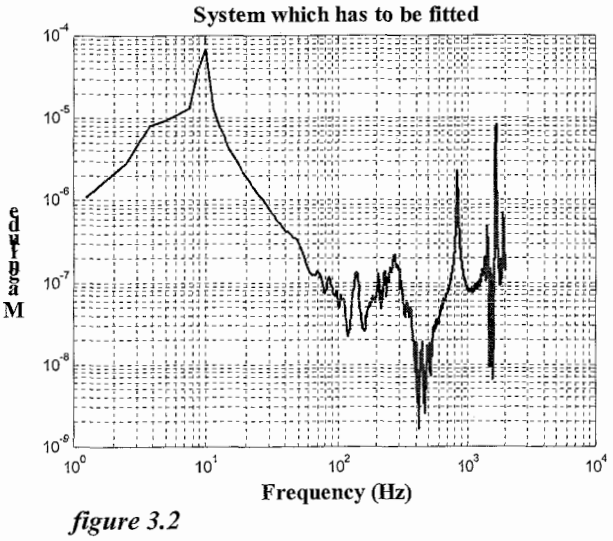


figure 3.2

One would expect a slope of –2 with some resonance peaks. Within the earlier explained range this is true, so after fitting this would be useful data. At this time the system P and the controller C have to be fitted for further calculations. For the fitting the tool *frfit.m* (see *DIET-Toolbox in matlab*) was used. This tool can easily fit frequency response functions. The input data of this tool consist of the frequency response data of the function one wants to fit, a frequency vector, the order of the denominator and the order of the numerator. While using this tool one can easily place

under and upper borders. For example below frequencies of about 10-20Hz the coherence of the sensitivity was too low. So the lower bound in the fitting lies around 15Hz. In the same way the upper bound was chosen around 500Hz. For the controller-fit borders were used of 10Hz below and 1000Hz above. The fitting algorithm also has the options of adding weighting factors. So important intervals can be weighted more heavier than the simpler intervals. After fitting, C(s) (figure 3.3) and especially P(s) (figure 3.4) become clearer.

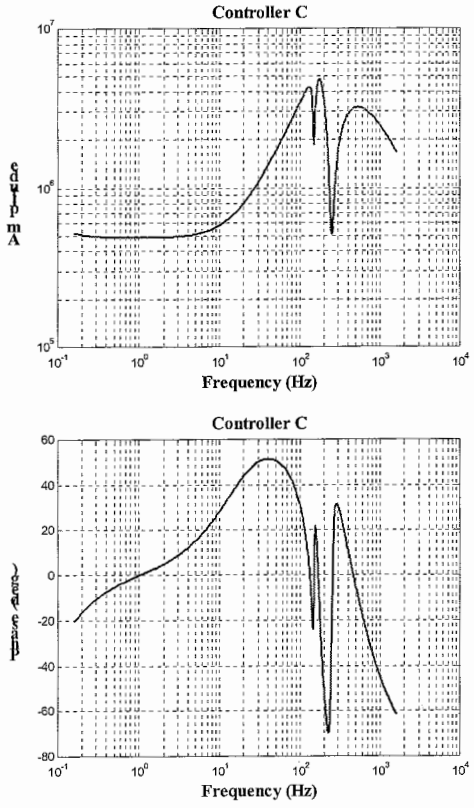


figure 3.3

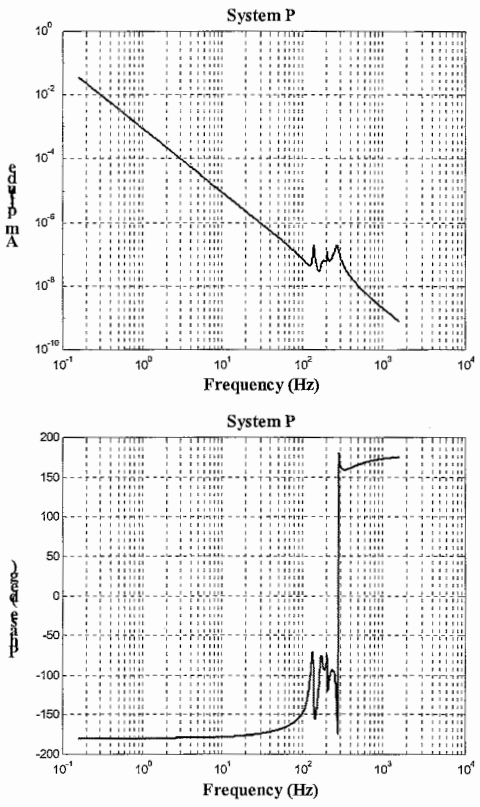


figure3.4

Notice the –2 slope with some resonances peaks of P(s). Now P(s) and C(s) are known, the open-loop PC(s) can be calculated and the bode diagram can be drawn, which has been done in figure 3.5. As can be seen from the nyquist diagram (figure 3.6), the system is stable, with a gain margin of 4 and a phase margin of 50°.

