Observer with delayed sampled data

Hommels, R.P.M.

Award date:
2009

Eindhoven University of Technology
Department of Electrical Engineering
Control Systems

TU/e

Observer with delayed sampled data

by R.P.M. Hommels

Master of Science Thesis

Name:
R.P.M. Hommels
Student number:
s021848

Project period: December 2006 - October 2007
Report Number: 07A-06

Commissioned by:
P.P.J. van den Bosch

Supervisor:
A. Juloski

Contents

List of Figures

List of Tables

1 Introduction

The target is to optimize a printed circuit board(PCB) assembly machine of Assembléon also known as a pick and place machine. The present PCB machines are capable of placing 150.000 components in one hour with an accuracy of 40 micrometers. One of the problems at these speeds is that the cable slabs wear out. Because these machines are sold worldwide maintaining them is expensive. A possible solution to this problem is to replace the current cable slabs with wireless communication.
The pick and place machines of Assembléon has 4 degrees of freedom. Besides allowing movement on the x-, y- and z-axis this machine is capable to turn the components in the right angle with the ϕ-axis.
Controlling these axis wireless with the desired accuracy is quite a challenge. One of the problems that occur is the fact that wireless communication is much slower than wired communication. This will result in communication delays. A delay in the control loop can result in inaccuracy and even instability. Also the mutual communication between the axis will be delayed. So the different axes have no real-time position information of each other. This can slow down the placement and the desired placement-ratio will not be achieved. This aspect that will be handled within this thesis.

2 Problem Description

First we will look where it is possible to replace wired communication with wireless communication. Two wireless control structures occur and will be further analyzed. The two possible structures are depicted in figure 1 and figure 2. We will call then respectively structure A and structure B. Both structures bring their own advantages and disadvantages.

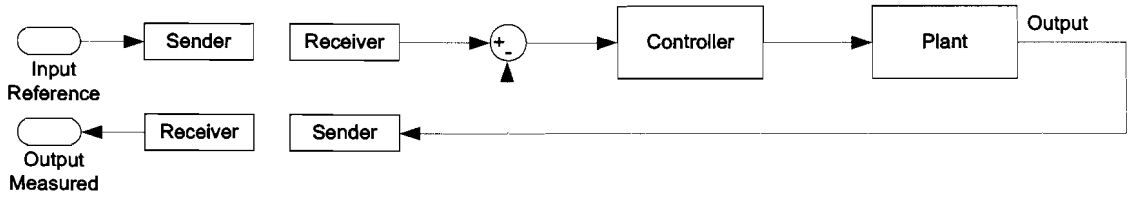


Figure 1: *A classic closed-loop structure(A)*

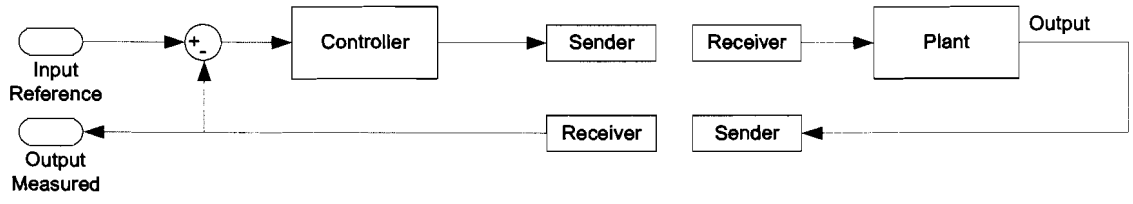


Figure 2: *A closed-loop over a wireless channel structure(B)*

Option A is by means of a classic closed-loop control scheme. Only the reference values and the measured output values will be transmitted wireless. Option B is a closed-loop over a wireless channel. The advantages and disadvantages are shown in table 1.

Table 1: *Compare structure options*

Structure A	Structure B
+ always stabilizing in a case of a unstable plant	− possibility to instability in case of an unstable plant
+ less fast and reliable wireless link needed (position controlled)	− fast and more reliable wireless network needed (real-time)
+ no delay the in control loop	− delay can destabilize the control loop
− no central control point with real-time information of all axes is available	+ one central control point with real-time information of all axes is available

In this thesis we will focus on structure A, because this is the structure that Assembléon considers implementing in their pick and place machines. There will be 2 local controllers. One controller will be responsible for the y-movement and the other controller for the movements of the x-, z- and phi-axis. Before the z-movement, placement of the component, can occur, the controller must be sure that the y- and x-axis are on the right spot and the ϕ-axis has the right angle.

2.1 Overall picture

In figure 3 a possible control structure is brought up. To the y-actuator no moving cables are attached so it is unnecessary to replace the existing wiring with a wireless connection. The three controllers have a collective clock to be able to control the different actuators synchronously.

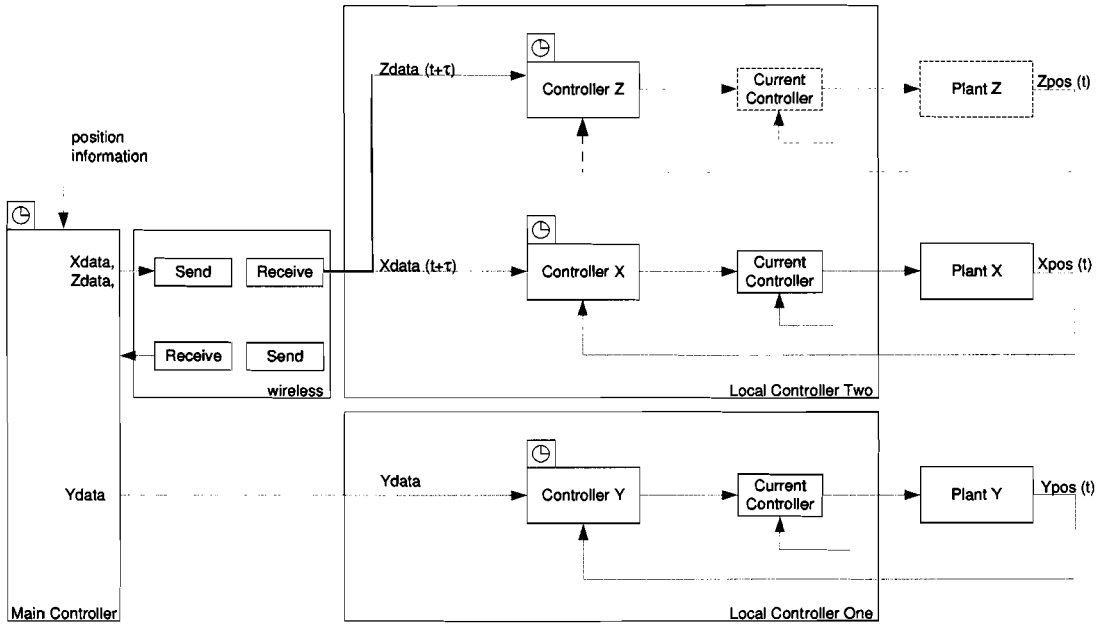


Figure 3: *Illustration of a possible control structure*

To determine the positions of the components that need to be mounted a snapshot of the PCB will be made. Now the positions of the components can be determined and will be processed. Every snapshot will deliver position information on the next five components that need to be placed. This contains the pickup coördinates and the place coördinates. Now for every component the optimal position profiles can be generated. Accordingly a start time is determined so the x- and y-axis will start synchronously.
The placing/picking of the next five components will occur successive. But before a com-

ponent can be placed or picked the position of the x-axis and y-axis must be confirmed. In other words the z-controller must be updated with the position of the x-axis and the y-axis. This will ensure that the right component will be picked on the desired place. When starting with the next position profile the axis will be in their start position. This position must also be confirmed.
But here the wireless connection between the controllers will interfere. As indicated before the position information is not real-time available. Of course we don't want to wait on position information to be transmitted. This will take too much time and we can't comply with the current specifications. An option is to estimate the position. In this thesis we will look if an estimation is fast and accurate enough.
We will estimate the position of the x-axis and y-axis. So the controller Z and controller X from figure 3 can be provided with the position information of the y-axis. This way controller Z has enough information to place a component and controller Y knows when to proceed with the next position profile. The controller Y will be provided with the position information of axis X. So this controller can also independently proceed with the next position profile.
This problem will be singled out in this thesis.

We will start by getting full control of the pick an place machine Assembléon donated for research purposes. This machine is delivered with a embedded computer. Because we want to be in full control we will replace this computer with our own research environment. For that some adjustments are made on the system. These are explained in chapter 3
Before taking full control it is wise to study the hardware. In chapter 4 linear permanent magnet motors are analyzed and a relation between the current and the force is determined. Now we have established a full understanding of the system we are able to control the pick and place machine. In this chapter also the model of the x-axis and y-axis are designed.
The controllers designed for both motors are explained in chapter 5.
Now the model of the motor and the controller is known a good model of the overall system can be made. This is done in chapter 6. This model will be used to design an observer. An observer is capable to correct the different states in a model due to a known error. This error is due to the fact that the model does not contain all the dynamics of the real system. As indicated before is the real position information in transmitted wireless so this error is not real-time available. The observer designed capable to process delayed position information can also be found in chapter 6.

3 System adjustments

The pick and place machine is controlled by an embedded computer. This closed system gives no information about the control signals, and therefore offers no options to remote analyse and control the system. Therefore we want to control the pick and place machine from our own control environment dSpace.
This contains feeding both motors with a 3 phase current. For this suitable current amplifiers needs to be implemented. These amplifiers needs to be supplied with 2 current reference signals and external extrenal on/off signal. Finally the incremental encoders needs to be tapped for position information.

3.1 dSpace

The used dSpace board is cp1104. This board has 2 incremental encoders which is sufficient to read the position information of the x- and y-axis. The board has enough analog outputs the supply the reference currents to control the current amplifier. Two digital outputs will be used to enable and disable the current amplifiers. The control algorithms is now capable to . These control algorithms can be easily uploaded on the board's microprocessor. Which gives full control of the pick and place machine and all control signals can be visualized on the screen or be stored on the computer for futher analysis.

3.2 Incremental encoders

For position information the robot is equiped with incremental encoders. For the different axis different encoders are used.

Table 2: *Incremental encoders specification*

Axis	Manufacturer	Resolution	Interface
X	MicroE M1510-40	0.5 μm	Differential RS422
Y	Heidenhain ERO 1470	1.5 μm	Differential RS422

The wireing between the encoders and the processor is done with differential wiring for low distortion levels. Because the used dSpace panel is equipped with the same diffetential RS422 incremental encoder ports, the data signals can be directly tapped from the motor and connected to the dSpace panel.

3.3 Current control

For the current controllers amplifier from the manufacturer Elmo motion control are used. For the x-actuator a FLU-3/100 is used and for the y-actuator a FLU-15/60 is used. The inputs of the amplifier are current references. To follow these current references a built-in control loop is used. The current reference drives a pulse width modulation(PWM) module. This PWM module switches MOSFETS to connect the positive or negative voltage to the

winding. The resulting current is measured and fed back. This closed loop control will regulate the duty cycle of the PWM module such that the measured current is equal to the current reference. The motor is a three phase motor. The current amplifier needs only to be fed with two phase currents. The third phase current is calculated from the first and second current, such that the sum of the currents is zero.

Table 3: *Specification of the current amplifiers*

	Unit	Motor X-axis	FLU-3/100	Motor Y-axis	FLU-16/60
$I_{nominal}$	A_{eff}	0.87	3.3	6.9	15
I_{peak}	A_{eff}	3.1	6.6	21.5	30
$V_{nominal}$	V	36	85	48	49
V_{max}	V	-	100	-	60

The amplifier has mutiple tuning options and is equipped with an Current Gain Control(CGC). A CGC improves the performance of low induction motors. Because both motors are low induction motors this option is enabled. It reduces the proportional gain of the current loop by approximately 70%.
When looking at table 3 the FLU-3/100 looks a bit over dimensioned but that is with premeditation. The amplifier is equipped with a Current Feedback Multiplier(CFM), which multiplies the feedback current by 2 and consequently the followinf changes occur,
- the current gains are devided by 2
- the current monitor is multiplied by 2
- the current limits are devided by 2

Notice that the amplifier is well suited for controlling the x-motor. The settings can be found in table 4

Table 4: *Current amplifier settings*

	FLU-3/100	FLU-16/60
Current Feedback Multiplier (CFM)	YES	NO
Current Gain Control (CGC)	YES	YES

4 Linear permanent magnet motor

4.1 Dynamics

For movement in both directions a permanent magnet motor is used. A previous study from Antoine Verweij about the Control of a permanent magnet linear motor is used [1]. For the x-movement a linear UC-motor from the manufacturer Technotion is used. For the y-movement a rotating MSSI-motor from Wittenstein. Comparing the shape of both motors is totally different. The translator of the linear motor moves with one degree of freedom while the rotating motor is rotating. Still the working principle is the same as illustrated in figure 4.

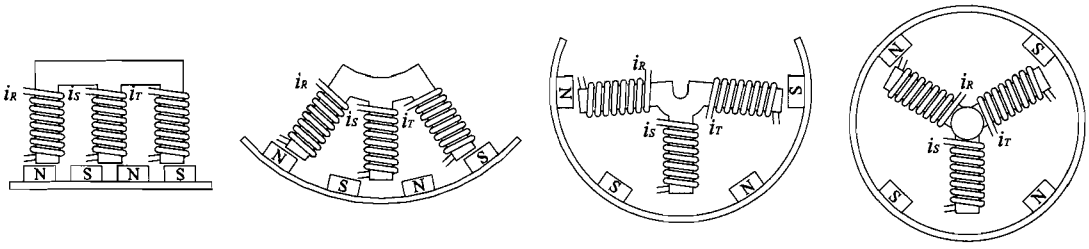


Figure 4: *Transformation from a linear motor to a rotating motor*

Both motors have a stator with permanent magnets and three translators. By sending a current through the windings of the translators a magnetic field is generated. The attracting and repelling forces between the permanent magnets and the generated magnetic field will result in a movement of the translator.
First the relation between the current and the force will be determined. By looking at the voltage over the motor the total power can be calculated. We can separate the mechanical power from the electrical power. We are only interested in the mechanical power, because this results into movement.
The voltage over one winding can be descibed with the following relation,

$$u = i \cdot R + \frac{d\psi}{dt} \; with \; \psi = N\phi \; (coupled \; flux) \tag{1}$$

where u is the voltage over the winding, i is the current through the winding, R is the winding's resistance and ψ is the coupled flux,with,

$$\frac{d\psi}{dt} = \frac{d\psi}{dx} \cdot \frac{dx}{dt} + \frac{d\psi}{di} \cdot \frac{di}{dt} \tag{2a}$$

$$L = \frac{d\psi}{di} \tag{2b}$$

$$\frac{d\psi}{dt} = \frac{d\psi}{dx} \cdot \frac{dx}{dt} + L \cdot \frac{di}{dt} \tag{2c}$$

where L is the inductance.

The total Power P_{total} of one winding can be calculated, with:

$$P_{total} = u \cdot i = i^2 \cdot R + \frac{d\psi}{dx} \cdot \frac{dx}{dt} \cdot i + \frac{1}{2}\frac{dLi^2}{dt} \tag{3}$$

From this equation we can distinguish the mechanical power and the electrical power. Because we are at this time only interested in the mechanical power $P_{mechanical}$ the electrical power will be neglected.

$$P_{mechanical} = \frac{d\psi}{dx} \cdot \frac{dx}{dt} \cdot i = F \cdot v(t) \tag{4a}$$

$$F = \frac{d\psi}{dx} \cdot i \tag{4b}$$

The coupled flux is position dependent with the following relation,

$$\psi(x) = -\hat{\psi} \cdot cos(p(x) - p) \tag{5a}$$

$$i(x) = -\hat{I} \cdot sin(\phi(x) - \phi) \tag{5b}$$

and,

$$p(x) = \phi(x) = \frac{\pi x}{\tau} \tag{6}$$

Were $\phi(x)$ is the current angle which follows the translator displacement, $p(x)$ is the position of the translator, ϕ and p and the unknown current and position offset, and τ is the distance between the permanent poles of the stator.

The force can be calculated,with:

$$F(x) = \frac{\pi}{\tau} \cdot \hat{\psi} \cdot sin(-p(x) - p) \cdot \hat{I} \cdot sin(\phi(x) - \phi) \tag{7a}$$

$$F(x) = \hat{I} \cdot K_{ph}\frac{1}{2}\{cos(-p - \phi) - cos(\frac{2\pi x}{\tau}) - p + \phi\} \; where \; K_{ph} = \frac{\pi}{\tau} \cdot \hat{\psi} \tag{7b}$$

K_{ph} is a constant depending on the geometry and strength of the magnets.
The average force will be,

$$\bar{F}(x) = \frac{1}{2} \cdot K_{ph} \cdot \hat{I} \cdot cos(p + \phi) \tag{8}$$

And the maximum force is reached when $p = -\phi$,which leads to,

$$F_{max}(x) = \frac{1}{2} \cdot K_{ph} \cdot \hat{I} \cdot (1 - cos(\frac{2\pi x}{\tau} + 2p) \tag{9}$$

Now the force for one winding can be calculated. As illustrated in figure 4 the linear motor contains three windings. The total force can be calculated by taking the sum of the three windings (see equation 10a). Keep in mind that the windings are related with a 120^o electrical phase shift with respect to each other.

$$\begin{aligned} F_R &= \hat{I} \cdot K_{ph} \cdot \frac{1}{2} \cdot \left\{ cos\left(-p-\phi\right) - cos\left(\frac{2\pi x}{\tau} - p + \phi\right) \right\} \\ F_S &= \hat{I} \cdot K_{ph} \cdot \frac{1}{2} \cdot \left\{ cos\left(-p-\phi\right) - cos\left(\frac{2\pi x}{\tau} - p + \phi + \frac{2\pi}{3}\right) \right\} \\ F_T &= \hat{I} \cdot K_{ph} \cdot \frac{1}{2} \cdot \left\{ cos\left(-p-\phi\right) - cos\left(\frac{2\pi x}{\tau} - p + \phi + \frac{4\pi}{3}\right) \right\} \end{aligned} \tag{10a}$$

$$F_{total} = K_t \cdot \hat{I} \cdot cos(p+\phi) \tag{10b}$$

As shown in equation 10b the force is dependent on the offset position of the translator p and the offset current angle ϕ. The relation between the force, position and phase is shown in figure 5. As you can see in the worst case the force can be zero. Then the motor will not move.
For optimal control we want to apply the maximum force. Therefor the motor needs to be aligned. The alignment procedure will be explained in section 4.2.

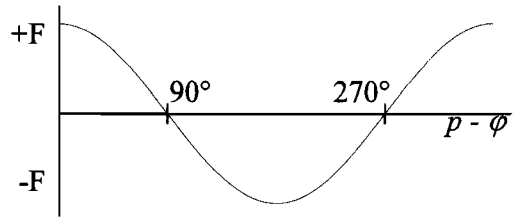


Figure 5: *Force, Position and Phase curve*

4.2 Alignment procedure

In the previous section was shown that when the offset position of the translator (p) minus current angle (ϕ) is $\frac{1}{2}\pi$ or added with a plural from π the force is always zero regardless of the current. To overcome this problem an alignment procedure will be introduced. This procedure will be written in C^{++} and implemented in simulink as an S-function.
The concept in this procedure is that the translator movement will be kept to an absolute minimum. However it is impossible to have no movement at all because the only available indicator of the position comes from the incremental encoders. This is solved by moving the translator in one direction and the next movement in the opposite direction. This way the movement will be kept to a minimum.

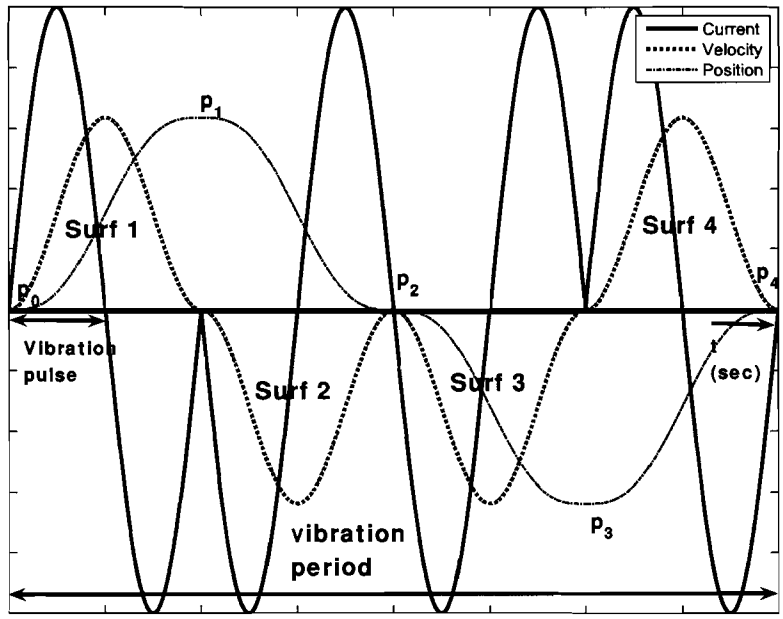


Figure 6: *Movement during one vibration pulse*

This is done with a vibration pulse. Such a pulse is shown in figure 6. During a vibration pulse the translator will move in a positive direction and back and the other way around. The total amount of movement will be determined by summing the four surfaces.
The result of one vibration period is,

$$RESULT = Surf1 - Surf2 - Surf3 + surf4 \tag{11a}$$

$$RESULT = (p_1 - p_0) - (p_2 - p_1) - (p_3 - p_2) - (p_4 - p_3) = 2p_1 - p_0 - 2p_3 + p_4 \tag{11b}$$

Assume that $\hat{I}$ is large enough to overcome the friction then there are three possibilities,

1. $cos(p - \phi) = 0$
   Then
   $p_0 = p_1 = p_2 = p_3 = p_4 = 0 \;\} \; RESULT = 0$

2. $cos(p - \phi) > 0$

$$\left.\begin{array}{l} p_0 = p_0 \\ p_1 = p_0 + \Delta \\ p_2 = p_0 \\ p_3 = p_0 - \Delta \\ p_4 = p_0 \end{array}\right\} RESULT = 2p_0 + 2\Delta - p_0 - 2p_0 + 2\Delta + p_0 = 4\Delta$$

3. $cos(p - \phi) < 0$

$$\left.\begin{array}{l} p_0 = p_0 \\ p_1 = p_0 - \Delta \\ p_2 = p_0 \\ p_3 = p_0 + \Delta \\ p_4 = p_0 \end{array}\right\} RESULT = 2p_0 - 2\Delta - p_0 - 2p_0 - 2\Delta + p_0 = -4\Delta$$

The incremental encoder registers position movement with accuracy of micrometers as shown in table 2. To be sure that the position movement is caused by the applied force and not by current noise or mechanical vibrations a threshold detection level is considered. It means that the absolute movement has to be of a certain level in positive or negative direction. With this knowledge with a high degree of certainty it can be said that the motor has moved due to the applied current.
The alignment procedure makes use of these vibration pulses. When looking at the relation in equation 10b it shows that the total force depends on the offset position p, the current offset angle ϕ and the current amplitude $\hat{I}$. When used the vibration pulses the position p of the translator will practically stay the same. So, by manipulation ϕ the zero crossings in the FPP curve (figure 5) can be found. The alignment procedure is based on this relation. The complete alignment procedure consists of five procedures and will now be explained in more detail.

1. Safety regulation
   During the whole process a couple of factors will be monitored. We make sure that the maximum speed, position and current are not exceeded. When these factors are exceeded the procedure will make an emergency stop. Which means that the currents will be set to zero, the current amplifier will be shutdown, and the controller will be disabled.

2. Test procedure
   The purpose of this procedure is to test if the motor moves. A vibration pulse is send trough the coils with a very small $\hat{I}$ and a ϕ of zero. When no movement is detected it can be that $cos(p - \phi) \approx 0$ or $\hat{I}$ is to small to overcome the friction. So the next vibration pulse ϕ will be increased by 90^o and $\hat{I}$ will be increased by twenty percent. If motor movement is registered during three vibration periods with the same ϕ and $\hat{I}$, the test is completed successfully. Now, the zero search procedure is started. When the test fails the alignment procedure will be aborted.

3. Zero search procedure
   Looking again at equation 10b and figure 5 the total force is related with a cos function. The zero search procedure will find by means of an iterative procedure the point where no force is generated. Now $p + \phi = \frac{1}{2}\pi \pm k\pi$ where k is a real integer. When the ϕ is found the maximal force can be found by increasing or decreasing with $\frac{1}{2}\pi$ to find the maximal positive or negative force. The iteration process will be explained with a numerical example.

Example:

(a) Assume for a certain current angle ϕ,$(p + \phi) = 30^o$. Taking the conditions assumed in figure 7a. The force will be positive so a positive position movement is calculated after one vibration pulse. Now the current angle ϕ will be incremented by $d\phi = 90^o$

(b) So $(p + \phi) = 120^o$. Looking at figure 7b a negative force is generated. Now the new ϕ will be decremented with $d\phi = \frac{1}{2}d\phi = 45^o$.

(c) Now $(p + \phi) = 75^o$. According to figure 7c the force is smaller then the friction, no movement will be detected. Now the $\hat{I}$ will be increased by 20% shown in figure 7d.

(d) During the next vibration period positive movement will be detected. Again a new ϕ will be calculated $d\phi = \frac{1}{2}d\phi = 22.5^o$.

(e) When during the following vibration pulses movement is detected the ϕ will increase or decrease by $\frac{1}{2}\phi$ according the positive or negative position detection. When no movement is detected $\hat{I}$ will be increased till 50% of the maximal motor current is reached.

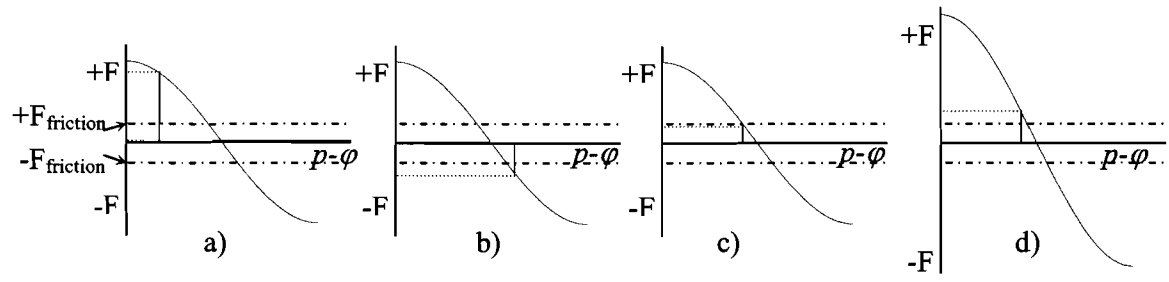


Figure 7: *Zero search*

4. Homing procedure
The final part of the alignment procedure is moving the axis to it's zero position. The zero position is indicated by an index signal or just the minimal position of the total range. A constant amplitude $\hat{I}$ is fed to the linear motors. $\hat{I}$ is chosen to be just large enough to overcome the friction. When the index signal is detected the incremental encoders will be set to zero and a corresponding current angle will be calculated. Now that the relation with the position and current angle is known the robot is ready to be controlled. The homing procedure will go trough the following procedures.

(a) Homing_status = 0
Because it isn't known which zero point at the FPP curve (see figure 5) is found it is unknown in which direction the motor moves when feeding it with a positive current. Obviously it is useful that a positive current relates to a positive movement. So a small vibration pulse is made with a positive current. When a positive movement is detected we can proceed. If a negative movement is detected ϕ will be corrected with 90^o.

(b) Homing_status = 1
Wait until the motor has stopped moving.

(c) Homing_status = 2

i. With an index pulse
The motor will move to the beginning of the axis and the next part of the homing procedure is called.

ii. Without an index pulse
The motor will move to the beginning of the axis and the homing procedure is finished. The incremental encoders will be set to zero and the corresponding ϕ will be calculated.

(d) Homing_status = 3
The motor will move very slowly to the other side of the axis. When the index pulse is detected the Homing_status is set to 4.

(e) Homing_status = 4
Wait until the motor has stopped moving.

(f) Homing_status = 5
The motor will move as slow as possible to find the index pulse again. When the pulse is detected the incremental encoders will be set to zero and the corresponding ϕ will be calculated.

5. Ready procedure
When the alignment is successfully completed the software will come in its ready state and will stay in this routine. It keeps on controlling the safety regulation and will send an enable signal to the controller.

The alignment code can be found in the appendix A.

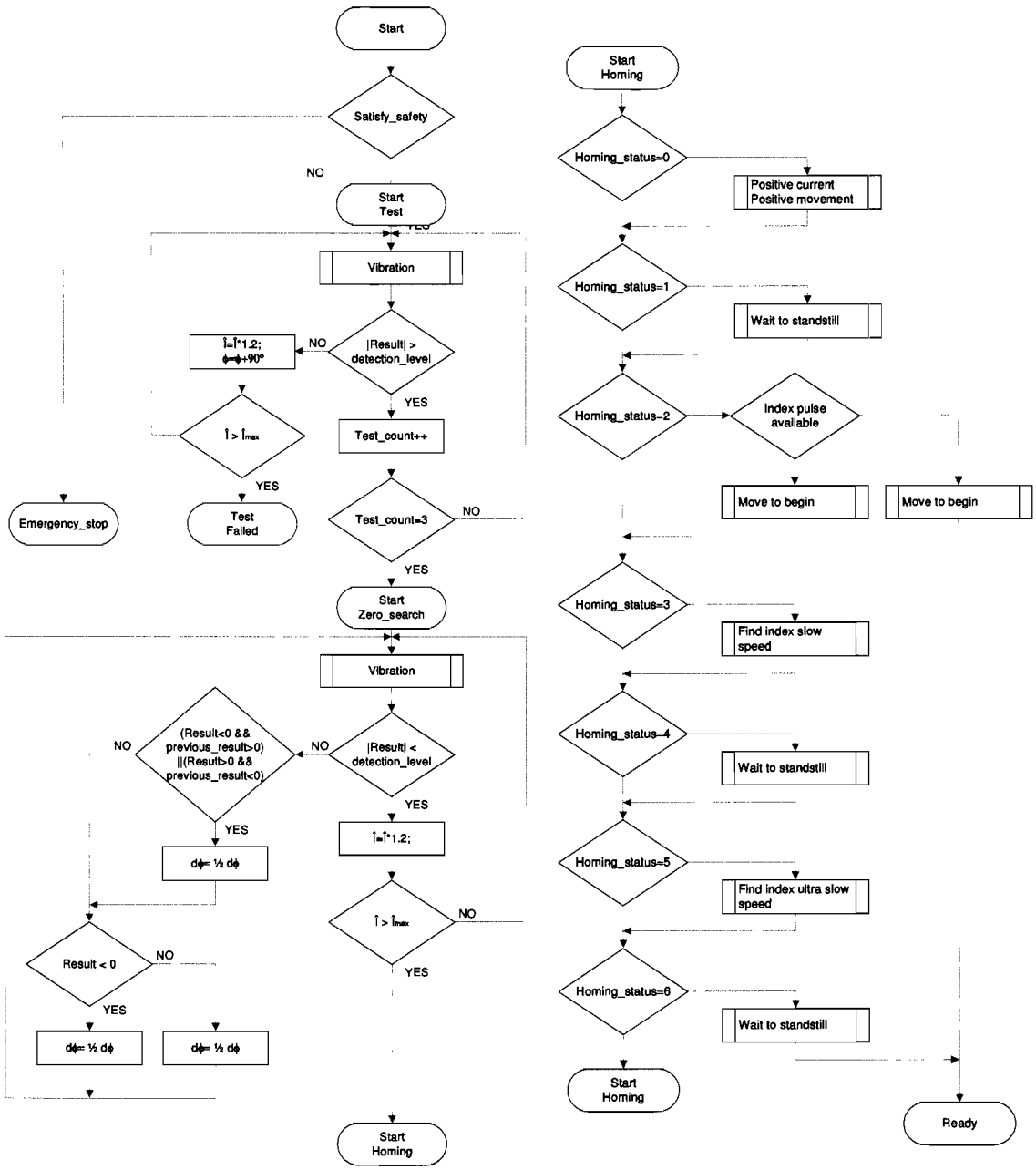


Figure 8: *Flowchart alignment procedure*

4.3 Modeling

In this paragraph the models of the motors will be made. This models will be used for simulation and later on for the estimation. For modeling a the motors a first order principle model will be made. This is done with the known physical relations of the dynamics of the motor. We are interested in the relation between the current and the movement. So this will be modeled.
The linear motor used for the x-axis can be seen as a moving mass due to a certain force. Note that the motor mass m experiences friction D. We get,

$$F - Dv = ma \tag{12}$$

Where v is the velocity and a the acceleration.

$$F - D\dot{x} = m\ddot{x} \tag{13}$$

Where x is the position.

The relation between the force and the current can be found in equation 10b. The relation between the current and the position is given by,

$$I = \frac{D}{K_t} \cdot \dot{x} + \frac{m}{K_t} \cdot \ddot{x} \tag{14}$$

The transfer function for the model is,

$$P(s) = \frac{x}{I} = \frac{K_t}{m \cdot s^2 + D \cdot s} \tag{15}$$

The y-axis uses a rotating motor. This can be modeled the same way. The angular force that cause rotation is called torque(T). The rotating mass will be modeled with moment of inertia(J). Which can be seen as the mass in basic dynamics. Here also a the rotation experiences friction(D). So the following physical relations will hold.

$$T - D\omega = J\alpha \tag{16}$$

Where ω is the angular velocity and α is the angular acceleration.

$$T - D\dot{\theta} = J\ddot{\theta} \tag{17}$$

Where θ is the angular position.

This rotation is converted of a linear movement due to a spindle. One rotation results in a movement of 3 centimeter. This is called the gear ratio (K_{gr}). So the gear ratio will

be $\frac{0.03}{2\pi}$.
So the transfer function of the y motor will be,

$$P(s) = \frac{x}{I} = \frac{K_t K_{gr}}{J \cdot s^2 + D \cdot s} \tag{18}$$

For simulation the D is replaced with a coulomb & viscous block. Coulomb friction represents the offset friction and viscous represents the friction of the mass. This way a more accurate model is realized. Also a quantizer is added to simulate the behavior of an incremental encoder and a saturation block is used to constrain the position. The speed is calculated to determent the covered distance in one sample time. This results in the following models. The motor for the x-axis can be found in figure 9 and the motor model for the y-axis in figure 10.

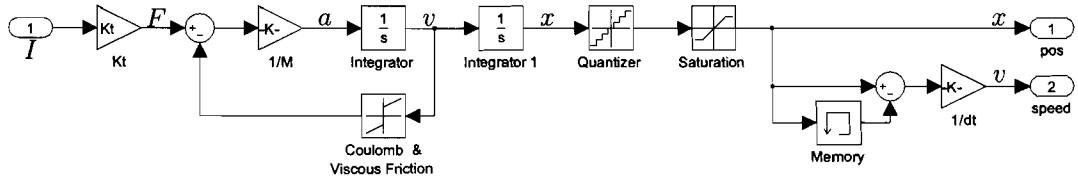


Figure 9: *Simulation model of a motor X-axis*

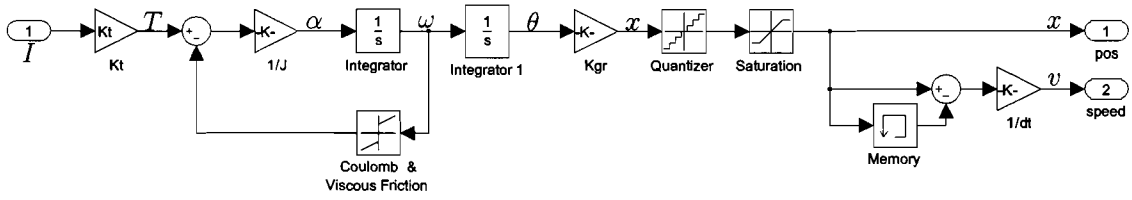
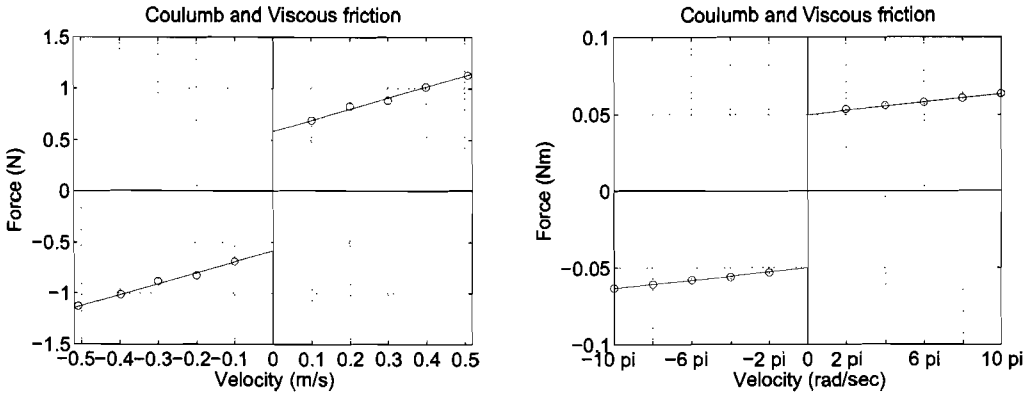


Figure 10: *Simulation model of a motor Y-axis*

4.4 Coulomb and viscous friction

The coulomb and viscous friction can be determined by driving the motor at different speeds. At every speed the used force is determined. From the linear relation between the speed and the force the viscous friction can be determined. The force at a velocity of zero is the coulomb friction. This is also called the offset friction. The measurements for determining the coulomb and viscous friction from the x-motor and the y-motor are shown in figure 11. The values are shown in table 5.