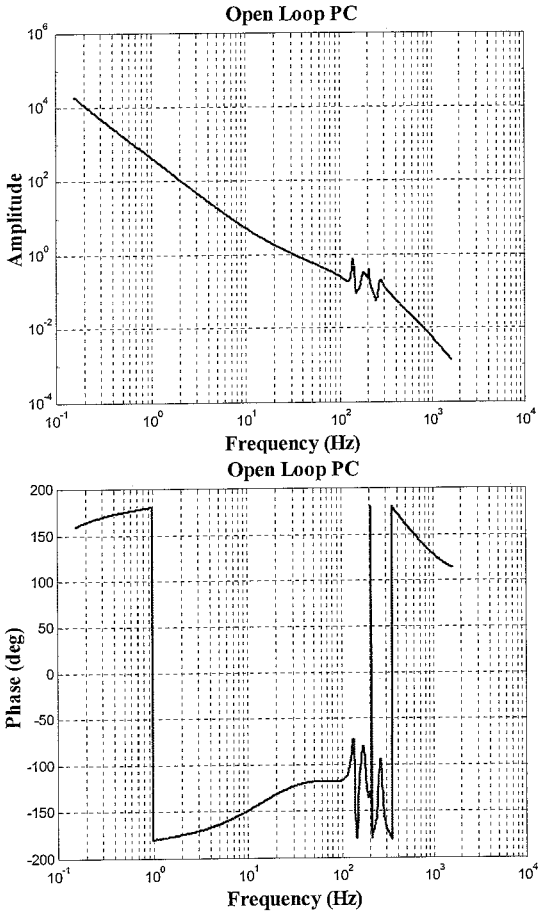


figure 3.5

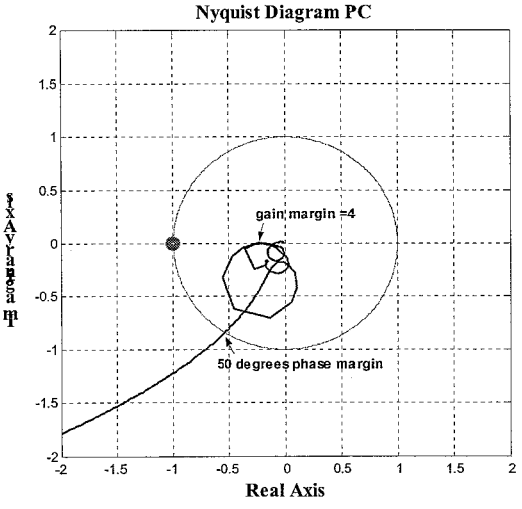


figure 3.6

At this moment one knows P(s), C(s) and PC(s). Since,

$$S(s)=\frac{1}{1+P(s)\cdot C(s)} \text{ and } H(s)=\frac{P(s)\cdot C(s)}{1+P(s)\cdot C(s)}$$

, H(s) and S(s) can be calculated (Figure 3.7).

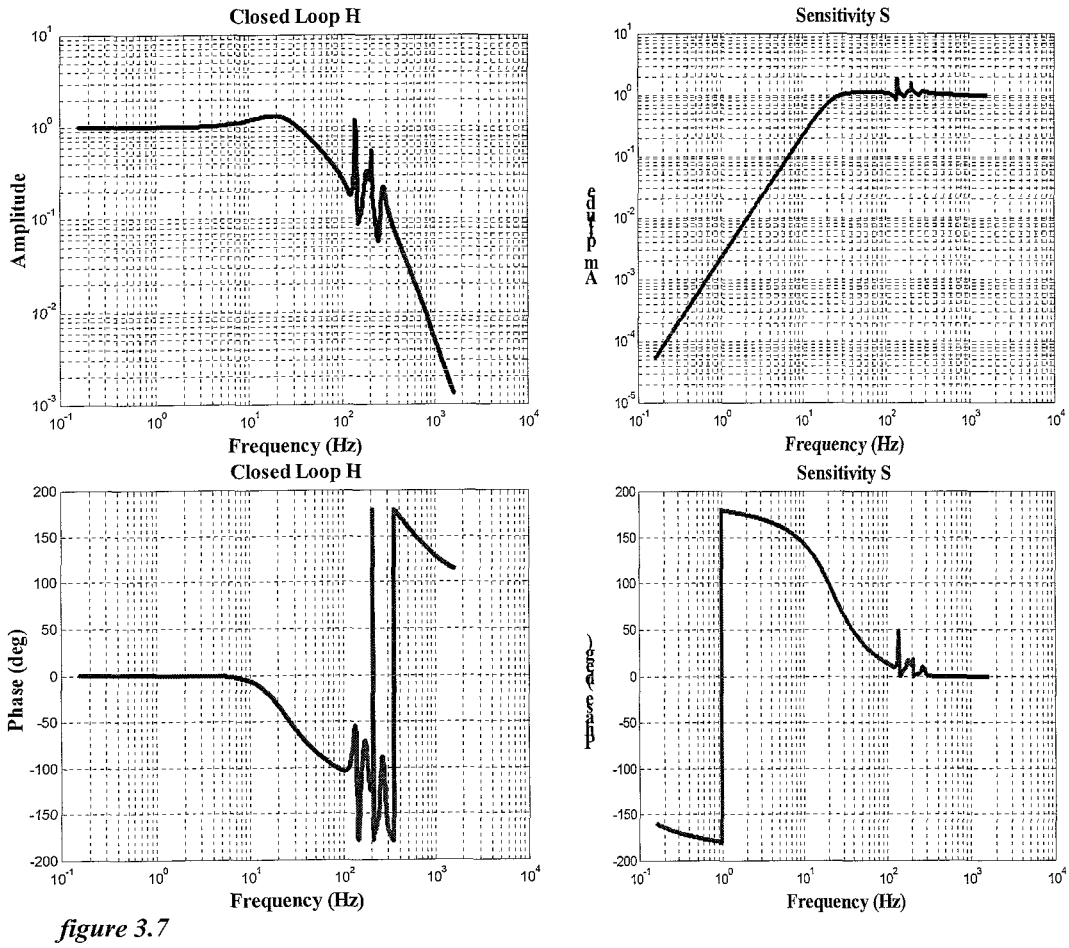
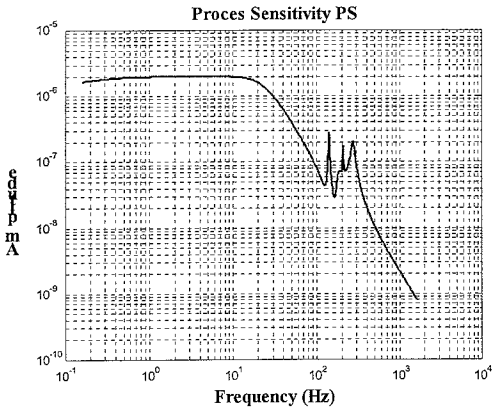
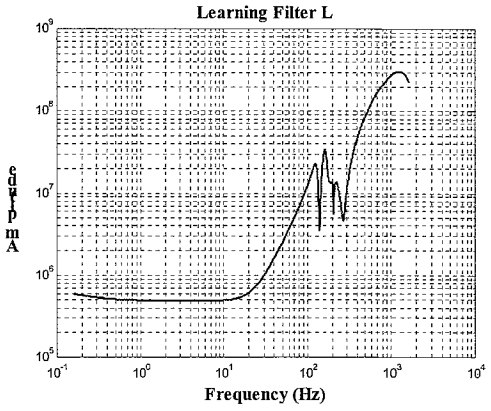