(a) Coulomb and viscous friction X-axis

(b) Coulomb and viscous friction Y-axis

Figure 11: *Coulomb and viscous friction*

4.5 Simulation alignment procedure

With the model as described in chapter 4.3 the alignment procedure can be simulated. The simulation is done in simulink. The simulink file that is used is shown in figure 12. The alignment from the x-axis and the y-axis doesn't differ too much in amount with the procedure of the x-axis. Therefore only the y-axis alignment procedure will be explained. The alignment procedure is implemented in simulink as an S-function. This is shown in figure 12 as the block called *alignmentY*. This block has four inputs. Looking form top to bottom the first two are the measured position and the calculated speed. Also the index signal, when available, must be known for the homing procedure. The last one is the starting signal for alignment procedure. The alignment procedure has 8 outputs. The first two outputs are the calculated phase currents depending of ϕ.
This will be calculated by equation 5b. The third phase current will be externally calculated by the current amplifier. For simulation this is added by the model. Besides generating the phase current, the alignment procedure is also responsible for resetting the incremental encoder. It is capable of enabling and disabling the current amplifier and the controller. The output *StatusY* shows in witch routine the alignment procedure is currently running. The output *debugY* is for debugging purposes. This way a variable can be observed during simulation. The last output is the determined ϕ in relation with the unknown position.

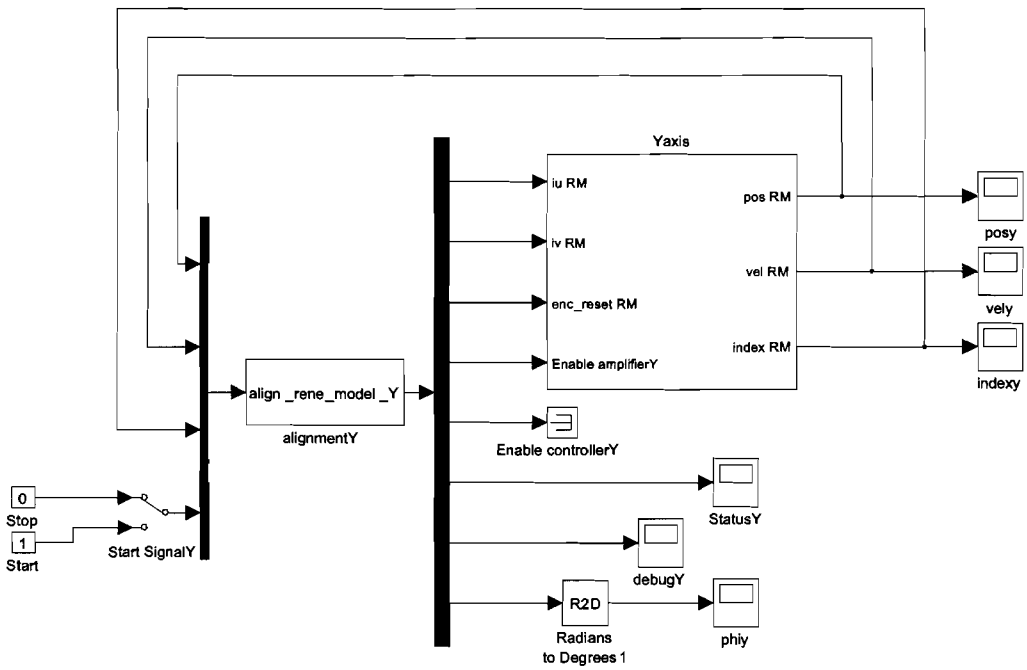


Figure 12: *Alignment procedure scheme in simulink*

Now we look at the sub block *'Yaxis'* shown in figure 13. As input we have 2 phase current depending on ϕ from the alignment procedure. The relation between the force and the current is shown in equation 7a. This relation is used to determent the generated force. This is depicted in figure 13. So the relation between current and force is implemented. Also is shown how the third phase current is calculated. The relation between force and position is determined in subsection 4.3. The found relation for the used axis will be added in sub block *'Force − position'*. These relation can be found in equation 15 and 18 and are depicted in figure 9 and 10.

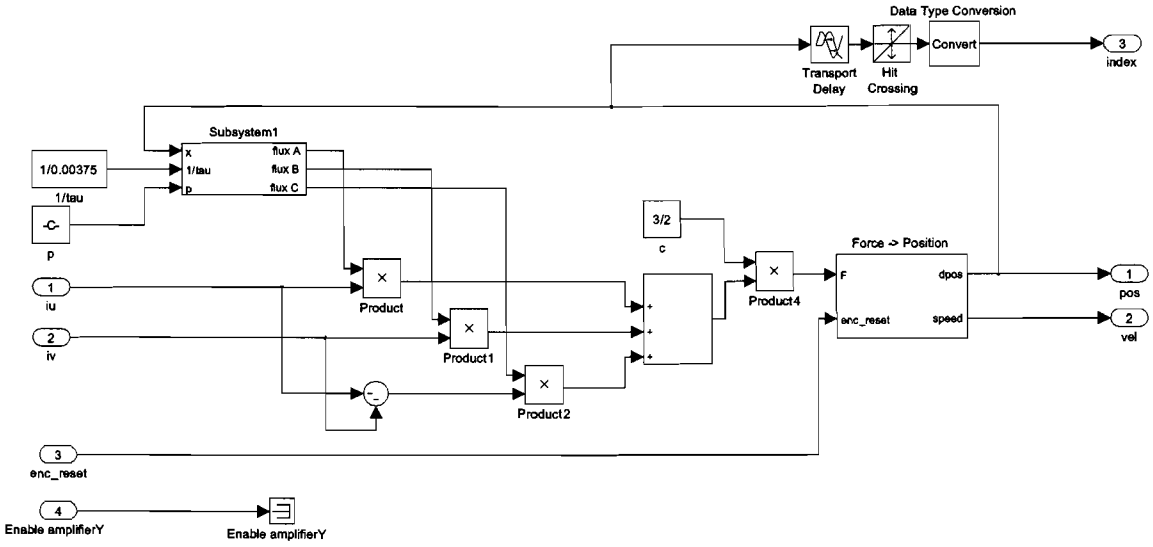
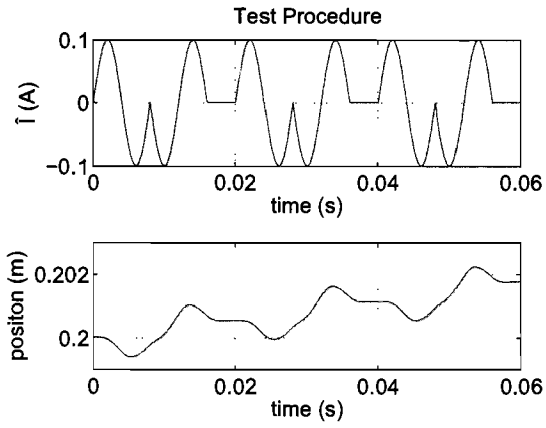


Figure 13: *Model Y-axis*

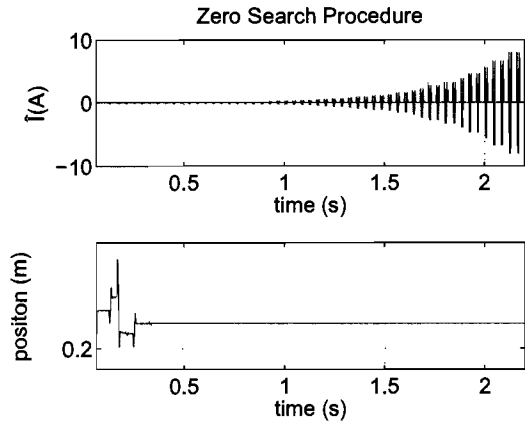
The used parameters for simulation can be found in table 5

Table 5: *Model parameters*

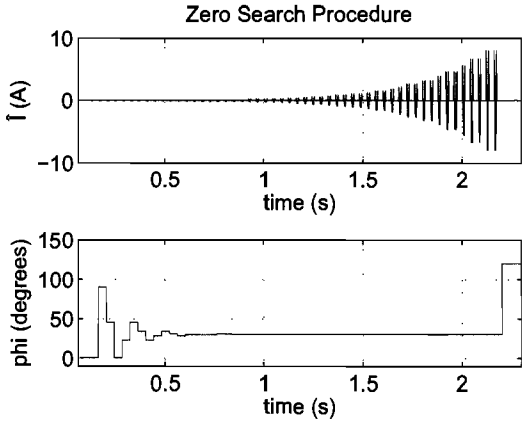
Model parameters	Y	X
Motor constant (K_t)	0.11 Nm/A_{eff}	11.4 N/A_{eff}
Gear Ratio (K_{gr})	$\frac{0.03}{2\pi}$ m/rad	-
Moment of inertia (J)	$5.0^{-5} kgm^2$	-
Mass (m)	-	0.6 kg
Tau (τ)	3.75 $^{-3}$ m	8.25 $^{-3}$ m
Viscous friction	4.0 $^{-3}$ Nms/rad	0.6 Ns/m
Coulomb friction (D)	5.0 $^{-2}$ Nm	1.1 N



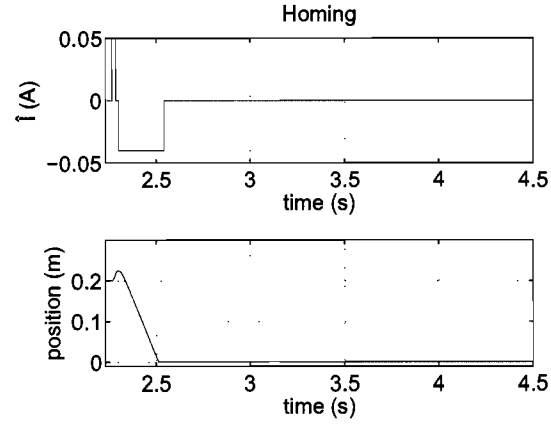
(a) Test pulses



(b) Zero search procedure



(c) Zero search procedure



(d) Homing

Figure 14: *Alignment results*

In the subfigures of figure 14 the simulation result are depicted. First we see in figure 14(a) three vibration pulses which will detect if the motor actually moves. When three movements are detected above the threshold the test is completed. Then the zero search procedure start. In this case the maximal current of the motor is 20 A. For this procedure the maximum current is restricted up to 8 A. When 8 ampere is applied and no movement is detected the procedure is finished. In figure 14(b) you can see that the current increases and the movement decreases. Till $\phi = p$. In figure 14(c) the corresponding ϕ is plotted. Here the iteration process is nicely displayed. At the end of the zero search you see that ϕ is incremented with 90^o. So the maximum force will be used. From the final picture 14(d) can be concluded that the alignment procedure is finished successfully. A constant $\hat{I}$ will result in a linear movement.

4.6 Implementation dSpace

To connect the PC with the pick and place machine the same simulink file is used as shown in figure 12. Only the model in sub-block *Yaxis* will now be replaced with the real system. To communicate with one of the axes 4 ports of the dSpace I/O system are used. This is shown in figure 15. These are 2 digital to analog convertors (DAC), one pin of the digital I/O port and the incremental encoder port.

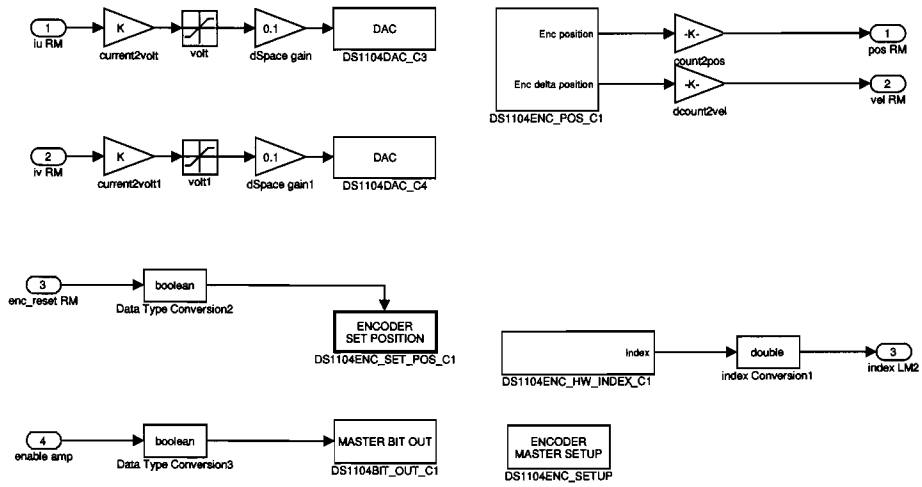


Figure 15: *dSpace interface to one axis*

First we will explain the two DAC's. The current amplifier needs two reference signals to generate the three phase current. For this the two DAC's are used. In front of both DAC's we see a *current2volt* gain, a block called *volt* and a *dSpace* gain. The *current2volt* gain will scale the desired current to the right voltage reference. The current amplifiers work with a 0 to 10 Volt current reference. So in case of the x-axis 10 Volt must represent the maximal peak current of 3.3 Ampere. So the *current2volt* gain will be $\frac{10}{3.3}$. The peak current of the y-axis is 11 Ampere so here the gain will be $\frac{10}{11}$. The blocks *volt* will limit the current.
Further has dSpace one peculiarity. If simulink sends a reference of 1 Volt to the DAC, the dSpace I/O panel will send out 10 Volt. To fix this the dSpace gain is introduced.
The third input is the incremental encoder reset signal. This is a part of the incremental encoder port. Because the dSapce panel can't process a variable other then a boolean the variable from the S-function needs to be converted to a boolean. When this signal is high the incremental encoder will be reset to zero. The final input is to enable the amplifier. This variable is for the same reason converted to a boolean and sets one pin of the digital I/O connector. This will enable the current amplifier.

One of the outputs is the position. The position will be determent by the available incremental encoder on the I/O panel. The incremental encoder contains a counter that will be incremented or decremented by one with a specified accuracy. For the x-axis one count represents $0.5\mu m$ and for the y-axis $1.5\mu m$. The *count2pos* gain will convert the counted position movement to the corresponding movement expressed in meters. The gain *dcount2vel* will calculates the speed by determining the movement in one sample. Output number three represents the index signal. This signal will be high at some point of the axis. This way the actual position can be determent. This boolean will be converted to a double so it can be read by the S-function during the alignment procedure.

5 Controller design

In this chapter we will go through the design of the controller. For both motors a different controller will be designed. We clarify the way the the controllers are designed and show the transfer function of the controllers. Also the accuracy of the controls will be tested. The results of controlling the system models and the actual plant can be found in this chapter.
For designing the controllers a frequency response design method is used. First an appropriate bandwidth of the total transfer will be determined. This we will call B_c and the open loop magnitude at this frequency must be 0 dB.
The bandwidth will be determined by looking at the bode plots supplied by Assembléon. These are results from a series of tests Assembléon has performed on their pick and place machine. The results can be found in appendix B.
In the plots the resonance frequencies of the system are exposed. From this we can derive till which frequency the plants can be controlled. It shows that the controller for the x-axis has a clean response up to a frequency 50 Hz and the controller for the y-axis has a clean response up to a frequency of 100 Hz.
The machine that we will use during this experiment is only one arm of a pick and place machine. This arm stands up side down and is not nailed down. That's why a less high frequency controller will be designed. The controlled closed loop system of the x-axis will be designed with a bandwidth up to 30 Hz. The closed loop controlled system of the y-axis with a bandwidth up to 70 Hz. The bandwidths and the corresponding sample times can be found in table 6.
First a phase lead compensator will be designed around the bandwidth. This will improve its phase margin and will increase stability. A rule of thumb advices to place the pole at three times and the zero at one third of the bandwidth(B_c). A roll-off pole will be added, which makes the system insensitive for higher frequencies than the desired bandwidth(B_c). This pole will be placed far away so it will not interfere with the lead filter. Here the rule of thumb advice to place the pole at six times the bandwidth(B_c). To guarantee that the final error and the steady state error will be zero an integrator is added. Finally an additional zero will be added symmetrically around the lead filter, to prevent interference and the lead filter. This zero will be placed a sixth times the bandwidth(B_c). So the different parts of the controller are shown in equations 19, 20 and 21.

$$H_{pl}(s) = \frac{s - z_1}{s - p_1} \tag{19}$$

$$H_{roleoff}(s) = \frac{1}{s - p_2} \tag{20}$$

$$H_i(s) = \frac{s - z_2}{s} \tag{21}$$

This leads to the following transfer function of the controller, designed,

$$H_{controller}(s) = K_c \cdot H_{pl} \cdot H_{roleoff} \cdot H_i = K_c \frac{(s - z_1) \cdot (s - z_2)}{s \cdot (s - p_1) \cdot (s - p_2)} \tag{22}$$

The pole and zero for the lead filter will be placed around the bandwidth (B_c) with the relation

$$z_1 = \frac{B_c}{3} \tag{23a}$$

$$p_1 = B_c \cdot 3 \tag{23b}$$

The roll-off pole has the following relation with the bandwidth (B_c) and zero will be placed symmetrically

$$p_2 = B_c \cdot 6 \tag{24a}$$

$$z_2 = \frac{B_c}{6} \tag{24b}$$

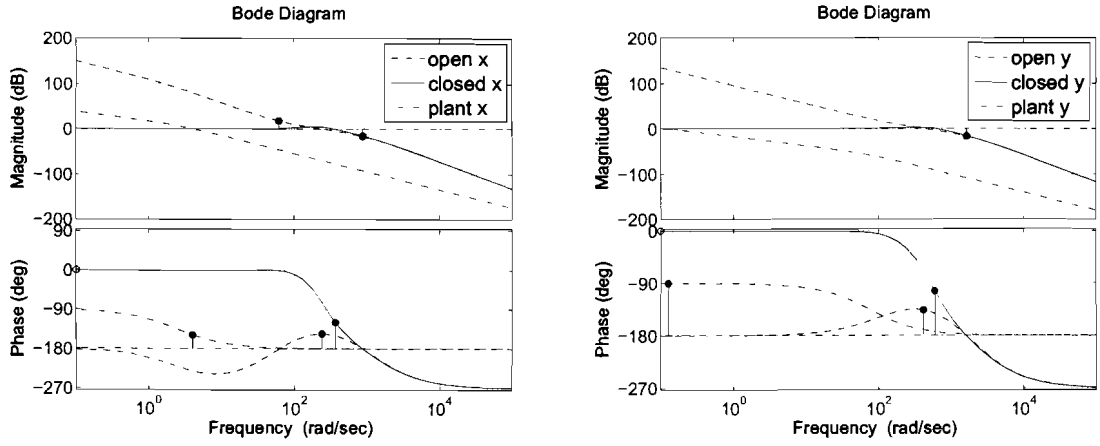
K_c will be chosen in such a way that the open loop magnitude is 0 dB at the $\omega = B_c$.

The total transfer function of the controller will have the form,

$$H_c(s) = K_c \frac{s^2 + \frac{B_c}{2} \cdot s + \frac{B_c^2}{18}}{s^3 + 9 \cdot B_c \cdot s^2 + 18 \cdot B_c^2 \cdot s} \tag{25}$$

Table 6: *Bandwidth*

Motor axis	Bandwidth (B_c)	K_c	Appropriate T_s
Y	420 $\frac{rad}{sec}$	$6.6 \cdot 10^5$	$\approx \frac{1}{5 \cdot B_c} = 3 \cdot 10^{-3}$
X	240 $\frac{rad}{sec}$	$1.4 \cdot 10^8$	$\approx \frac{1}{5 \cdot B_c} = 5 \cdot 10^{-3}$



(a) Bode plots X-axis

(b) Bode plots Y-axis

Figure 16: *Bode plots controllers*

The plots show a clean closed-loop response of 0 dB up to the desired bandwidth. For higher frequencies the gain will decrease so the system will be insensitive for higher frequencies. The the controlled closed-loop system of the x-axis has a phase marge of 61^o and the controlled closed-loop system of the y-axis has a phase marge of 77^o, which is satisfying.

5.1 Position en Speed Trajectory

Assembléon will use a trajectory profile to move the axes to the specified position. For this study a third order reference trajectory design tool for Matlab/Simulink/RTW R11/R12 is used. This tool is developed by dr.ir. M.J.G. van de Molengraft (René).
With this tool it is possible to generate position profiles with a desired speed, acceleration and jerk. Also it is possible to give in a start time. This is suitable for synchronization purposes.

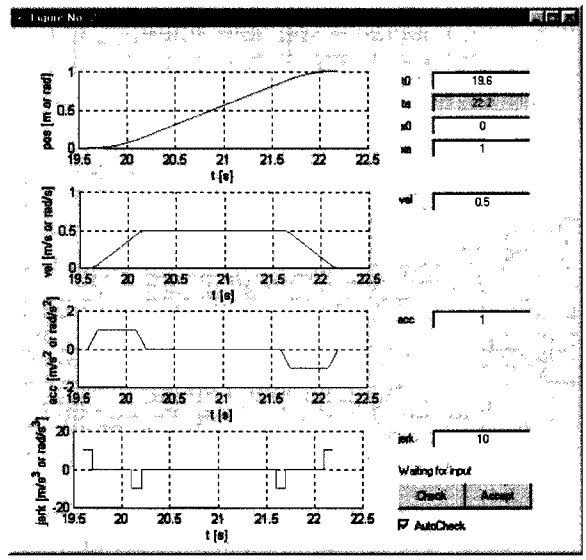


Figure 17: *Trajectory profile generator*

5.2 Simulation of the controllers

The total simulink simulation scheme is shown in figure 18. This scheme doesn't differ much from the simulink scheme from figure 12, which was used for simulating the alignment procedure. Comparison shown that in figure 18 two blocks are added: block *'ControllerX'* and block *'Interface'*. In this section these blocks will be further explained.

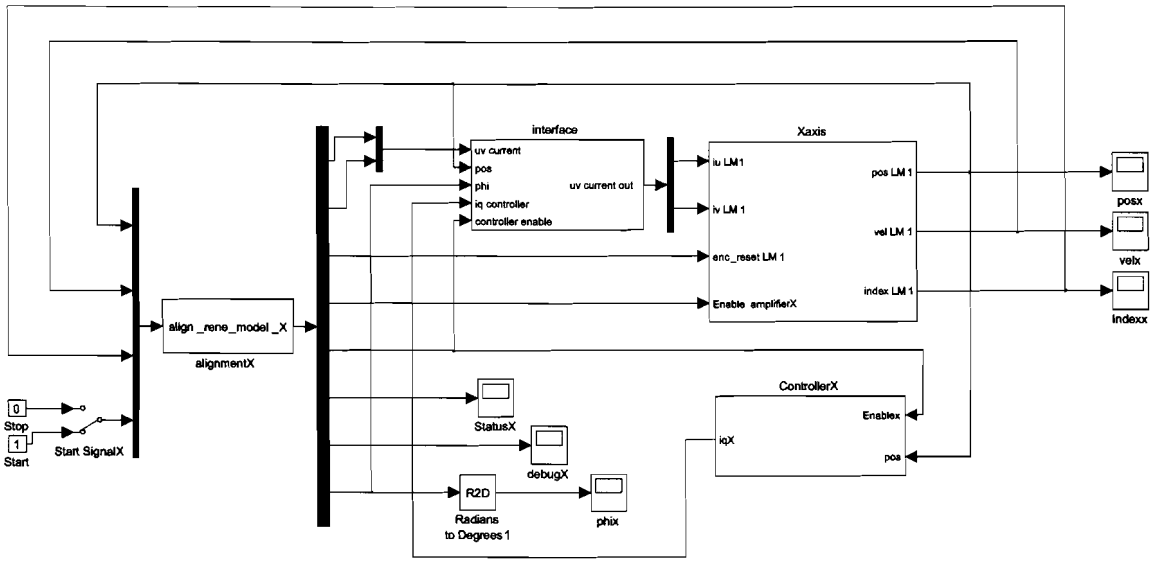


Figure 18: *Simulink simulation file with controller*

The first block that is added contains the controller and is called *'ControllerX'*. The designed controller found in equation 25 and can be worked out to the following relation,

$$y = -9B_c\frac{1}{s}y - 18B_c^2\frac{1}{s^2}y + K_c(\frac{1}{s}u + \frac{B}{2}\frac{1}{s^2}u + \frac{B^2}{18}\frac{1}{s^3}u) \tag{26}$$

This representation is implemented in simulink and is shown in figure 19.

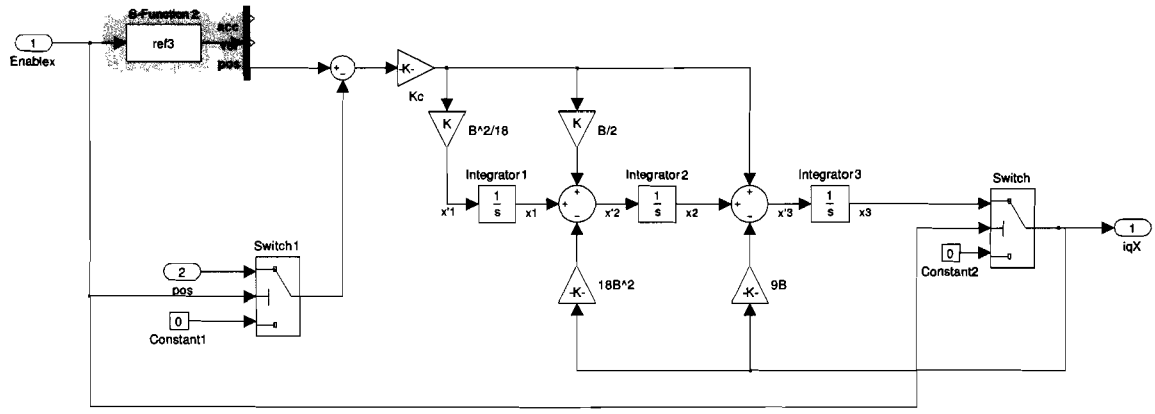


Figure 19: *Controller in simulink*

The block *'Ref3'* is responsible for the position profile. The position profile will start when the alignment procedure is finished and the controller is enabled. The position stays not zero during the alignment procedure. To prevent unwanted turn-on transients the states of the controller will be kept zero due to the two switches.

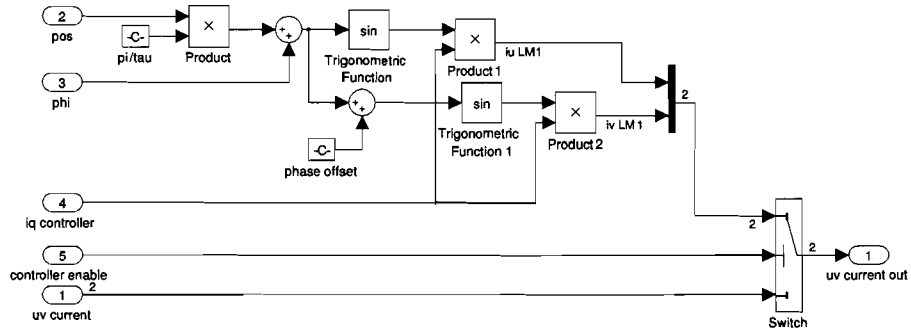
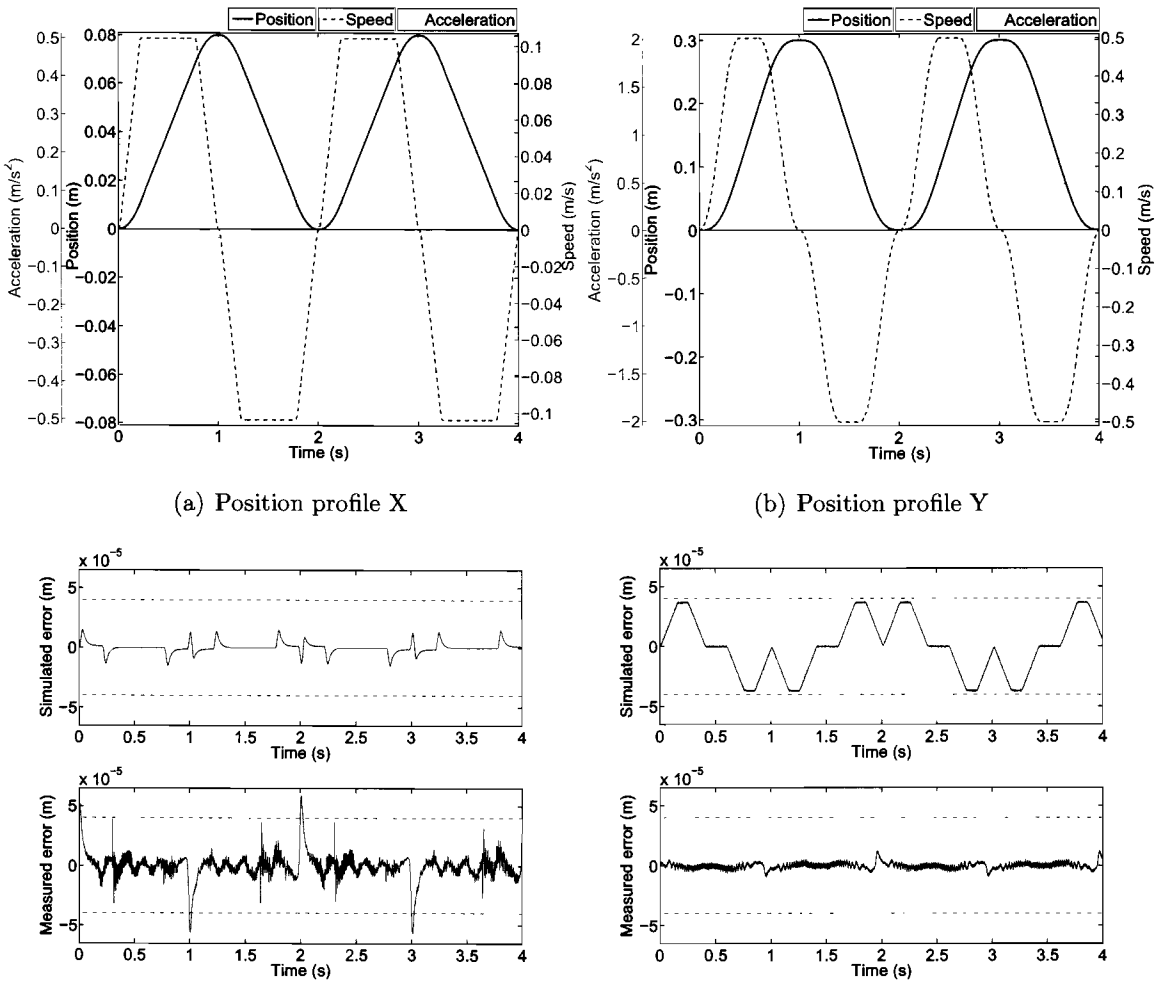


Figure 20: *Sub-block interface*

The second block that is added is called *'interface'*. This block is shown in figure 20. Notice the output currents are dependent of the controller enable signal. If the controller is not enabled, the currents from the alignment procedure are used. As indicated before the alignment procedure generates the $\hat{I}$ with the corresponding flux depending on ϕ. The controller only generates $\hat{I}$. When the controller is enabled this part must be added to the current. This is done is sub-block *'interface'*. Here is shown how the corresponding coupled flux depending on the determined ϕ and position is added to the current.

Figure 21 shows the results of the simulation. For both systems a different position profile is generated. The simulated error is determined with the model described in subsection 4.3. We see that the overall error stays within the $40\mu m$ margin. The $40\mu m$ margin is shown as the dotted line.
We see that for the y-axis the overall error stays within $40\mu m$. This is the accuracy Assembléon guarantees on their site. For the x-axis we see a small peak above this border. This occurs by acceleration en is due to the viscous friction. These worse results can also be imputed to the difference in bandwidth of both controllers.



(a) Position profile X

(b) Position profile Y

(c) Position error Controller X

(d) Position error Controller Y

Figure 21: *Simulation results controller*

6 Design an observer compatible delayed position information

Before we can designing a good observer we need a good model of the system. In the first subsection of this chapter will describe how this model is found. In subsection 6.2 we see how an observer can be designed capable to deal with delayed position information. In the following subsections shown 8 observer's, 4 for both axes. Each capable to handle different delay's in the position information.
Further the stability of the observer will be analyzed. Finally you can find some results of the accuracy of the observers in practice and it's sensitivity for noise.

6.1 Full order estimator

The full order estimator contains a model of the motor with the corresponding controller. With this a representation of the dynamics of the overall plant will be made, which is used to make a good estimation of the system states can be made. The system will be described in the following notation,

$$\dot{x} = \mathbf{A} \cdot x + \mathbf{B} \cdot u \tag{27}$$

$$y = \mathbf{C} \cdot x \tag{28}$$

With $\mathbf{A} = n \times n$ $\mathbf{B} = n \times m$ $\mathbf{C} = m \times n$

The transfer function of the model of the motor was described in equation 15. The transfer function of the controller is given in equation 25. When the two systems are combined the following scheme is obtained.
Input(u) will be supply with the position profile and the output(y) will be the position estimation.

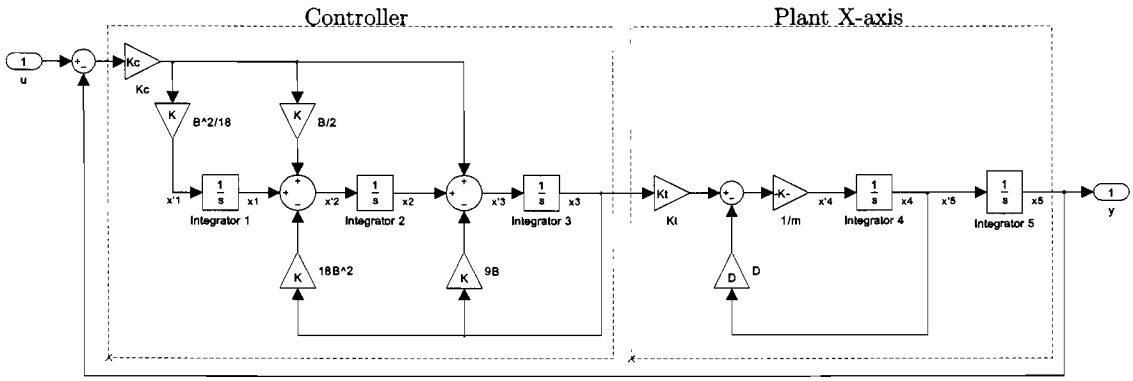


Figure 22: *Combining Controller en Model*

The theorems that are used can be found in the articles [2, 3].
The observer equation can be described as,

$$\dot{\hat{x}}(t) = \mathbf{A}\hat{x}(t) + \mathbf{B}u(t) + \mathbf{K}(y(t - rT_s) - \hat{y}(t - rT_s)) \tag{33}$$

With $\mathbf{K} = n \times m$ and r is a positive integer representing the delay expressed in number of samples.

This can be written to,

$$\dot{\hat{x}}(t) = \mathbf{A}\hat{x}(t) + \mathbf{B}u(t) + \mathbf{KC}(x(t - rT_s) - \hat{x}(t - rT_s)) \tag{34}$$

Assume that the real system is equivalent to the model then the error between the real states and the states of the observer can be defined as,

$$\xi(t) = x(t) - \hat{x}(t) \tag{35}$$

The derivative of the error can be described as,

$$\dot{\xi}(t) = \dot{x}(t) - \dot{\hat{x}}(t) = \mathbf{A}\xi(t) - \mathbf{KC}\xi(t - rT_s)) \tag{36}$$

where the delayed error will be available from impulsive samples subjected to a delay of r samples [37],

$$\xi(t - rT_s) = \sum_{k=-r}^{\infty} \xi(t - rT_s)\delta(t - kT_s - rT_s) \tag{37}$$

which will result in the general solution,

$$\xi(t) = e^{\mathbf{A}T_s}\xi_0 - \sum_{k=-r}^{\left\lceil \frac{t}{T_s} - r \right\rceil} e^{\mathbf{A}(t - kT_s - rT_s)}\mathbf{KC}\xi(kT_s) \tag{38}$$

where,

$\xi_{-r} = \xi_{-r+1} = \ldots = \xi_{-1} = \xi_0$
ξ_0 = initial value of the error
$\left\lceil \frac{t}{T_s} \right\rceil$ = largest integer nummer less then $\frac{t}{T_s}$

It can be shown that,just before the sample moment, the observer error satisfied the following relation,

$$\xi((i+1)T_s^-) = e^{\mathbf{A}T_s}\xi(iT_s) - e^{\mathbf{A}T_s^-}\mathbf{KC}\xi(i - r)T_s^-) \tag{40}$$

Introducing the next variables,

$$w_j(i) = \xi_j((i - r + j - 1)T_s^-), \quad j = 1, 2, 3, \ldots, r + 1 \tag{41}$$

which can be captured in the following matrix,

$$\begin{bmatrix} w_1(i+1) \\ w_2(i+1) \\ \vdots \\ w_r(i+1) \\ w_{r+1}(i+1) \end{bmatrix} = \begin{bmatrix} 0 & I & 0 & \ldots & 0 \\ 0 & 0 & I & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & \ldots & I \\ -e^{AT_s}KC & 0 & 0 & \ldots & e^{AT_s} \end{bmatrix} \cdot \begin{bmatrix} w_1(i) \\ w_2(i) \\ \vdots \\ w_r(i) \\ w_{r+1}(i) \end{bmatrix} \tag{42}$$

Where K is a vector with the dimension $n \times m$.

This system is depicted in figure 25,

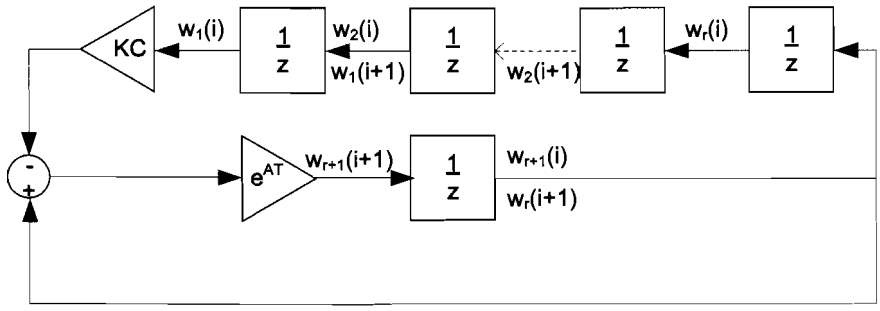


Figure 25: *Error system observer*

Now the following relation is obtained,

$$w(i+1) = \mathbf{H} \cdot w(i) \tag{43}$$

Now we have a mathematical solution of the problem. To guarantee a stable observer the eigenvalues **H** must lie in the unit circle.