figure 3.7

As one can see the bode diagrams are as one would expect; the visible resonances, the zero slopes (at H(s) for low frequencies and with S(s) for high frequencies), the +2 slope at S(s) for low frequencies and the –2 slope for high frequencies of H(s). In order to calculate a learning filter L, the next step would be to calculate the process sensitivity PS (see section 2.5). Since P(s) and S(s) are known, PS(s) can be calculated. This bode diagram is plotted in figure 3.8. With the *ZPETC* algorithm (explained in section 2.5) L(s) can be calculated. L(s) is plotted in figure 3.9. As explained before L(s) has to be the inverse of PS(s). To check how far this calculated L(s) is a good inverse indeed, one can look at the bode diagram of PS(s) times L(s). If L(s) is the perfect inverse of PS(s), then L(s) times PS(s) has an amplitude of 1 and zero phase. As one can see from figure 3.10 L(s) is reasonable up to 500 Hz. From this point on a robustification filter Q is needed (see section 2.6).





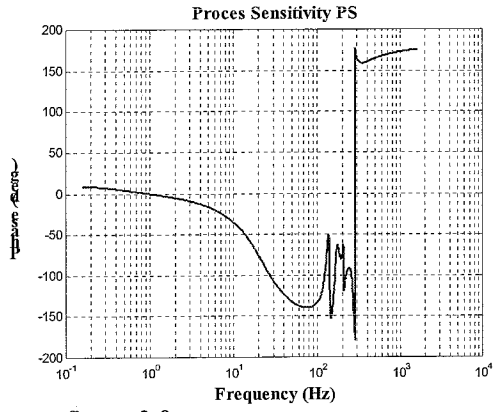


figure 3.8

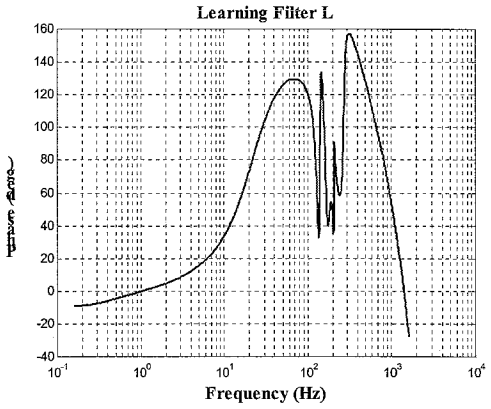
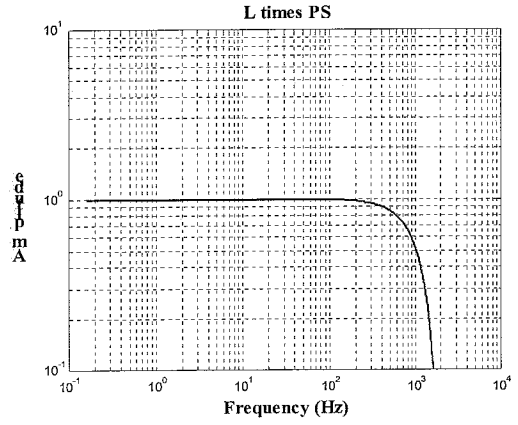


figure 3.9



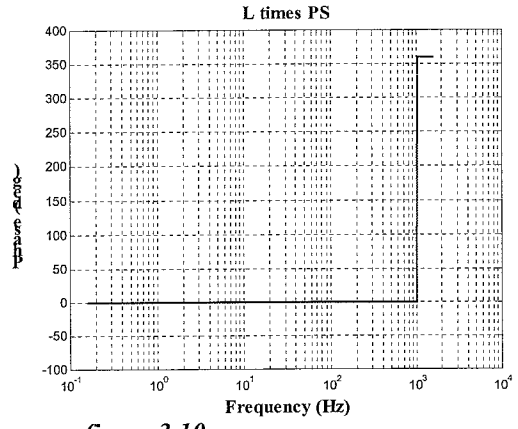


figure 3.10

The Q-filter used is a tenth-order Butter worth filter with a cut-off frequency of 250 Hz. This has been done in order to keep the convergence criterion inside the robustness circle (i.e. the 0.5 region (section 2.6)). At this point the learning filter L(s)

and the robustification filter Q(s) are available, so the time has come for learning on the simulation level. A model of the control scheme was built in Simulink (see Appendix A). In this scheme the following parameters appear:

T=time	[s]
Po=setpoint	[m]
State-space matrices for the controller:	
arf, brf,crf,drf	[-]
State-space matrices for the system:	
ameas, bmeas,cmeas,dmeas	[-]

This simulation procedure is also included in the used m-file *learning.m* (see Appendix B). In this m-file every other step, which has been done in this report, is displayed. The iteration-loop was stopped after 10 iterations. At this point there can be assumed that the error almost completely converged to its minimum value. The results of this simulation will be presented in Chapter 4.

4 RESULTS

4.1 SIMULATION RESULTS

The trajectory on which learning control has to be performed was a displacement over 0.1 meters. This setpoint is obtained using the m-file *setp_3c* from the *DIET Toolbox* in *Matlab*. The following parameters were used:

- Displacement: 0.1 m
- Velocity: 0.5 m/s
- Acceleration: 16 m/s^2
- Jerk: 750 m/s^3

As was explained earlier the first feedforward, which was offered to the system, was the standard feedforward. This feedforward has the value of the acceleration of the system multiplied by the approximate mass of the system. In figure 4.1.1 one can see the error after one simulation (thus after applying the standard feedforward).