To find the relationship between **K** and the poles of the system, the characteristic equation will be studied.
The characteristic equation of the error is given by,

$$P_d = det[\lambda I - \mathbf{H}] = 0 \tag{44}$$

$$P_d = det[\lambda^{r+1} I - \lambda^r e^{\mathbf{A}T_s} + e^{\mathbf{A}T_s}\mathbf{KC}] = \sum_{j=0}^{N} \gamma_j \lambda^{N-j} = 0 \tag{45}$$

Where N = n(r+1)

This can be rewritten as,

$$P_d = det[\lambda^r(\lambda I - e^{\mathbf{A}T_s})] \cdot det[I + \frac{e^{\mathbf{A}T_s} \cdot \mathbf{KC}}{\lambda^- r(\lambda I - e^{\mathbf{A}T_s})}] \tag{46}$$

We assume that **K** is of rank one. Following an approach similar to that of Gopinath [4], 46 can be simplified to,

$$P_d = \lambda^{nr} P + tr\left(\frac{\lambda^{nr-r} \cdot P \cdot e^{\mathbf{A}T_s}\mathbf{KC}}{\lambda I - e^{\mathbf{A}T_s}}\right) \tag{47}$$

Where tr denoted the trace of the matrix, and,

$$P = det[\lambda I - e^{\mathbf{A}T_s}] = \sum_{i=0}^{n} a_i \lambda^{n-i} \tag{48}$$

is the characteristic polynomial of the observer dynamics shown in equation 27 and 28, Noting that,

$$(\lambda I - e^{\mathbf{A}T_s})^{-l} = \sum_{l=0}^{\infty} (e^{\mathbf{A}T_s})^l \lambda^{-(l+i)} \tag{49}$$

outside an appropriate region in the complex plane. And using the Calley-Hamilton theorem [5], we equate of λ^{N-p} on both sides of 47 and we get

$$\gamma_p = a_p \qquad 0 \leqslant p \leqslant r \tag{50a}$$

$$\gamma_p = a_p + tr\left[\sum_{k=0}\sum_{l=0} a_k (e^{AT_s})^l Q\right]_{k+l+1=p-r} \qquad r < p \leqslant n+r \tag{50b}$$

$$\gamma_p = 0 \qquad n+r < p \leqslant N \tag{50c}$$

Where $Q = e^{AT_s}KC$ We want to find the values of **K** so this system is stable and the error will decay to zero. The more delays the more coefficients of gamma will be fixed. Where γ are the coefficients of the characteristic equation of 45 and a_k are the coefficients of the characteristic equation of the observer system described in equation 27 and 28. This only holds when **K** is of rank one. Also notice that the more delays the slower the system will be.

6.3 Poles of the Observer

6.3.1 Observer with No Delay

First an observer is designed with real time position information. The poles of the error system can be placed free in the unit circle. The poles will be placed as 10 times the bandwidth as proposed in [5]. So five poles will be placed according,

$$H_{desired} = \frac{1}{(z - e^{-10*B*T_s})^5} \tag{51}$$

The γ's of the error system can now determined from $H_{desired}$ and relates as follows with the system,

$$\gamma_0 = a_0 \tag{52a}$$
$$\gamma_1 = a_1 + tr\left[a_0(e^{\mathbf{A}T_s})^0 Q\right] \tag{52b}$$
$$\gamma_2 = a_2 + tr\left[a_0(e^{\mathbf{A}T_s})^1 Q + a_1(e^{\mathbf{A}T_s})^0 Q\right] \tag{52c}$$
$$\gamma_3 = a_3 + tr\left[a_0(e^{\mathbf{A}T_s})^2 Q + a_1(e^{\mathbf{A}T_s})^1 Q + a_2(e^{\mathbf{A}T_s})^0 Q\right] \tag{52d}$$
$$\gamma_4 = a_4 + tr\left[a_0(e^{\mathbf{A}T_s})^3 Q + a_1(e^{\mathbf{A}T_s})^2 Q + a_2(e^{\mathbf{A}T_s})^1 Q + a_3(e^{\mathbf{A}T_s})^0 Q\right] \tag{52e}$$
$$\gamma_5 = a_5 + tr\left[a_0(e^{\mathbf{A}T_s})^4 Q + a_1(e^{\mathbf{A}T_s})^3 Q + a_2(e^{\mathbf{A}T_s})^2 Q + a_3(e^{\mathbf{A}T_s})^1 Q + a_4(e^{\mathbf{A}T_s})^0 Q\right] \tag{52f}$$

Because $Q = e^{AT_s}KC$ the K can be determined. After some calculation the K can be found. For both axes they can be found in table 7.

Table 7: *Found K's for zero delay*

X axis		Y axis	
$\mathbf{K_x} =$	$\begin{bmatrix} 7.2 \cdot 10^9 \\ 1.8 \cdot 10^7 \\ 2.9 \cdot 10^3 \\ 4.6 \cdot 10^2 \\ 1 \cdot 10^0 \end{bmatrix}$	$\mathbf{K_y} =$	$\begin{bmatrix} 4.9 \cdot 10^{10} \\ 1.2 \cdot 10^8 \\ -4.3 \cdot 10^3 \\ 6.2 \cdot 10^4 \\ 2.1 \cdot 10^2 \end{bmatrix}$

It is also possible to place them on the imaginary-axis but that wasn't improving its performance. Almost a dead beat system is realized. This means that the error converge to zero in one sample time.

6.3.2 Observer with One Sample Delay

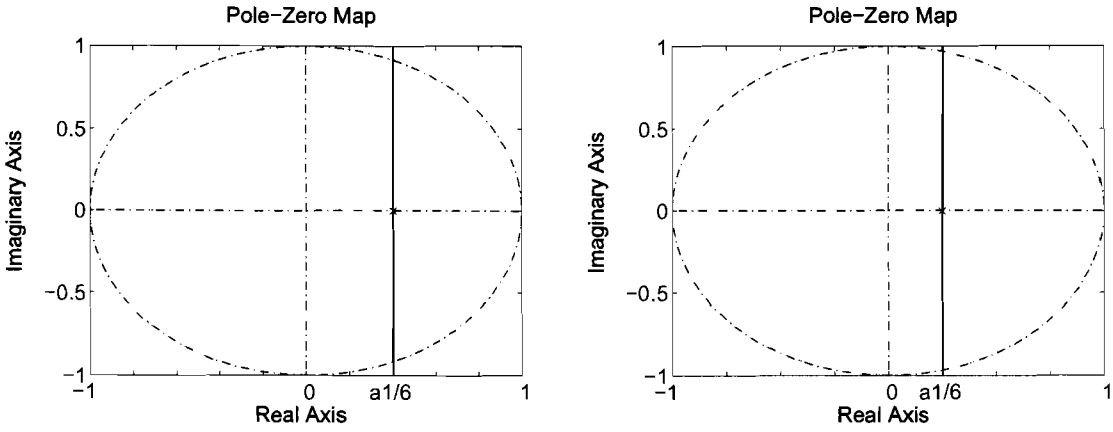
In this section the gain K of an observer will be determined in case the position information undergoes a delay of one sample. First we work out the found algorithm from sub-equation from 50.

$$\gamma_0 = a_0 \tag{53a}$$
$$\gamma_1 = a_1 \tag{53b}$$
$$\gamma_2 = a_2 + tr\left[a_0(e^{\mathbf{A}T_s})^0 Q\right] \tag{53c}$$
$$\gamma_3 = a_3 + tr\left[a_0(e^{\mathbf{A}T_s})^1 Q + a_1(e^{\mathbf{A}T_s})^0 Q\right] \tag{53d}$$
$$\gamma_4 = a_4 + tr\left[a_0(e^{\mathbf{A}T_s})^2 Q + a_1(e^{\mathbf{A}T_s})^1 Q + a_2(e^{\mathbf{A}T_s})^0 Q\right] \tag{53e}$$
$$\gamma_5 = a_5 + tr\left[a_0(e^{\mathbf{A}T_s})^3 Q + a_1(e^{\mathbf{A}T_s})^2 Q + a_2(e^{\mathbf{A}T_s})^1 Q + a_3(e^{\mathbf{A}T_s})^0 Q\right] \tag{53f}$$
$$\gamma_6 = 0 + tr\left[a_0(e^{\mathbf{A}T_s})^4 Q + a_1(e^{\mathbf{A}T_s})^3 Q + a_2(e^{\mathbf{A}T_s})^2 Q + a_3(e^{\mathbf{A}T_s})^1 Q + a_4(e^{\mathbf{A}T_s})^0 Q\right] \tag{53g}$$
$$\gamma_7 = \gamma_8 = \gamma_9 = \gamma_{10} = 0 \tag{53h}$$

According to these relations we see that 5 extra poles need to be placed. Four of them lie at the origin. Also a new constrain 53b is added and we have less freedom in the pole placement. This constrain implies that the first term of the characteristic equation of the error system γ_1 must correspond with a_1 the first term of the characteristic of the system dynamics. The first term of a characteristic equations contains the sum of all poles. We want to realize a fast system so the error converge rapidly to zero. So we want to place these poles as far as possible on the left side of the right half plane of the unit circle. Now the poles of the error system can be determined. It would be obvious to divide a_1 by 6 an the place of the poles are known.

$$H_{desired} = \frac{1}{\left(z + \frac{a_1}{6}\right)^6 \cdot z^4}$$

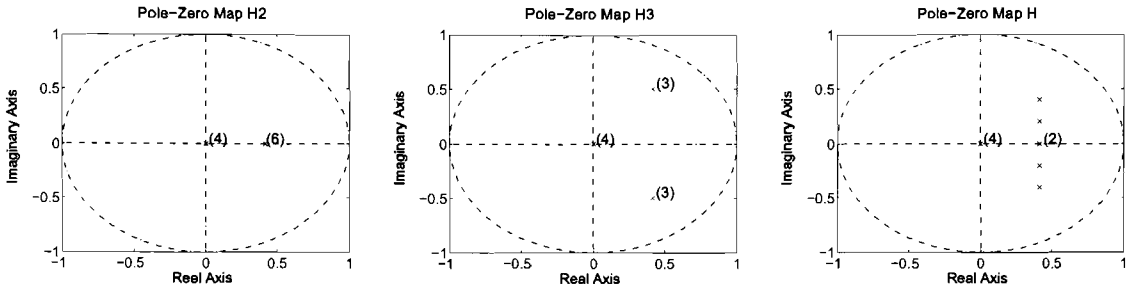
However now we forget a great part of the unit circle namely the imaginary part. The error can converges faster to zero when placing the the poles on the imaginary axis.



(a) x-axis error system (b) y-axis error system

Figure 26: *Pole place indication error system*

The constrain will still hold a pole pair is placed on the vertical line shown in figure 26. By taking the the sum of such a pole pair the imaginary part will be canceled. To find a better solution the poles need to be placed on the vertical line. So there is still a lot of freedom for pole placement.
When placing all the poles on the real axis the system will comparatively react slower than when placed high on the imaginary axis. An impulse response is used to find the settling time. The pole placement is shown in figure 27 and the corresponding settling times can be found in figure 28.

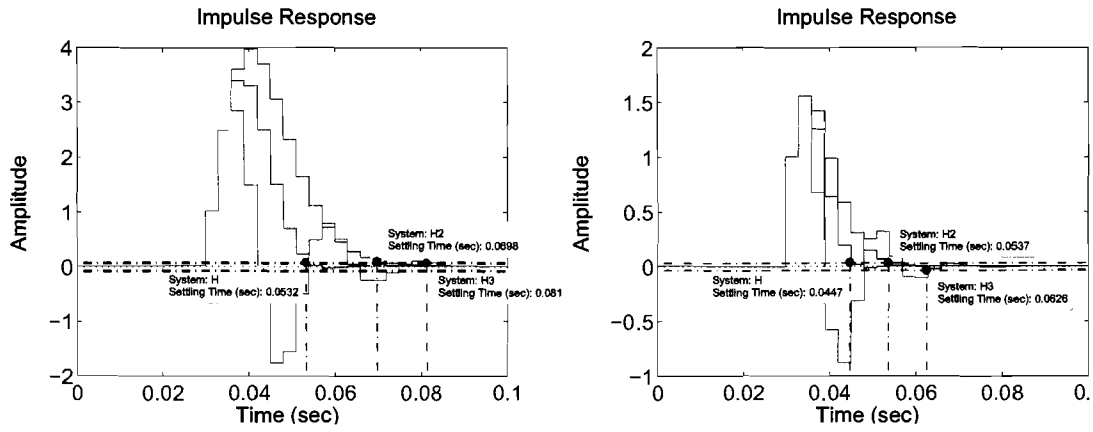


(a) Pole placement Real (b) Pole placement Imaginary (c) Pole placement Compromise

Figure 27: *Pole placements with one delay*

When placing all the poles high on the imaginary axis but just within the unit circle a lot of overshoot will occur. This results in a fast system but due to the overshoot we still have a large settling time. This is shown in figure 28 as the system $H2$. A compromise needs to be made. When we place 2 poles on the real axis causing no overshoot and placing two pole pairs on the imaginary axis causing overshoot the error converges faster to zero.

The real poles are still capable to correct the overshoot. This is shown in figure 27(c) and depicted in figure 28 as system H. This results in a faster system to correct the initial error. The pole placement for the x-axis is shown in figure 27. The corresponding impulse response is shown in figure 28(a). The poles off the y-axis have the same structure and the impulse response in shown in figure 28(b).



(a) Impulse response X

(b) Impulse response Y

Figure 28: *Impulse Response with one delay*

Now the place of the poles are determined, the γ's from the sub-equations 53a can be calculated. The corresponding gain vector K can be determined with the fact that $Q = e^{AT_s}KC$. In table 9 the feedback gain vector K for both axes are shown.

Table 8: *Poles error system with a delay of one sample*

$p_{\#}$	X axis	Y axis
p_1 p_2	0.4137	0.1876
p_3 p_4	0.4137 ± 0.2i	0.1876 ± 0.2i
p_5 p_6	0.4137 ± 0.4i	0.1876 ± 0.4i
p_7 p_8 p_9 p_{10}	0	0

Table 9: *K for delay is one sample*

X axis		Y axis	
$\mathbf{K_x} =$	$5.7 \cdot 10^8$ $6.7 \cdot 10^8$ $1.0 \cdot 10^6$ $-1.2 \cdot 10^4$ $7.9 \cdot 10^0$	$\mathbf{K_y} =$	$3.6 \cdot 10^{10}$ $3.4 \cdot 10^{11}$ $3.1 \cdot 10^7$ $-3.3 \cdot 10^7$ $1.4 \cdot 10^4$

6.3.3 Observer with Two Samples Delay

In this section an observer with a delay of two samples determined. A consequence of the additional delay is that the system will be slower. An indication of the slowest pole is known. It must be on the right side of the horizontal line shown in figure 26(a) and 26(b). We will look now again at the algorithm. The equations can be written as follows,

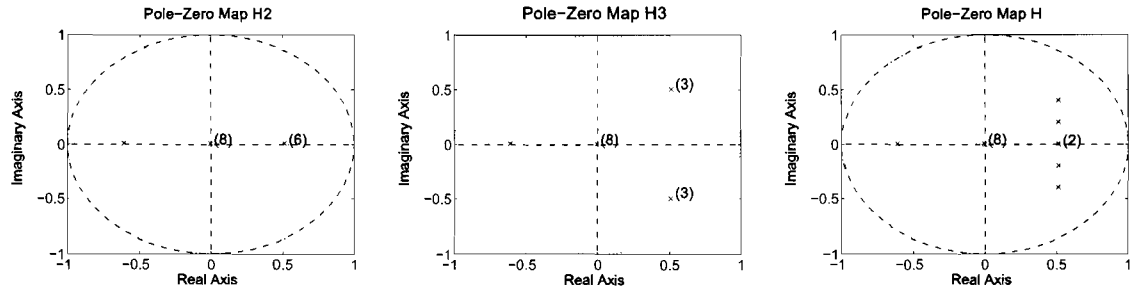
$$\gamma_0 = a_0 \tag{54a}$$
$$\gamma_1 = a_1 \tag{54b}$$
$$\gamma_2 = a_2 \tag{54c}$$
$$\gamma_3 = a_3 + tr\left[a_0(e^{\mathbf{A}T_s})^0 Q\right] \tag{54d}$$
$$\gamma_4 = a_4 + tr\left[a_0(e^{\mathbf{A}T_s})^1 Q + a_1(e^{\mathbf{A}T_s})^0 Q\right] \tag{54e}$$
$$\gamma_5 = a_5 + tr\left[a_0(e^{\mathbf{A}T_s})^2 Q + a_1(e^{\mathbf{A}T_s})^1 Q + a_2(e^{\mathbf{A}T_s})^0 Q\right] \tag{54f}$$
$$\gamma_6 = 0 + tr\left[a_0(e^{\mathbf{A}T_s})^3 Q + a_1(e^{\mathbf{A}T_s})^2 Q + a_2(e^{\mathbf{A}T_s})^1 Q + a_3(e^{\mathbf{A}T_s})^0 Q\right] \tag{54g}$$
$$\gamma_7 = 0 + tr\left[a_0(e^{\mathbf{A}T_s})^4 Q + a_1(e^{\mathbf{A}T_s})^3 Q + a_2(e^{\mathbf{A}T_s})^2 Q + a_3(e^{\mathbf{A}T_s})^1 Q + a_4(e^{\mathbf{A}T_s})^0 Q\right] \tag{54h}$$
$$\gamma_8 = \gamma_9 = \gamma_{10} = \gamma_{11} = \gamma_{12} = \gamma_{13} = \gamma_{14} = \gamma_{15} = 0 \tag{54i}$$

Notice that now two constraints must be fulfilled and 5 extra poles need to be placed compared to the system with one delay. Eight poles will be placed in the origin (see equation 54i). To place the other seven poles constraint $\gamma_1 = a_1$ (eq.54b) must still hold. γ_1 is again the sum of all poles. Beside that the second constraint $\gamma_2 = a_2$ (eq.54c) is added. When we now divide a_1 by seven the first constraint will hold. The sum of the poles will correspond to a1 but the second constraint will not hold. When we look at the place of the poles we see that they lie on the left side of the horizontal line in figure 26. This mean the system will be faster, which is not feasible because an extra delay is added. The system can't be faster. When placing the poles symmetric on the left and on the right side of the horizontal line from figure 26 a solution can be found that hold the constraints. Only by found solution the poles lie outside the unit circle, which will result in instability. When placing one pole on the left half plane of the unit circle and the other poles just across the horizontal line a nice solution can be found. The poles will have a minimum exceeding to the right side of the horizontal line. So the system speed will be minimal slowed down. The one pole on the left half plane can be seen as a correcting pole to make the sum according constrain from equation 54b. This pole has hardly any influence on the dynamics of the error system.
The solution placing all the poles places on the real axis will be determined. These can be found with solving the following relation,

$$H_{desired} = \frac{1}{\left(z + \left(\frac{a_1}{6} - c\right)\right)^6 \cdot (z + (6c)) \cdot z^8}$$

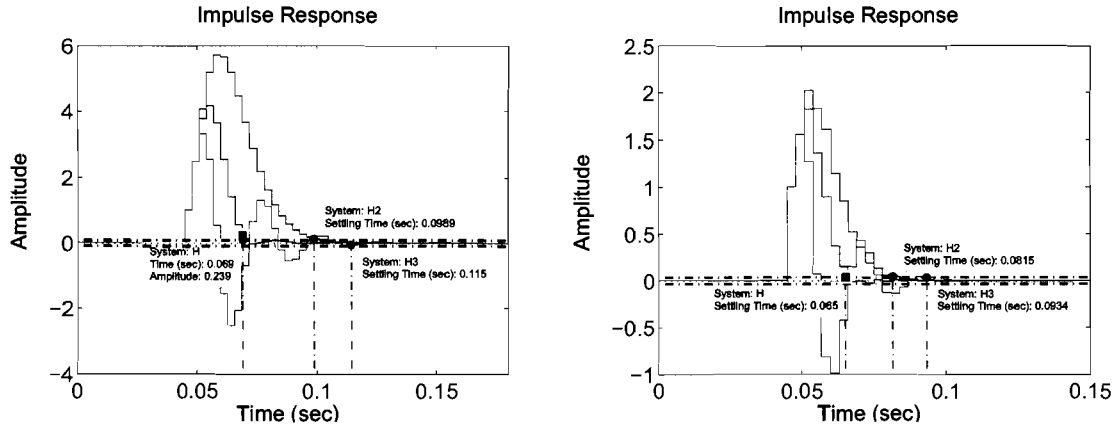
constrain 54b is satisfied and constrain 54c can be met with the factor c.
In this case the poles are placed on the real axis, leaving room for improvement.