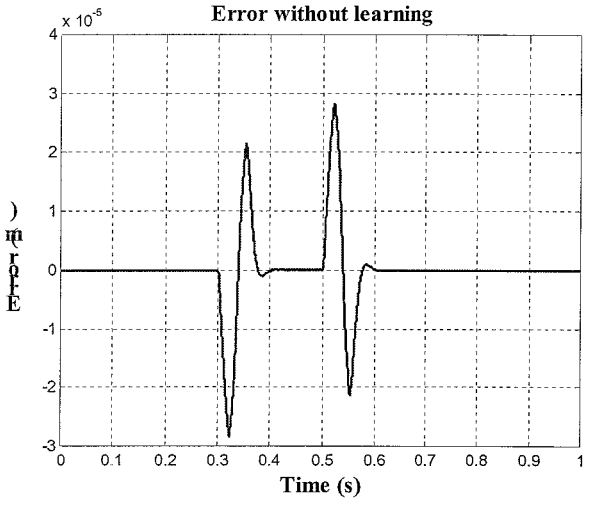


figure 4.1.1

This error is now being filtered through the learning filter L and the robustification filter Q. Since the learning algorithm is defined (section 2.2) according to:

$$U_{k+1}(s) = Q(s)\cdot(U_k(s) + L(s)\cdot E_k(s))$$

and the error after applying the standard feedforward:

$$E_1(s) = Y(s) - PS(s)\cdot U_1(s)$$

The improved input for trial two will be of the form:

$$\begin{aligned} U_2(s) &= Q(s)\cdot(U_1(s) + L(s)\cdot E_1(s)) \\ &= Q(s)\cdot(U_1(s) + L(s)\cdot(Y(s) - P(s)S(s)\cdot U_1(s))) \end{aligned}$$

This feedforward u_2 causes the following error (figure 4.1.2).

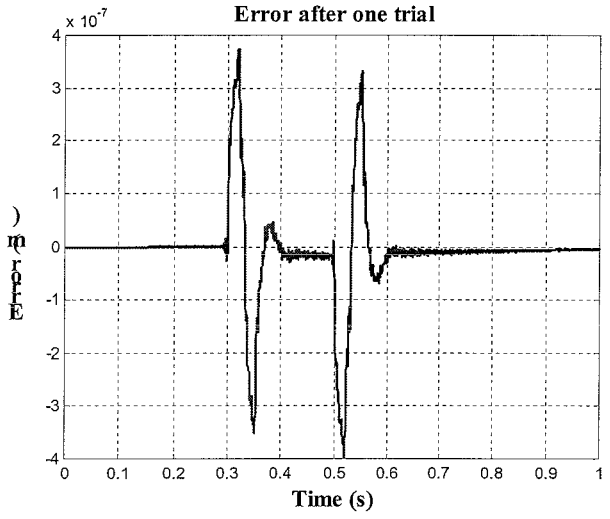


figure 4.1.2

The error after one trial is about a factor 50 times smaller than the error without learning.
The next error is around a factor 10 smaller than the one after one trial (figure 4.1.3).

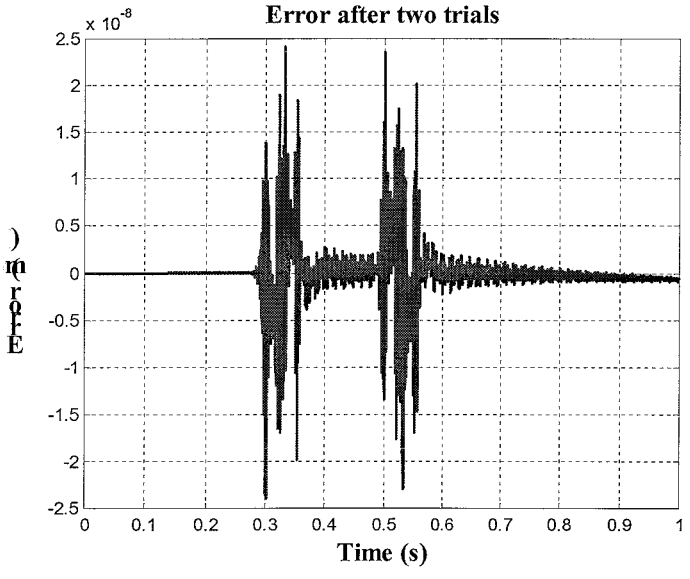


figure 4.1.3

After convergence (the criterion used was to stop after 10 trials) the error has become a factor 1500 smaller than the error without learning control (figure 4.1.4).

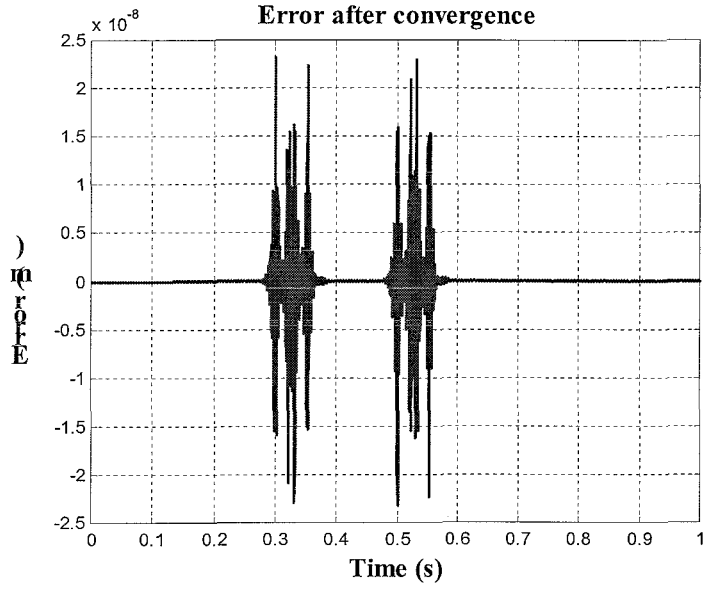


figure 4.1.4

As one can see the error has become very small. One can also adjust the learning algorithm a bit by introducing a learning gain. The learning gain is the value with which the output of the learning filter is multiplied before it goes into the robustification filter. In the next formula 'lg' is the learning gain:

$$U_{k+1}(s) = Q(s)\cdot(U_k(s) + \lg\cdot L(s)\cdot E_k(s))$$

In the simulation the learning gain was chosen to be 1, but in practice it is wise to take this value a bit smaller (i.e. 0.85). After this simulation process, the derived feedforward can be compared with the standard feedforward . As can be seen in the detailed figures from figure 4.1.5, the learnt feedforward starts earlier with its excitation and also ends later