In figure 29 the pole placement for the x-axis is shown. This placement is also representative for the y-axis. In figure 29(a) we see the poles placed on the real axis. This corresponds with system $H2$ in figure 30(a) and 30(b) respectively to the x- and the y axis. In figure 29(b) we see the pole placement on the imaginary axis, which is system $H3$ is the sub-figures from 30. The final plot 29(c) is the improved pole placement and correspond with system H in the sub-figures in figure 30. Notice this system has the fastest settling-time.



(a) Pole placement Real (b) Pole placement Imaginary (c) Pole placement Compromise

Figure 29: *Pole placement X two delay*



(a) Impulse response X (b) Impulse response Y

Figure 30: *Impulse Response with two delays*

The poles for this solution can be found in table 10. The calculated feedback gains are shown in table 11.

Table 10: *Poles error system with a delay of two samples*

$p_{\#}$	X axis	Y axis
p_1	-0.5238	-0.7846
p_2 p_3	0.5010	0.3183
p_4 p_5	0.5010 ± 0.2i	0.3183 ± 0.2i
p_6 p_7	0.5010 ± 0.4i	0.3183 ± 0.4i
$p_8 \cdots p_{15}$	0	0

Table 11: *K for delay is two samples*

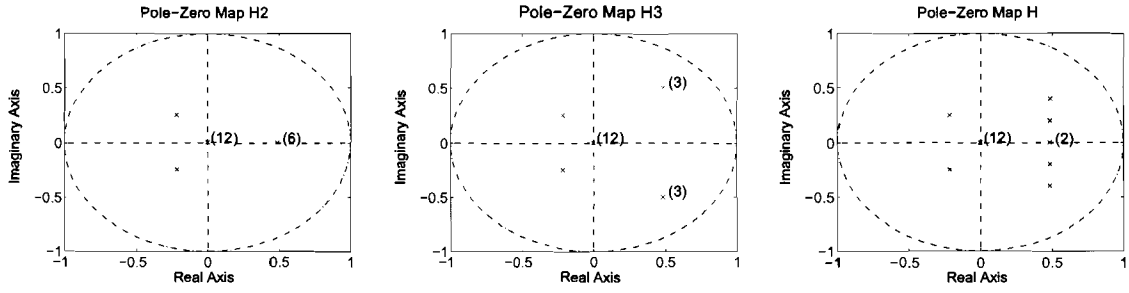
X axis		Y axis	
$\mathbf{K_x} =$	$\begin{bmatrix} 6.7 \cdot 10^{8} \\ 1.5 \cdot 10^{9} \\ 2.2 \cdot 10^{6} \\ -2.6 \cdot 10^{4} \\ 1.7 \cdot 10^{1} \end{bmatrix}$	$\mathbf{K_y} =$	$\begin{bmatrix} 1.5 \cdot 10^{11} \\ 1.9 \cdot 10^{11} \\ 1.7 \cdot 10^{8} \\ -1.8 \cdot 10^{8} \\ 6.5 \cdot 10^{4} \end{bmatrix}$

6.3.4 Observer with Three Samples Delay

The way as before the feedback gains for a delay of three samples are determined. Again a new constraint is added. Now there is a total 3 constraints. The first constrain will be satisfied with the pole placement. Now there are two constraints left so there will be two factors to fulfill these constraints. The following pole placement structure will be used with the factor b and c to determent the γ's of this error system. A solution is found with poles that have a minimum exceeding of the horizontal line from figure 26. And all poles lie within the unit circle.

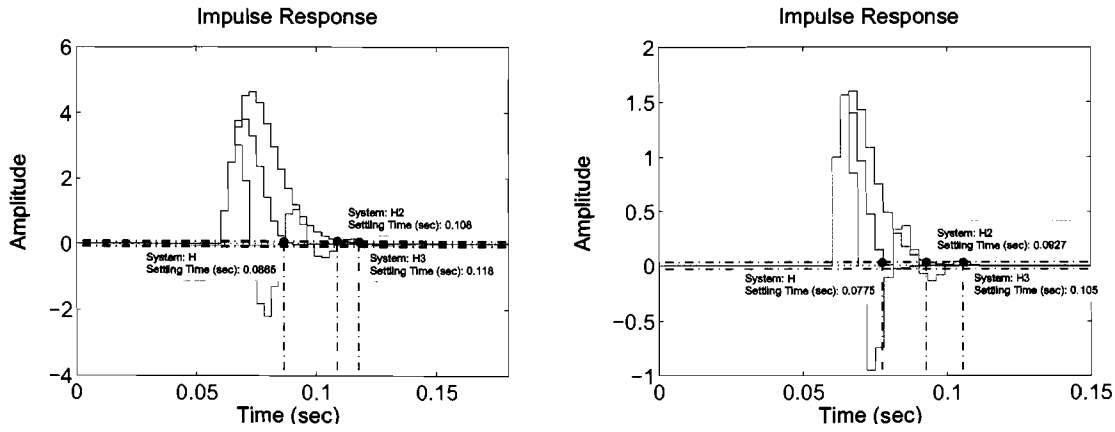
$$H_{desired} = \frac{1}{\left(\left(z + \left(\frac{a_1}{6} - c\right)\right)^6 \cdot (z + (3 \cdot c + b \cdot i)) \cdot (z + (3 \cdot c - b \cdot i))\right) \cdot z^{12}}$$

Afterwards the five poles on the right side of the unit circle will be distributed again over the imaginary axis. Again from three systems the settling time will be compared. The system $H2$ is the system with the poles on the right side of the unit circle on the real axis. $H3$ has all the poles on the right side of the unit circle on the imaginary axis. System H is the found compromise. The pole-placement is visualized in the sub-figures from figure 31. Here we see the pole placement of the x-axis error system. This is corresponding to the pole placement structure of the y-axis error system. The impulse responses from both systems with respect to the pole placement are shown in figure 32



(a) Pole placement Real (b) Pole placement Imaginary (c) Pole placement Compromise

Figure 31: *Pole placement X two delay*



(a) Impulse response X (b) Impulse response Y

Figure 32: *Impulse Response with thee delays*

The exact poles pole-placement and the calculated feedback gains can be found in the tables 12 and 13.

Table 12: *Poles error system with a delay of three samples*

$p_\#$	X axis	Y axis
p_1 p_2	-0.4233 ± 0.4610i	-0.6075 ± 0.5601i
p_3 p_4	0.5548	0.3901
p_5 p_6	0.5548 ± 0.2i	0.3901 ± 0.1i
p_7 p_8	0.5548 ± 0.4i	0.3901 ± 0.3i
$p_9 \dots p_{20}$	0	0

Table 13: *K for delay is three samples*

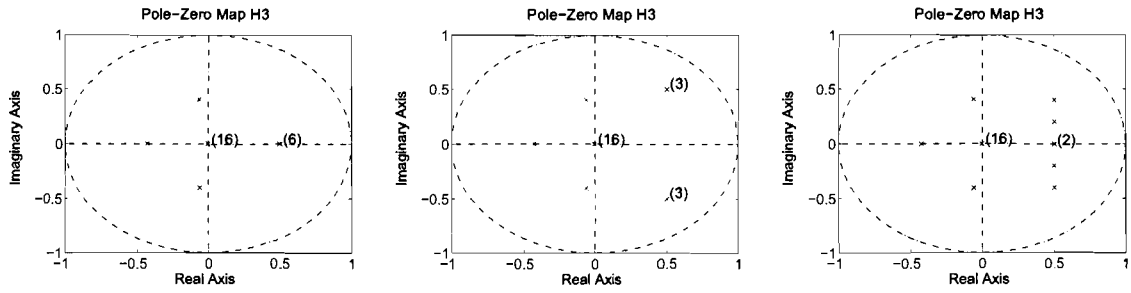
X axis		Y axis	
$\mathbf{K_x} =$	$\begin{bmatrix} 4.8 \cdot 10^8 \\ 1.1 \cdot 10^9 \\ 1.7 \cdot 10^6 \\ -2.0 \cdot 10^4 \\ 1.3 \cdot 10^1 \end{bmatrix}$	$\mathbf{K_y} =$	$\begin{bmatrix} 2.1 \cdot 10^{11} \\ 2.7 \cdot 10^{11} \\ 2.5 \cdot 10^8 \\ -2.6 \cdot 10^8 \\ 9.0 \cdot 10^4 \end{bmatrix}$

6.3.5 Observer with Four Samples Delay

With a delay of four samples there a fourth constraint will be added. The first constrain will again be fulfilled with the pole place structure. Now there are three unknowns which have to satisfy the remaining three constraints. For determining the pole placement the following pole place structure is used.

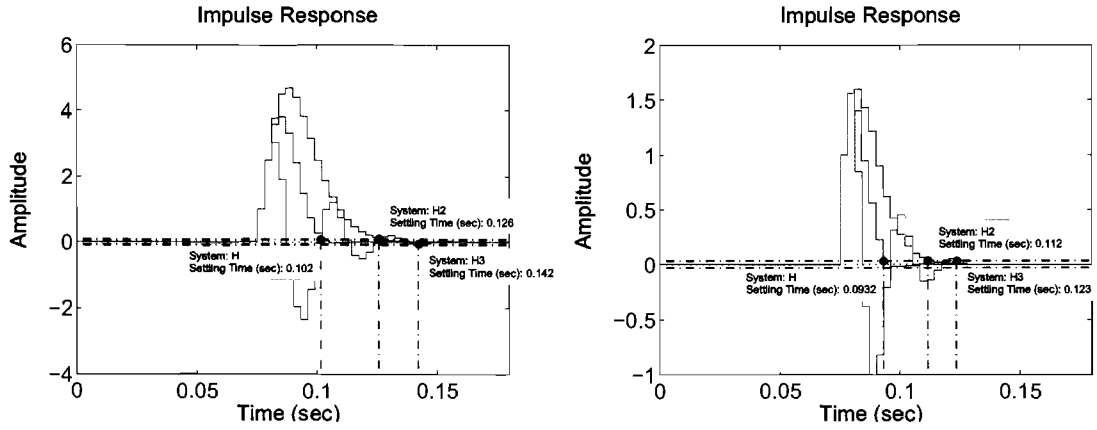
$$H_{desired} = \frac{1}{\left(\left(z + \left(\frac{a_1}{6} - c\right)\right)^6 \cdot (z + (x \cdot c + b \cdot i)) \cdot (z + (x \cdot c - b \cdot i)) \cdot (z + (6 - 2 \cdot x) \cdot c)\right) \cdot z^{16}}$$

The poles found for the x-axis are shown in figure 33(a). Here again the poles will be distributed over the imaginary axis. The final pole placement is shown in figure 33(c). The corresponding impulse response in shown in figure 34(a). The impulse response of the error system of the y-axis is shown in figure 34(b).



(a) Pole placement Real (b) Pole placement Imaginary (c) Pole placement Compromise

Figure 33: *Pole placement X four delays*



(a) Impulse response X

(b) Impulse response Y

Figure 34: *Impulse Response with four delays*

Table 14 and table 15 show the calculated poles and the calculated feedback gains for a system that uses 4 samples delayed data.

Table 14: *Poles error system with a delay of four samples*

$p_{\#}$	X axis	Y axis
p_1	-0.7154	-0.8402
p_2 p_3	-0.1959 ± 0.6912i	-0.3261 ± 0.7540i
p_4 p_5	0.5982	0.4363
p_6 p_7	0.5982 ± 0.2i	0.4363 ± 0.1i
p_8 p_9	0.5982 ± 0.4i	0.4363 ± 0.2i
$p_{10} \cdots p_{25}$	0	0

Table 15: *K for delay is four samples*

X axis		Y axis	
$\mathbf{K_x} =$	$5.7 \cdot 10^8$ $1.6 \cdot 10^9$ $2.4 \cdot 10^6$ $-2.8 \cdot 10^4$ $1.8 \cdot 10^1$	$\mathbf{K_y} =$	$2.5 \cdot 10^{11}$ $3.3 \cdot 10^{11}$ $2.9 \cdot 10^8$ $-3.2 \cdot 10^8$ $1.1 \cdot 10^5$

6.3.6 Observer error dynamics summarized

So for every sample the position information is delayed the error dynamics will be expand with five extra poles. Four of these poles lie in the origin and represent a delay. If the position information is one sample delayed, the settling time will take at least the time of five samples. The relation between the settling time and the number of delays the position information undergoes is shown in figure 35. As expected we can see the more delays the larger the settle time. Also it is shown that the x-axis is slower and will stay slower than the observer for the y-axis.

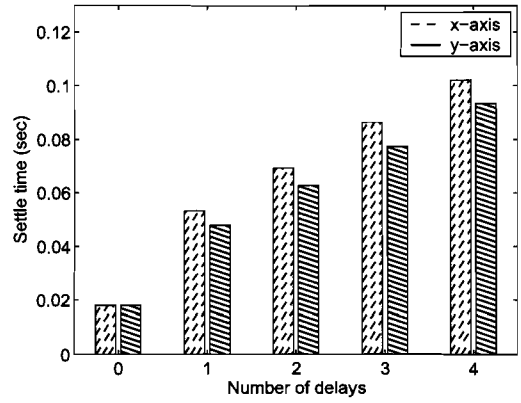
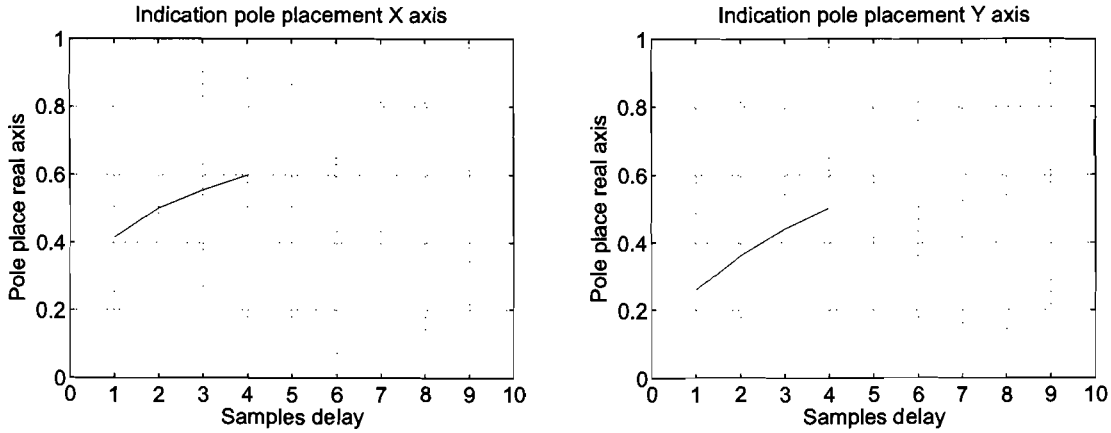


Figure 35: *Delay with corresponding settle time*

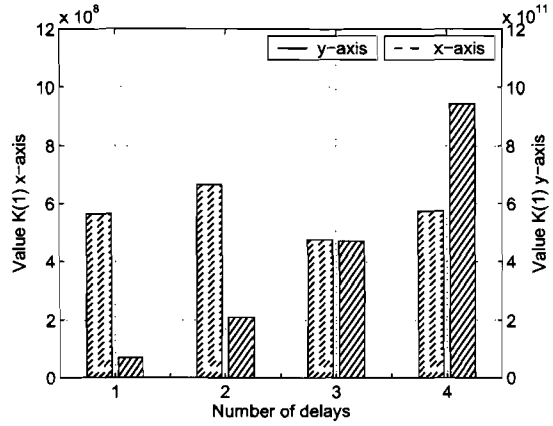
The poles of the error system must lie within the unit circle. In figure 36 we see the trend of the pole placement. The system will become slower and the poles will lie further to the edge of the unit circle. It is hard to say up to how many delays a stable observer can be designed. If the trend follows as depicted in figure 36 it looks that with 10 delays the poles of the error system can still be placed within the unit circle.
Looking form another point of view. Will this delayed position information improve the estimation? The observer will be used to estimate the position for placing a component. For every placement a position profile is generated. Such a position profile has an acceleration and deceleration part. The error of the estimation increases by acceleration and by deceleration. These errors are opposite proportional to each other. Because we want to estimate the position for placing or picking a component we have to deal with the deceleration part of the profile. So it isn't useful to correct the position at this point in time with an error due to the acceleration. On the contrary it will deteriorate the estimation.
So in our case using the fastest position profile with a frequency of 2 Hz we have 62.5 milliseconds of acceleration and 62.5 milliseconds of deceleration. The deceleration error can be used to correct the estimation. We make use of a sample frequency of 3 milliseconds so 20 samples delayed position error can still influence the estimation in a positive way.
So it is useful to correct the observer with a position error from more then 10 samples earlier.



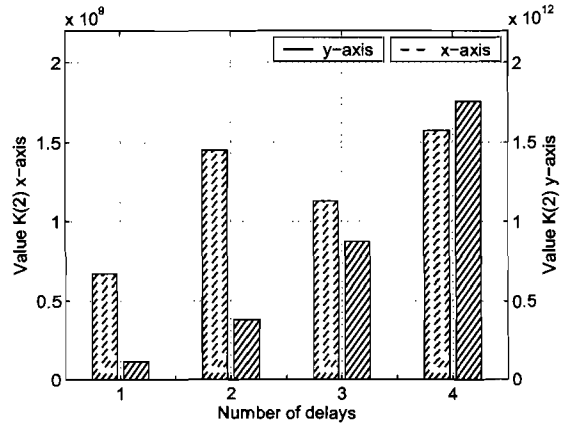
(a) Indication pole placement X-axis observer (b) Indication pole placement Y-axis observer

Figure 36: *Indication pole placement*

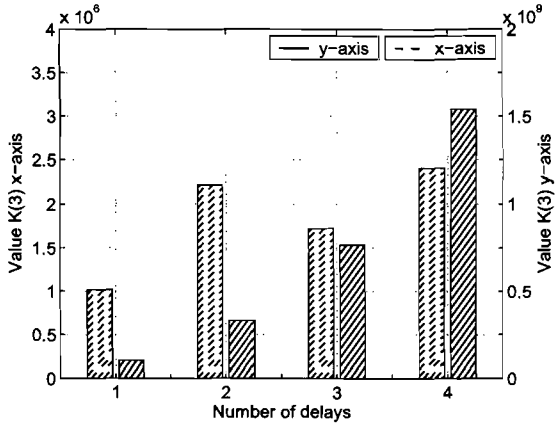
For every added delay in the error dynamics a different feedback gain is calculated. The feedback gain (K) is a vector with a dimension of 5. This means a different feedback value for every state. This vector will amplify or reduce the error. Now the error is scaled to make a appropriate correction on the different system states. In figure 37 we see 5 sub-plots. Every sub-plot contains the feedback gain on the different state. In figure 37(a), figure 37(b) and figure 37(c) we see the feedback gains on the controller states. In the figures 37(d) and figure 37(e) we corrected the gains on the states of the plant. These represent the speed and the position.
We see that the gains for the observer of the y-axis increases with respect to the use of more delayed data. This means that the error from a couple of samples before weighs higher than the error that is only one sample delayed. This is not what we expected.
The gains for the x-axis observer don't show this trend. The fluctuation between these values are not so startlingly.