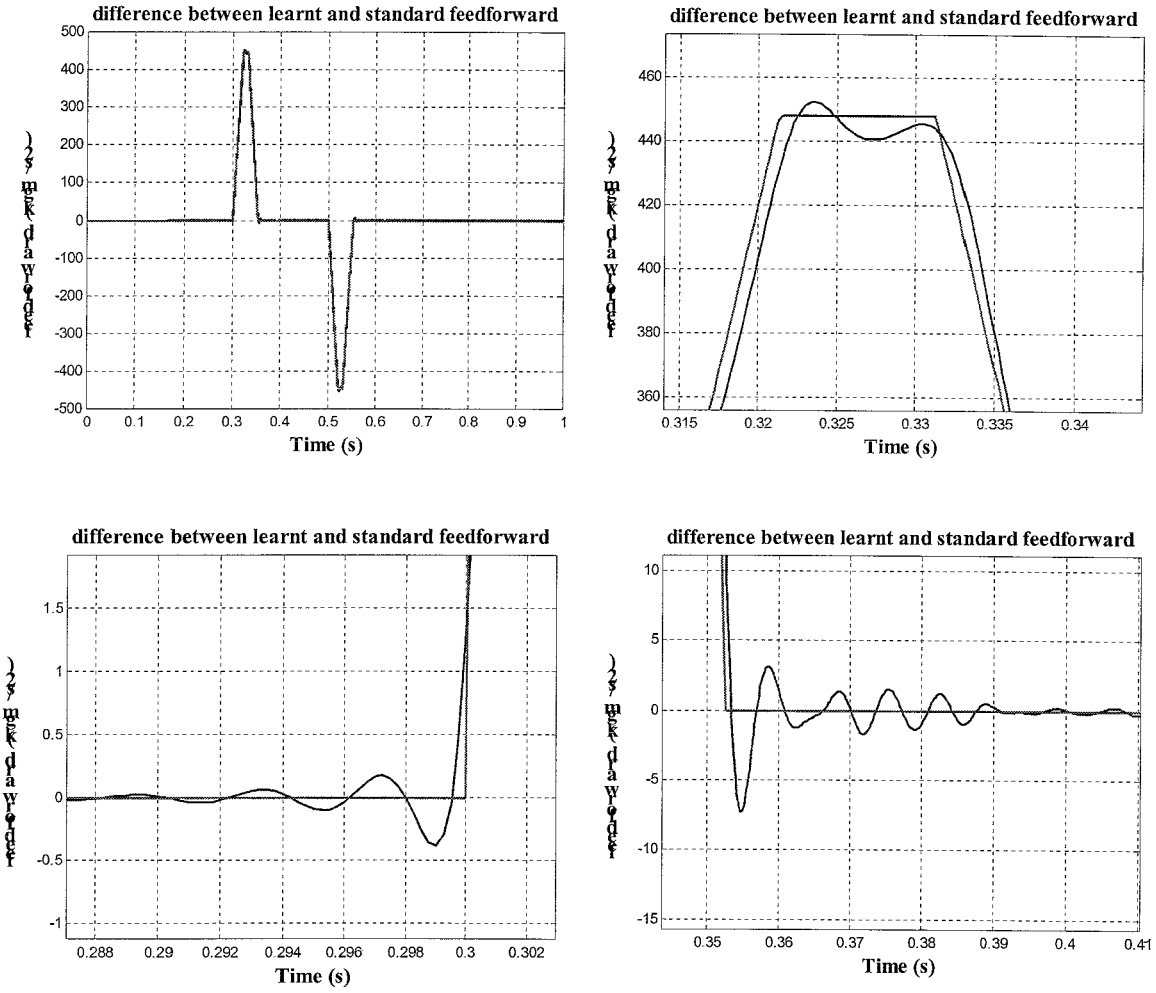

difference between learnt and standard feedforward
Time (s)
difference between learnt and standard feedforward
Time (s)
difference between learnt and standard feedforward
Time (s)
difference between learnt and standard feedforward
Time (s)

figure 4.1.5

4.2 EXPERIMENTAL RESULTS

The experiment with this learning controller on the *H-drive* was performed by the students *M.W.T. Koot* and *S.G.M.Hendriks*. The following information about this experiment were obtained trough personal communication. Although I didn't perform the experiments myself it seemed to me it would be interesting to show just a little bit already. For more detailed reports about the experimental results see the reports of *Hendriks* and *Koot*.

In the experiment the following parameters for the setpoint were used:

Displacement: 0.3 m
Velocity: 0.3 m/s
Acceleration: 5 m/s^2
Jerk: 500 m/s^3

In figure 4.2.1 the results of this experiment are shown. Displayed are the error at the first trial and the error after 10 trials. An *error-decreasing factor* of about 7 is reached here.

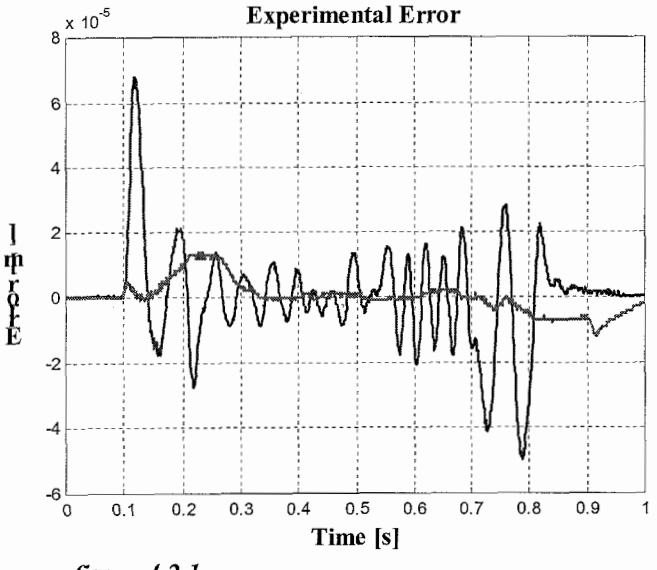


figure 4.2.1

Some remarks have to be made with respect to these results:

- There is still a low-frequent component in the error
- The time-delays in the measurements are not known precisely, so the error- and feedforwardsignal may not be synchronous
- The sensitivity measurement has a low coherence beneath 10Hz but, in spite of this, this part is also used in calculating the L-filter

5 CONCLUSIONS

An iterative learning controller can suppress the error made in a typical pick-and-place operation up to 250 Hz and reducing this error by a factor 7, as can be seen in this report. On simulation level a factor of 1500 is reached. This difference between the experimental and the simulation error is a result of:
the fact that the sensitivity measurement has a low coherence beneath 10Hz but, in spite of this, this part is also used in calculating the L-filter
the fact that time-delays in the measurements are not known precisely, so the error- and feedforwardsignal may not be synchronous
the presence of irreproducible disturbances and noise
the fact that there is always some plant uncertainty

When looking at the difference between the learnt- and standard feedforward, a remarkable phenomenon can be noticed. The learnt feedforward begins earlier with it's excitation than the setpoint itself. So one can conclude that, for eliminating some systematical errors, the system has to be excited a bit earlier before the setpoint begins and is still exciting the system after the setpoint has gone.

Some further research may be done at the following subjects to improve the performance of learning control:

- **Reconsider the feedback control method**

Before learning control is applied, make sure that the servomechanism is already tuned optimally.

- **Reconsider the learning controller design**

Because the current *ZPETC* inversion isn't reliable (continuous to discrete conversion before entering the *ZPETC* algorithm and the fact that it neglects unstable poles), one may develop some better methods to come to a good learning filter.

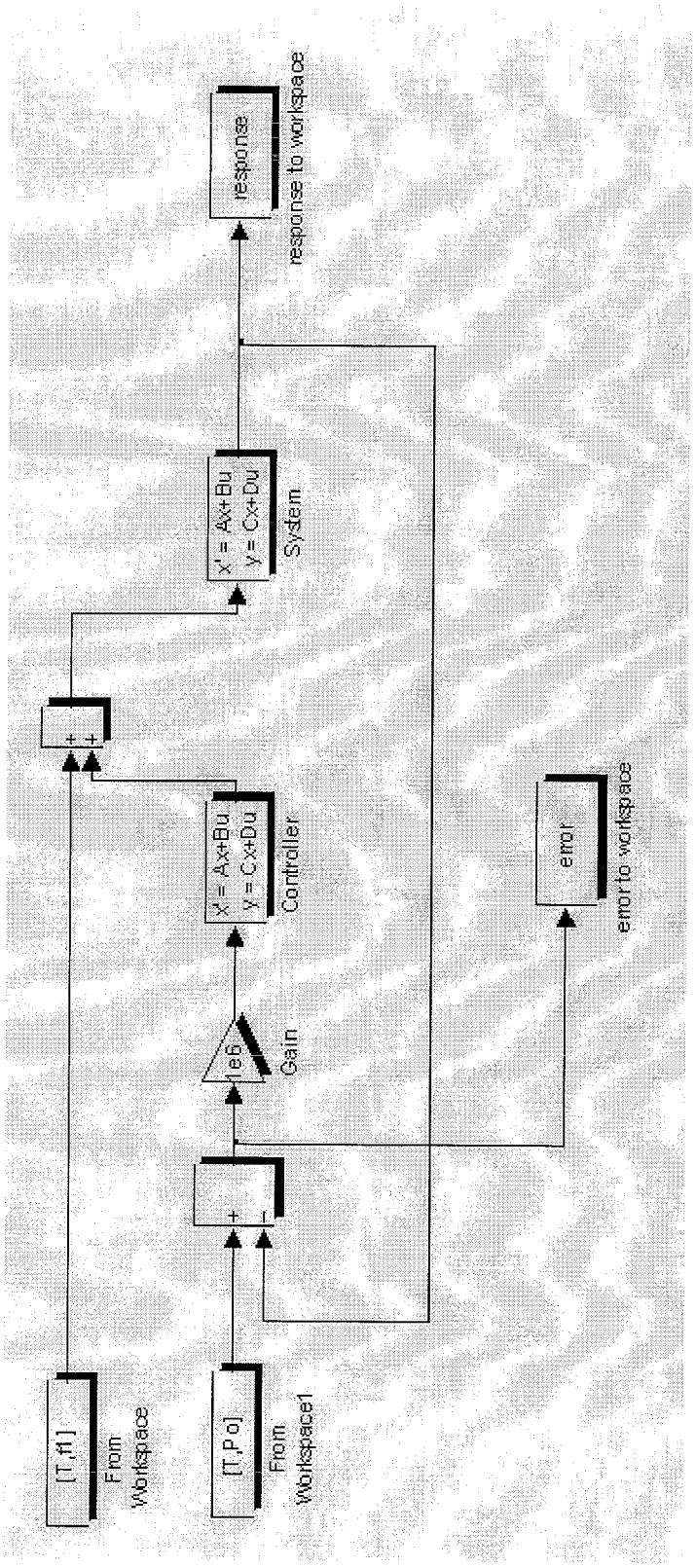
- **Consider the MIMO case**

Inevitable interaction between the x, y1 and y2 directions will mean that the error suppression in one direction may not suffice for the other two directions. This may be solved by a truly multivariable learning controller.