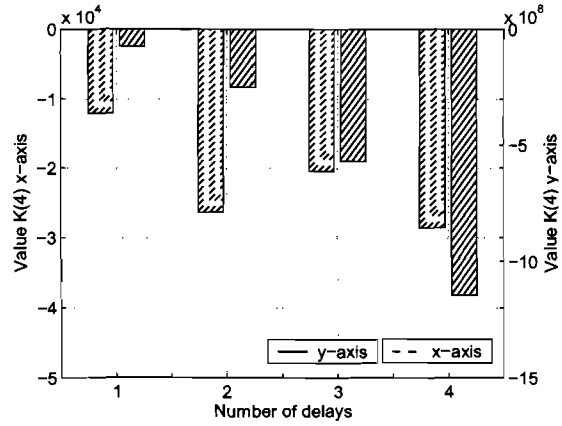
(a) K(1) on first controller state



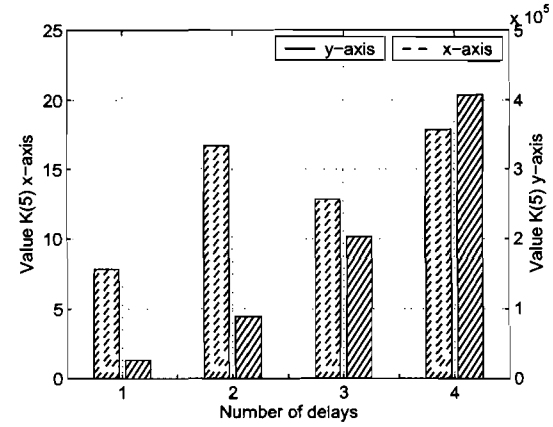
(b) K(2) on second controller state



(c) K(3) on force state



(d) K(4) on speed state



(e) K(5) on position state

Figure 37: *Feedback gains*

It would be convenient if the same feedback gain vector will create a stable observer for different delays. For every delay in the error dynamics an error matrix H shown in equation 42 is formulated. We can easily determine if this error system is stable for the different feedback gains(K_i where i represent the number of samples delay the vector is designed for). So for every error system is analyzed if the poles, with the different feedback gain, lie in the unit circle.
The results are shown in table 16. We see that every error system stays stable with the different feedback gain vectors. Here we distinguish the error system and the feedback gains from the y-axis and x-axis.

Table 16: *Stability of the error system with different K_i*'s*

Number of delays	Used K_i x-axis	Stable	Used K_i y-axis	Stable
One sample delay	K_1	yes	K_1	yes
	K_2	yes	K_2	yes
	K_3	yes	K_3	yes
	K_4	yes	K_4	yes
Two samples delay	K_1	yes	K_1	yes
	K_2	yes	K_2	yes
	K_3	yes	K_3	yes
	K_4	yes	K_4	yes
Three samples delay	K_1	yes	K_1	yes
	K_2	yes	K_2	yes
	K_3	yes	K_3	yes
	K_4	yes	K_4	yes
Four samples delay	K_1	yes	K_1	yes
	K_2	yes	K_2	yes
	K_3	yes	K_3	yes
	K_4	yes	K_4	yes

6.4 Verification

In the previous sections for different delays a stable error system is found. The error system was already discussed and is stable depicted in figure 25. In this subsection the total transfer of the observer will be determined to verify it's stability. The observer loop is show in figure 38 and figure 39.

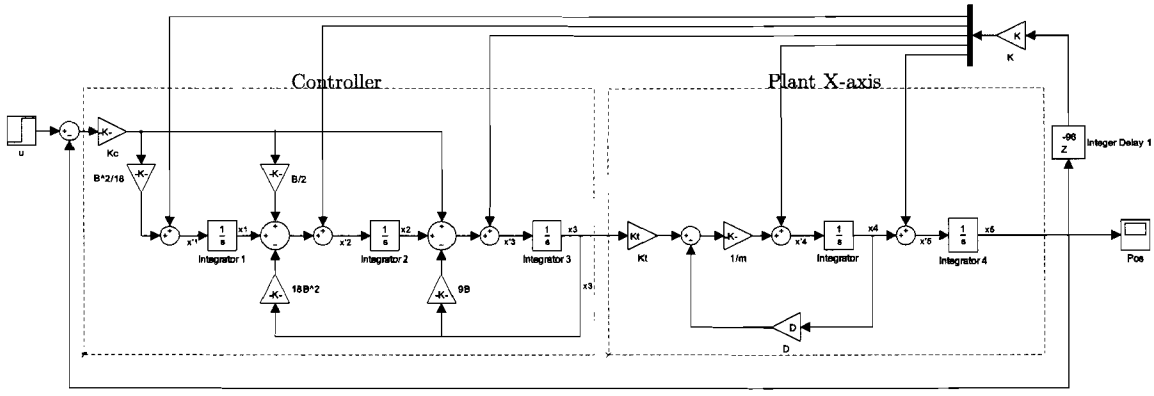


Figure 38: *Observer loop x-axis*

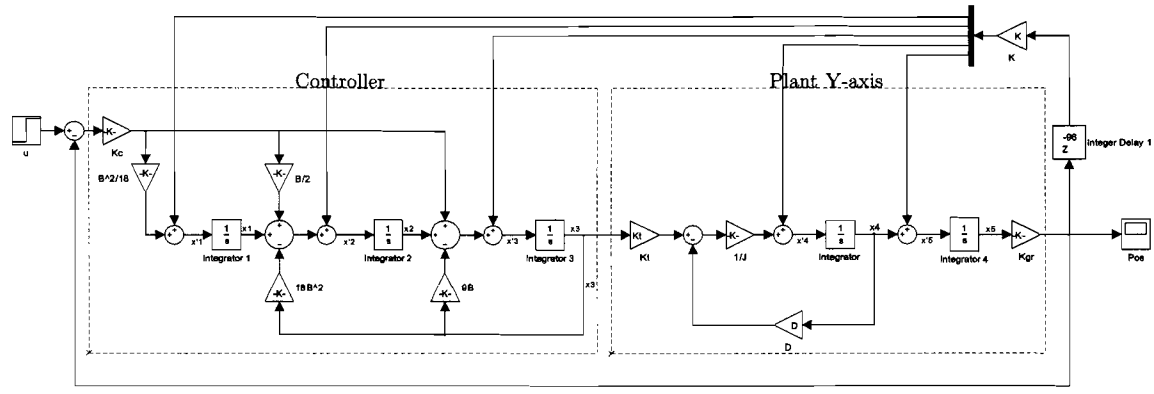


Figure 39: *Observer loop y-axis*

To analyze the system stability again a state-space model will be made. The state-space matrices for the observer for the x-axis and for the y-axis can be found in equation 30 and 31. To add delay terms matlab uses the following representation.

$$\dot{x}_1 = Ax(t) + Bu(t) + \sum_{j=1}^{N}(A_j x(t - t_j) + B_j u(t - t_j)) \tag{55a}$$

$$y(t) = Cx(t) + Du(t) + \sum_{j=1}^{N}(C_j x(t - t_j) + D_j u(t - t_j)) \tag{55b}$$

The Matrices A_{xj}, B_{xj}, C_{xj} and D_{xj} contain the delay terms for the x-axis observer. These are shown in equation 56. These represent the position that is delayed fed back to the observer states with the determined gain.

$$\mathbf{A_{xj}} = \begin{bmatrix} 0 & 0 & 0 & 0 & K(1) \\ 0 & 0 & 0 & 0 & K(2) \\ 0 & 0 & 0 & 0 & K(3) \\ 0 & 0 & 0 & 0 & K(4) \\ 0 & 0 & 0 & 0 & K(5) \end{bmatrix} \qquad \mathbf{B_{xj}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{56}$$

$$\mathbf{C_{xj}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{D_{xj}} = \begin{bmatrix} 0 \end{bmatrix}$$

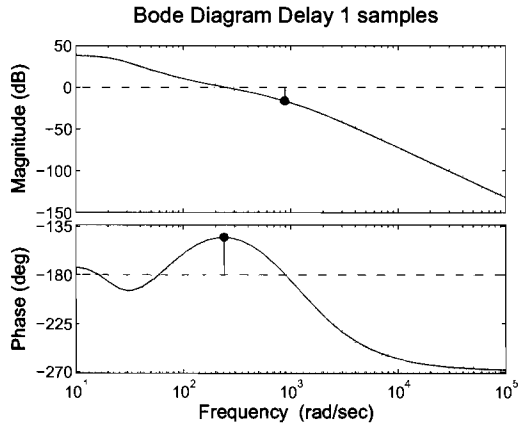
In equation 57 the delayed term of the observer for the y-axis are shown. These represent again the position.

$$\mathbf{A_{yj}} = \begin{bmatrix} 0 & 0 & 0 & 0 & K_g r K(1) \\ 0 & 0 & 0 & 0 & K_g r K(2) \\ 0 & 0 & 0 & 0 & K_g r K(3) \\ 0 & 0 & 0 & 0 & K_g r K(4) \\ 0 & 0 & 0 & 0 & K_g r K(5) \end{bmatrix} \qquad \mathbf{B_{yj}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{57}$$

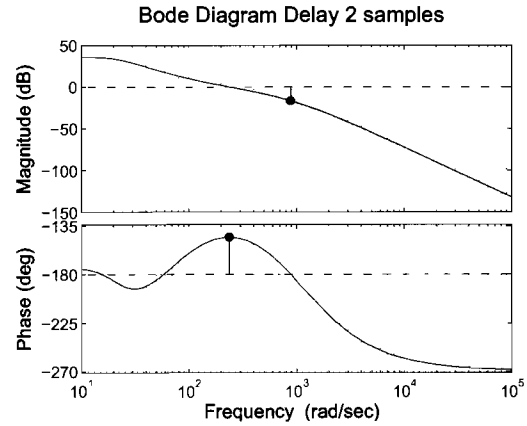
$$\mathbf{C_{yj}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{D_{yj}} = \begin{bmatrix} 0 \end{bmatrix}$$

Now with the matlab command '*delayss*' a state-space model with the delayed term can be generated. We now have the transfer function which can be checked for stability.

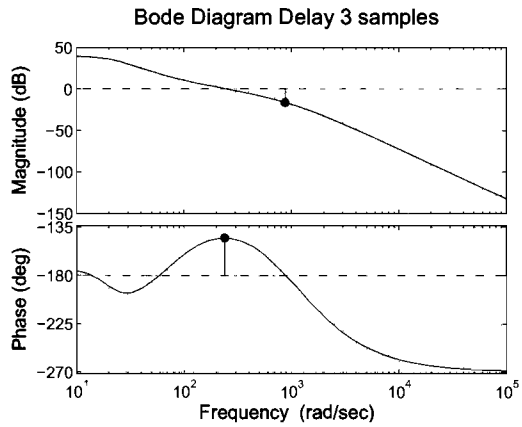
To check the stability the bode plot will be made and analyzed. In the sub figures of figure 40 we see four open-loop bode plots of the observer for the x-axis. The transfer function are almost identical despite of the different delays in the loop. All systems are stable and have a phase margin of 35^o and a phase gain of 17 dB.
Because of hardware compatibility the same sample frequency is used for both systems. The x-axis observer has a very small sample frequency in comparison with the y-axis. This explains the nice response.



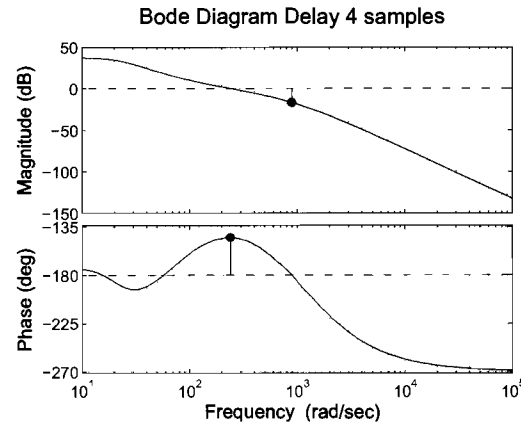
(a) Bode plot system X with One delay



(b) Bode plot system X with Two delay



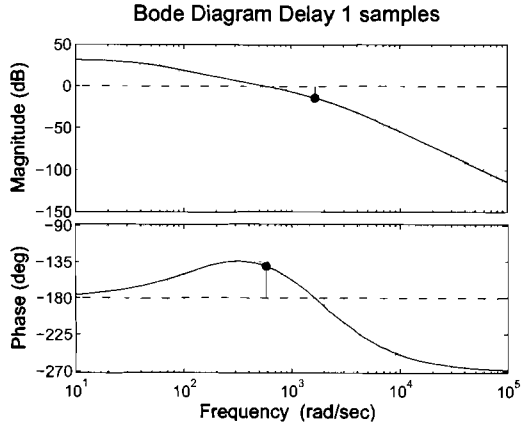
(c) Bode plot system X with Three delay



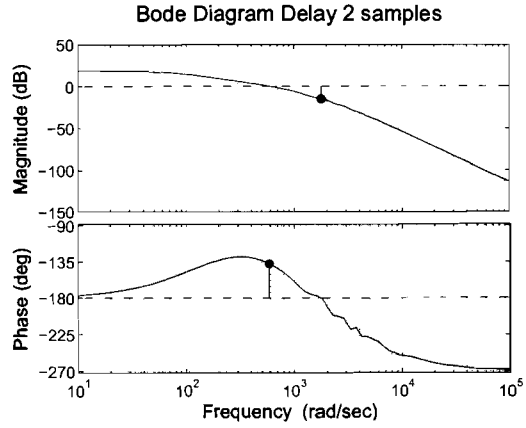
(d) Bode plot system X with Four delay

Figure 40: *Bode plot Observer loop X*

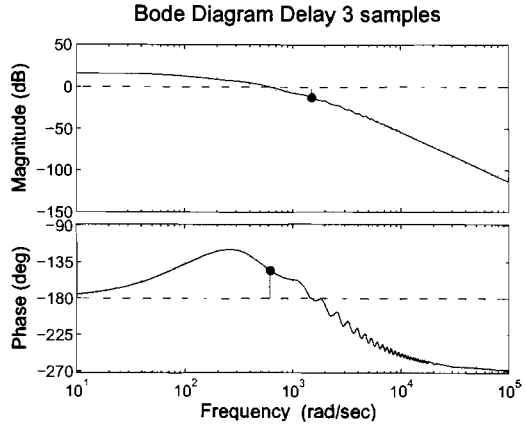
In the sub figures of figure 41 we see four open-loop bode plots of the observer for the y-axis. Here the minimum stability margins are also shown. With only one delay in the loop a phase margin of 40^o is realized and a phase gain of 14 dB. With two delays in the loop we have a phase margin of 43^o and a gain margin of 15 dB.
With three delays in the loop some oscillations show up in the response. Still a phase margin of 34^o is realized and a gain margin of 13 dB. With four delays in the loop more oscillation occur for higher frequencies. Still a stable observer is realized with a phase margin of 34^o but with a gain margin of 12 dB.



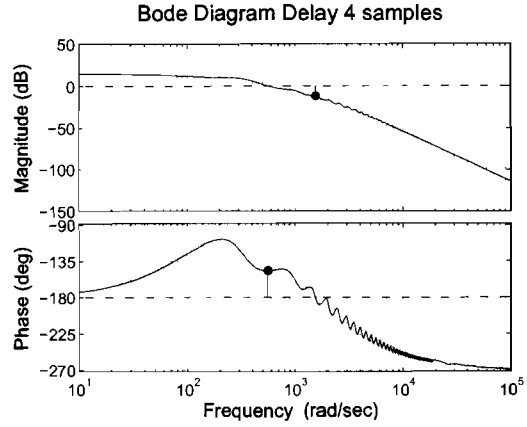
(a) Bode plot system Y with One delay



(b) Bode plot system Y with Two delay



(c) Bode plot system Y with Three delay



(d) Bode plot system Y with Four delay

Figure 41: *Bode plot Observer loop Y*

6.5 Simulation of the observer

We will look if the observer with delayed position information gives an improvement on the open-loop position estimation. For every delay the optimal determent feedback gain will be used. For simulation we make use of the simulink file diagram shown in figure 18, only one block is added containing the observer. This is shown in figure 42, the block is called *'Estimation'*.

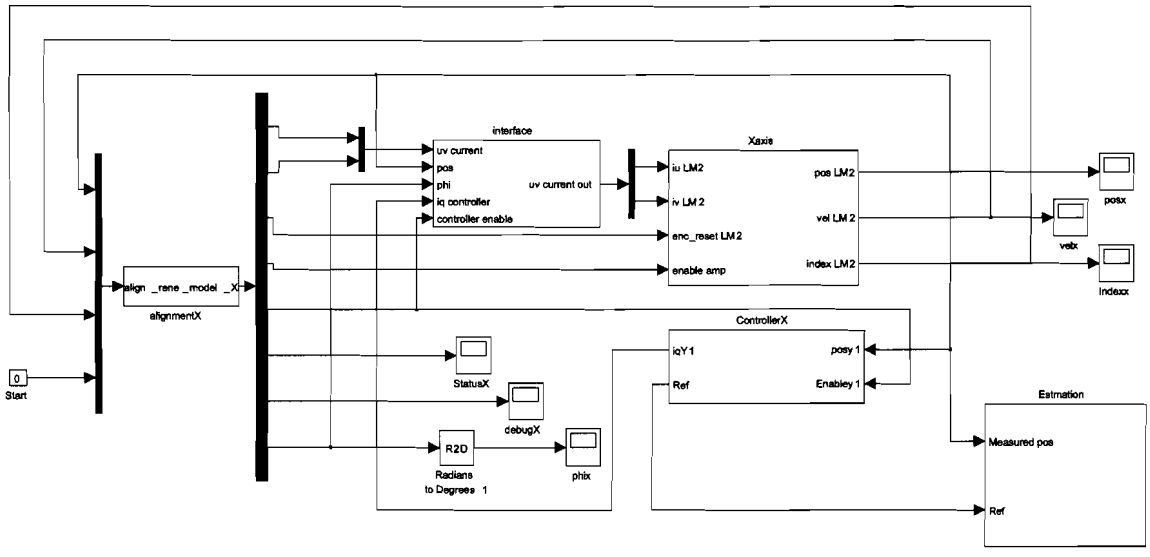


Figure 42: *Total simulation scheme with observer*

This contents of this block is depicted in 43. Here we see how the open-loop and closed-loop estimation is implemented in simulink. The system matrix is shown in equation 30 and 31. The used parameters can be found in table 5 and 6.