- **Taking operating point dependence in account**

All experiments in this report were performed in one operating point. In reality, the plant's dynamics vary according to the position. A model should be obtained that captures or estimates these variations, to assure convergence over the entire operating range.

APPENDIX A
SIMULINK CONTROL SCHEME



APPENDIX B

LEARNING.M

```
clear all
close all

%sample-time
Ts=0.25e-3;

%introduce setpoint
[Po,V,AC,J,time_vector,T_breakpoints,J_levels]=setp_3c(0.1,0.5,16,750,Ts,0.3
,1,0);
T=time_vector;

figure(1)
subplot(221),plot(T,Po),grid,,ylabel('Position'),xlabel('Time(s)'),title('Po
sition[m]')
subplot(222),plot(T,V),grid,,ylabel('Velocity'),xlabel('Time(s)'),title('Vel
ocity[m/s]')
subplot(223),plot(T,AC),grid,,ylabel('Acceleration'),xlabel('Time(s)'),title
('Acceleration[m/s2]')
subplot(224),plot(T,J),grid,,ylabel('Jerk'),xlabel('Time(s)'),title('Jerk[m/
s3]')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%controller
load regelaar.vna -mat;

contr=SLm.xcmeas(1,2).xfer;
hz=SLm.fdxvec;

figure(2)
loglog(hz,abs(1e6*contr)),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Controller C')

%sensitivity

load Sens7_28_4vec.mat
hzmeas=SLm.fdxvec;
wmeas=hzmeas*2*pi;

figure(3)
loglog(hz,abs(2*xfer)),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Sensitivity S')

fr=1./(2*xfer)-1;

syst=fr./(1e6*contr);

figure(4)
loglog(hz,abs(syst)),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('System P')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrams system P
load fit_plant2.mat

SYSP=ss(ameas,bmeas,cmeas,dmeas);

W=logspace(0,4,1000);
Hz=W/(2*pi);

[ReP, ImP, W] = nyquist (SYSP,W);
P = ReP + j*ImP;

A2(:,1) = abs(P);
F2(:,1) = angle(P)*(180/pi);
```

```
figure(5)
subplot(2,1,1),
loglog(Hz,A2),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude P')

subplot(2,1,2),
semilogx(Hz,F2),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase P')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrams controller C
load regmatr.mat

SYSC=ss(arf,-brf,crf,-drf);

SYSC=SYSC*1e6;

[ReC, ImC, W] = nyquist (SYSC,W);
C = ReC + j*ImC;

A3(:,1) = abs(C);
F3(:,1) = angle(C)*(180/pi);

figure(6)
subplot(2,1,1),
loglog(Hz,A3),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude C')

subplot(2,1,2),
semilogx(Hz,F3),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase C')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrams open-loop PC

SYSPC=series(SYSP,SYSC);

[RePC, ImPC, W] = nyquist (SYSPC,W);
PC = RePC + j*ImPC;

A4(:,1) = abs(PC);
F4(:,1) = angle(PC)*(180/pi);

figure(7)
subplot(2,1,1),
loglog(Hz,A4),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude PC')

subplot(2,1,2),
semilogx(Hz,F4),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase PC')

%nyquistdiagram

figure(8)
Axis([-2 0.5 -2 2])
nyquist(SYSPC,{1,1000})

hold on

plot(-1,0,'ro')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrams sensitivity S

SYSS=feedback(1,SYSPC);
```

```
[ReS, ImS, W] = nyquist (SYSS,W);
S = ReS + j*ImS;


A5(:,1) = abs(S);
F5(:,1) = angle(S)*(180/pi);

figure(9)
subplot(2,1,1),
loglog(Hz,A5),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude S')

subplot(2,1,2),
semilogx(Hz,F5),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase S')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrammen closed-loop H

SYSH=feedback(SYSPC,1);

[ReH, ImH, W] = nyquist (SYSH,W);
H = ReH + j*ImH;


A6(:,1) = abs(H);
F6(:,1) = angle(H)*(180/pi);

figure(10)
subplot(2,1,1),
loglog(Hz,A6),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude H')

subplot(2,1,2),
semilogx(Hz,F6),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase H')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%numerical cleaning
[a,b,c,d]=ssdata(SYSC);
sysc=pck(a-eye(size(a)),b,c,d);
syscbal=Gbalr(0,sysc);
[a1,b1,c1,d1]=unpck(syscbal);
SYSCnew=ss(a1,b1,c1,d1);

[a,b,c,d]=ssdata(SYSP);
sysp=pck(a-eye(size(a)),b,c,d);
syspbal=Gbalr(0,sysp);
[a2,b2,c2,d2]=unpck(syspbal);
SYSPnew=ss(a2,b2,c2,d2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrams proces sensitivity PS

SYSPS=feedback(SYSPnew,SYSCnew);

%discretisation
SYSPSd=c2d(SYSPS,Ts);

[RePS, ImPS, W] = nyquist (SYSPS,W);
PS = RePS + j*ImPS;

A7(:,1) = abs(PS);
F7(:,1) = angle(PS)*(180/pi);

figure(11)
subplot(2,1,1),
```

```
loglog(Hz,A7),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude PS')

subplot(2,1,2),
semilogx(Hz,F7),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase PS')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bode diagrams learning filter L

[aPSd,bPSd,cPSd,dPSd]=ssdata(SYSPSd);

[aL,bL,cL,dL,phd]=zpetc(aPSd,bPSd,cPSd,dPSd,1);

SYSL=ss(aL,bL,cL,dL,Ts);
[mL,pL]=bode(SYSL,W);

[ReL, ImL, W] = nyquist (SYSL,W);
L = ReL + j*ImL;

A8(:,1) = abs(L);
F8(:,1) = angle(L)*(180/pi);

figure(12)
subplot(2,1,1),
loglog(Hz,A8),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude L')

subplot(2,1,2),
semilogx(Hz,F8),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase L')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrams L times fitted PS

SYSLPS=series(SYSL,SYSPSd);

[ReLPS, ImLPS, W] = nyquist (SYSLPS,W);
LPS = ReLPS + j*ImLPS;

A9(:,1) = abs(LPS);
F9(:,1) = squeeze(angle(LPS)*(180/pi))+((180/pi)*(W*Ts*phd));

figure(13)
subplot(2,1,1),
loglog(Hz,(A9)),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude fitted LPS')

subplot(2,1,2),
semilogx(Hz,(F9)),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase fitted LPS')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrams L times measured PS

Smeas=(SLm.xcmeas(1,2).xfer).*2;
[mL2,pL2]=bode(SYSL,wmeas);
mL2=squeeze(mL2);pL2=squeeze(pL2);
SP=Smeas.*syst;
phaseSP=((angle(SP))./pi)*180;

figure(14)
subplot(211)
loglog(hzmeas,mL2.*abs(SP)),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude measured LPS')

subplot(212)
```

```
semilogx(hzmeas,pL2+((180/pi)*(wmeas*Ts*phd)')+phaseSP),grid,ylabel('Phase
(deg)'),xlabel('Frequency (Hz)'),title('Phasekarakteristiek measured LPS')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%bodediagrams robustification filter Q
%cut-off frequency
Qfreq=250;
order=10;
[numQ,denQ] = butter(order,Qfreq/(0.5*(1/Ts)));

SYSQ=tf(numQ,denQ,Ts);
[mQ,pQ]=bode(SYSQ,W);

[ReQ, ImQ, W] = nyquist (SYSQ,W);
Q = ReQ + j*ImQ;

A10(:,1) = abs(Q);
F10(:,1) = angle(Q)*(180/pi);

figure(15)
subplot(2,1,1),
loglog(Hz,A10),grid,ylabel('Amplitude'),xlabel('Frequency
(Hz)'),title('Amplitude Q')

subplot(2,1,2),
semilogx(Hz,F10),grid,ylabel('Phase (deg)'),xlabel('Frequency
(Hz)'),title('Phase Q')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%convergence criterion fitted LPS
%cirkel with radius 1
cc=0:0.01:2*pi;
cx=sin(cc);
cy=cos(cc);
figure(16)
plot(cx,cy)

hold on

ccx=0.5*cx;
ccy=0.5*cy;
plot(ccx,ccy)

hold on

plot(0,0,'ro')

hold on

%criterion
crit=Q.^2.*(1-L.*(P./(1+P.*C)));
[cr1,n]=shiftdim(crit);
plot(cr1),title('convergence criterion for fitted LPS'),grid

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
%convergence criterion measured LPS
%cirkel with radius 1
cc=0:0.01:2*pi;
cx=sin(cc);
cy=cos(cc);
figure(17)
plot(cx,cy)

hold on

ccx=0.5*cx;
ccy=0.5*cy;
plot(ccx,ccy)
```

```
hold on

plot(0,0,'ro')

hold on

%criterion for measured LPS
[mP,pP]=bode(SYSPnew,W);
[mC,pC]=bode(SYSCnew,W);
Pl=p2r(mP,pP);
Cl=p2r(mC,pC);
Ql=p2r(mQ,pQ);
pL=squeeze(pL);
mL=squeeze(mL);
L=p2r(mL,pL+((180/pi)*(W*Ts*phd)));
Ql=squeeze(Ql);Pl=squeeze(Pl);Cl=squeeze(Cl);
conv1= Ql.^2 .* ( 1 - L.*(Pl./(1+Pl.*Cl)) );

[cr1,n]=shiftdim(conv1);
plot(cr1),title('convergence criterion for measured LPS'),grid

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%filtering

%simulation standard feedforward
%standard feedforward
f1=28*AC;
sim('forwardf1')
figure(18)
plot(T,error),grid,ylabel('Error'),xlabel('Time'),title('Error without
learning')

%learning gain
lg=1;

%simulation first new feedforward
e0=error;
Le0=dlsim(aL,bL,cL,dL,e0);
Le0=[Le0(phd+1:4001);zeros(phd,1)];
f1=filtfilt(numQ,denQ,(f1+Le0));
sim('forwardf1')

figure(19)
plot(T,error),grid,ylabel('Error'),xlabel('Time'),title('Error after one
trial')

%simulation second new feedforward
e1=error;
Le1=dlsim(aL,bL,cL,dL,e1);
Le1=[Le1(phd+1:4001);zeros(phd,1)];
f1=filtfilt(numQ,denQ,(f1+Le1));
sim('forwardf1')

figure(20)
plot(T,error),grid,ylabel('Error'),xlabel('Time'),title('Error after two
trials')

%number of trials
cnt=2;

%filterloop

%prescribe maximum error
%while norm(fout,inf) > 1e-8

%or prescribe maximum iterations
for i=1:8;
```

```
cnt=cnt+1;
e1=error;
Le1=dlsim(aL,bL,cL,dL,e1);
Le1=[Le1(phd+1:4001);zeros(phd,1)];
f1=filtfilt(numQ,denQ,(f1+(lg*Le1)));
sim('forwardf1')
figure(21)
plot(T,error),grid,ylabel('Error'),xlabel('Time'),title('Error after xth
trials')
end

trials = cnt
maximum_error=norm(error,inf)

%standard and learnt feedforward
figure(23)
plot(T,f1),grid,ylabel('feedforward'),xlabel('Time'),title('learnt and
standard feedforward')
hold on
plot(T,f0,'r')

%difference between learnt feedforward en standard feedforward
figure(24)
plot(T,f1-f0),grid,ylabel('feedforward'),xlabel('Time'),title('difference
between learnt and standard feedforward')
```

APPENDIX C
BIBLIOGRAPHY

Chang F.K.K. (1997). Learning Control and Setpoint Design, *Application to a Wafer Stepper*.

Gaal E.W. (1995). Tuning and settling performance of the ALERT H-drive.