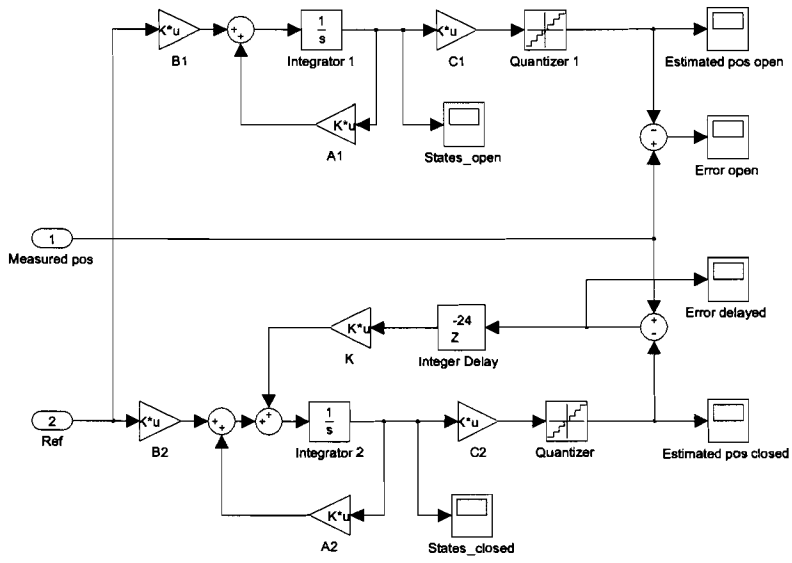


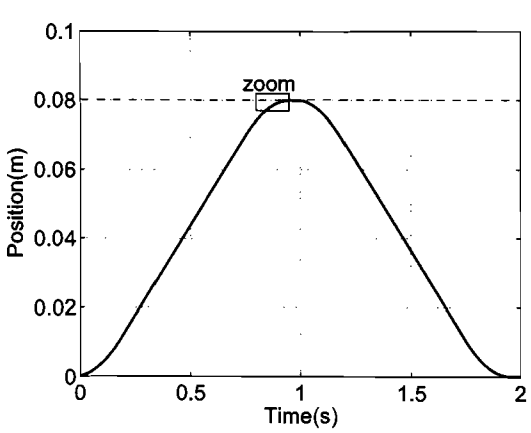
Figure 43: *Observer simulation scheme*

6.5.1 Position estimation

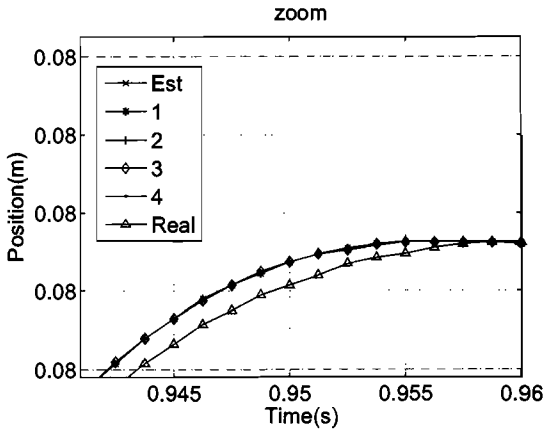
The purpose of this observer is to estimate the position so that a component can be placed fast and accurate on the PCB. Assembléon guarantees an accuracy of $40\mu m$. So we will look when the measured value of the desired position is within this range. To determine it's improvement the open-loop estimation and the closed-loop observer estimation will be compared with the measured position.
First of all the estimation must give a faster indication than the position information can be transmitted. Otherwise the real position can be send through the wireless channel and no improvement is presented. The sample time is 3 millisecond. So within 3 millisecond an accurate position estimation must be available. We will verify this comparing the time the real position is within the $40\mu m$ range with the time the estimation is within this range. The same position profile used for determining the controller accuracy will be used. These profiles can be found in chapter 5 and are depicted in figure 21(a) and 21(b).

First we will look at the observer for the x-axis. In figure 44(a) the used position profile is depicted. Within this picture a dotted horizontal line is shown. These represent the borders of $40\mu m$ from the desired position. When the position if within this range a component can be placed. So here the estimation must correspond very well with the real position information. This part is zoomed out in figure 44(b).
Here the dotted lines show the upper and lower limit of $40\mu m$ from the desired position of 8 cm. Also the measured position information is shown, the open-loop estimation and observer estimations who uses 1 up to 4 samples delay data. We see that the open-loop estimation and the estimations of the observer lie for the most part on each other. Here and there a small improvement is presented at this point of time. Looking at table 17 we see that the maximum error stays nearly the same. But an overall equal improvement is achieved in the average error. The best results are achieved by the observer with 2 delays.

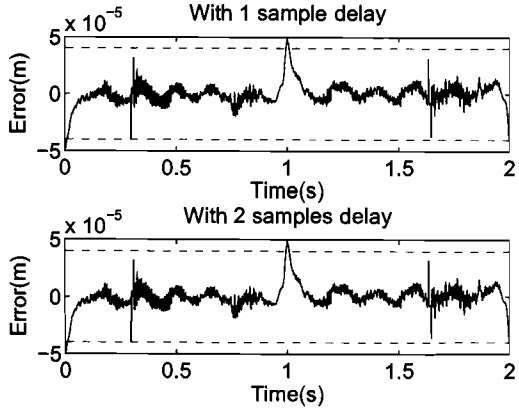
The same is done with the y-axis observer. In figure 45(b) we see that the different observers has all improved the open-loop estimation at this point of time. Also the average error and the maximum error is reduced. The results can be found in table 18. Here the best results are achieved with an observer with 4 samples delayed data.



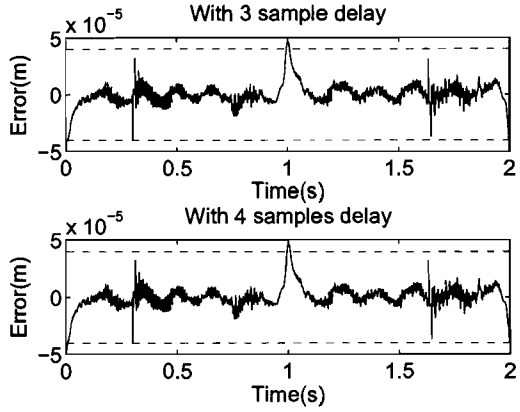
(a) Position profile X axis



(b) Estimation Improvement X axis Simulation



(c) Error with 1 and 2 samples delayed data

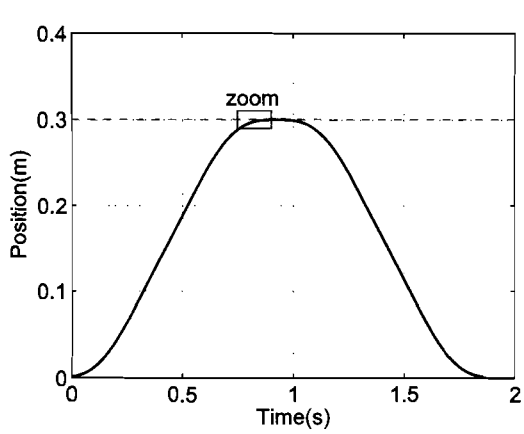


(d) Error with 3 and 4 samples delayed data

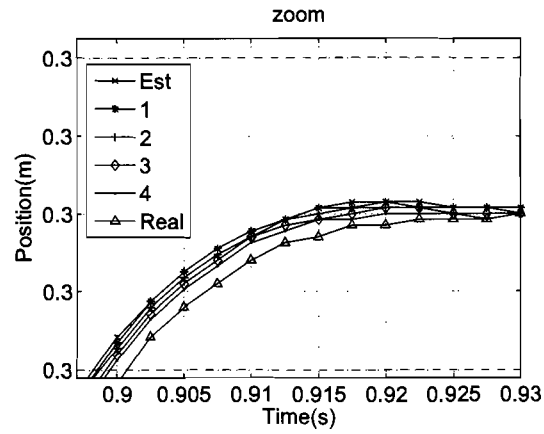
Figure 44: *Simulation observer x-axis*

Table 17: *Error position estimation x-axis*

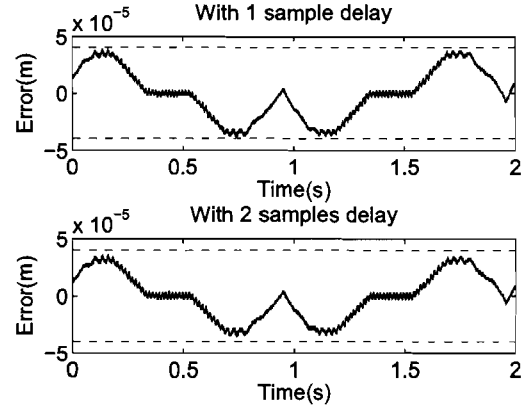
	Average Error	Maximum Error
Open loop	$5.5 \cdot 10^{-6}$	$47.5 \cdot 10^{-6}$
Delay	Average Error	Maximum Error
1	$6.4 \cdot 10^{-7}$	$47.5 \cdot 10^{-6}$
2	$6.3 \cdot 10^{-7}$	$47.0 \cdot 10^{-6}$
3	$6.4 \cdot 10^{-7}$	$47.5 \cdot 10^{-6}$
4	$6.4 \cdot 10^{-7}$	$47.5 \cdot 10^{-6}$



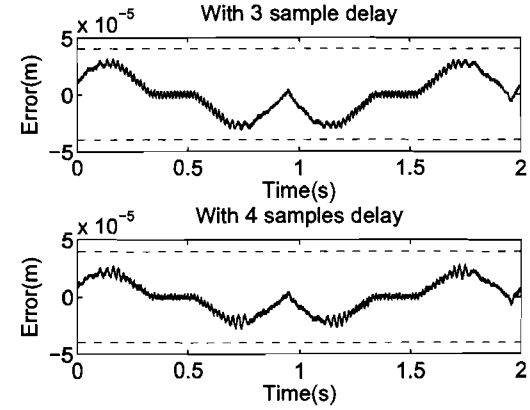
(a) Position profile Y axis



(b) Estimation Improvement Y axis Simulation



(c) Error with 1 and 2 samples delayed data



(d) Error with 3 and 4 samples delayed data

Figure 45: *Simulation observer y-axis*

Table 18: *Error position estimation y-axis*

	Average Error	Maximum Error
Open loop	$18 \cdot 10^{-6}$	$40 \cdot 10^{-6}$
Delay	Average Error	Maximal Error
1	$17{\cdot}10^{-6}$	$39 \cdot 10^{-6}$
2	$15{\cdot}10^{-6}$	$36 \cdot 10^{-6}$
3	$13{\cdot}10^{-6}$	$31 \cdot 10^{-6}$
4	$11{\cdot}10^{-6}$	$28 \cdot 10^{-6}$

6.5.2 Noise sensitivity

The results in this section are obtained by simulation in simulink. The measured position and estimated position will both be generated by a model. To introduce a position error two different models are used. The first model the block coulomb and viscous friction is added and the mass will be enlarged. This model will be used to simulate the real position. This one is placed in the sub-block called *'Model_with_error'*.
The other model will use the determined parameters. This way a representative error is introduced. In the sub figures of figure 47 and 48 we see the system sensitivity for noise. On both axes the same amount of noise is added to the simulated measured position. The simulation scheme is shown in figure 46.

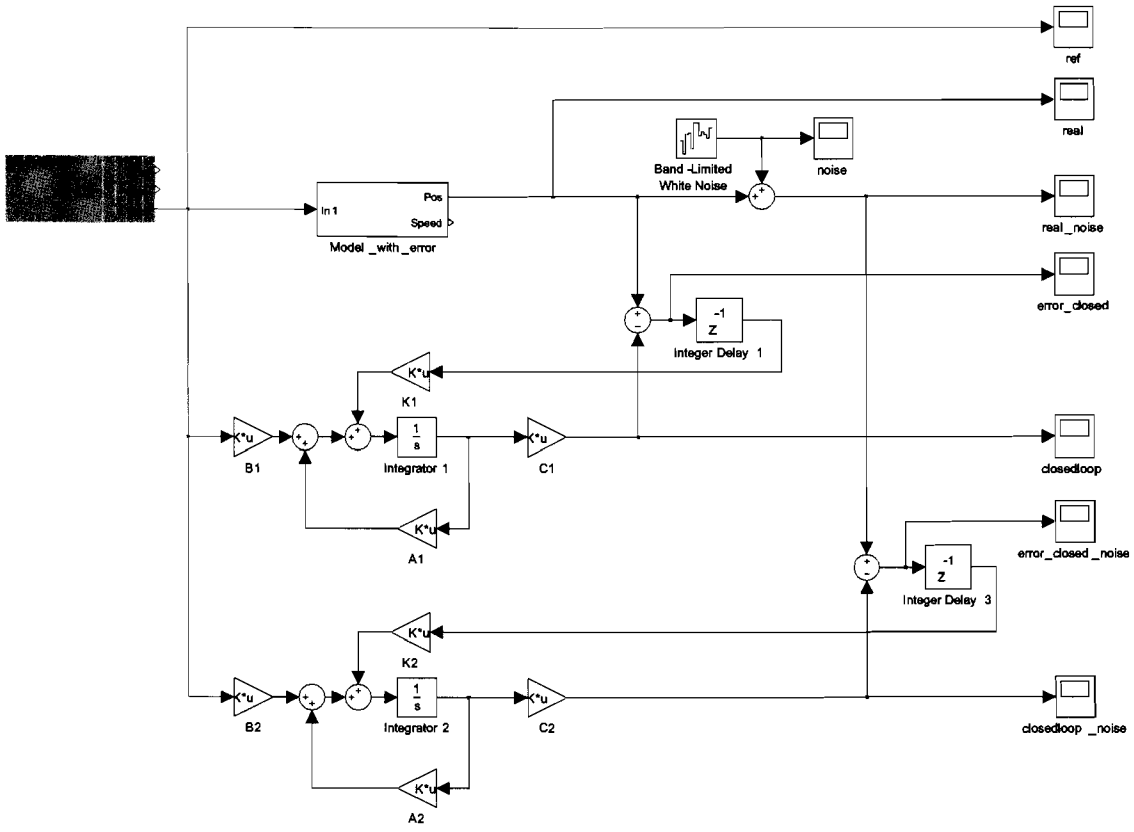


Figure 46: *Simulation with added noise*

For the x-axis the feedback gains are relative small and the improvement on the position information was small. This means that the system is also less insensitive to noise. The y-axis on the contrary is more sensitive for the delayed position information. This results is more sensitivity for noise. The power of the noise will be determined with the relation

shown in equation 58

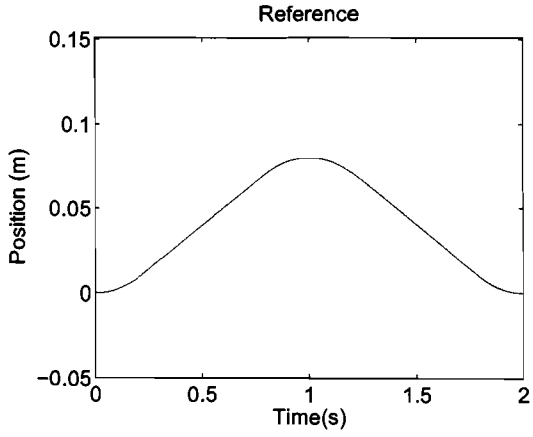
$$P_s = \frac{1}{T}\int_0^T s^2(t)dt \tag{58}$$

In the tables ?? and ?? shows the power of the added noise. For every observer the remaining noise on the position information is determined. This way the noise reduction can be calculated.

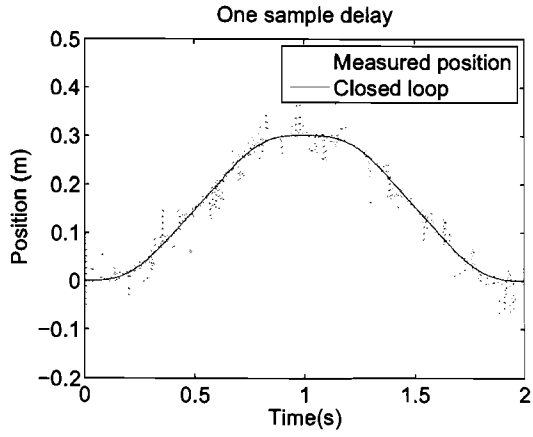
	Added noise	
	$1.0 \cdot 10^{-5}$ W	
Delay of	**Remaining noise**	**Noise Reduction**
One sample	$3.3 \cdot 10^{-10}$ W	-45 dB
Two samples	$1.1 \cdot 10^{-9}$ W	-39 dB
Three samples	$1.2 \cdot 10^{-9}$ W	-38 dB
Four samples	$1.2 \cdot 10^{-9}$ W	-38 dB

	Added noise	
	$1.0 \cdot 10^{-5}$ W	
Delay of	**Remaining noise**	**Noise Reduction**
One sample	$8.0 \cdot 10^{-10}$ W	-40 dB
Two samples	$2.6 \cdot 10^{-8}$ W	-25 dB
Three samples	$5.0 \cdot 10^{-8}$ W	-22 dB
Four samples	$7.6 \cdot 10^{-8}$ W	-21 dB

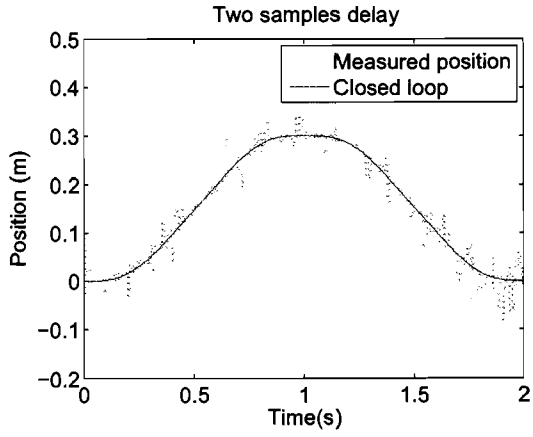
Notice that the estimation from the observer of the x-axis will be less influenced by more delayed values. This results in a smaller correction on the position estimation with a more delayed value. The advantage is that this observer is less sensitive for noise on delayed data.
The observer for the y-axis is more sensitive for delayed values. This results in a better position estimation with delayed values. The disadvantage is that this observer will be more sensitive for noise on delayed data.



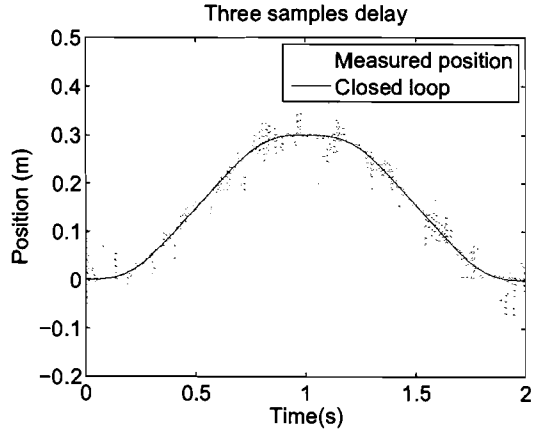
(a) Reference



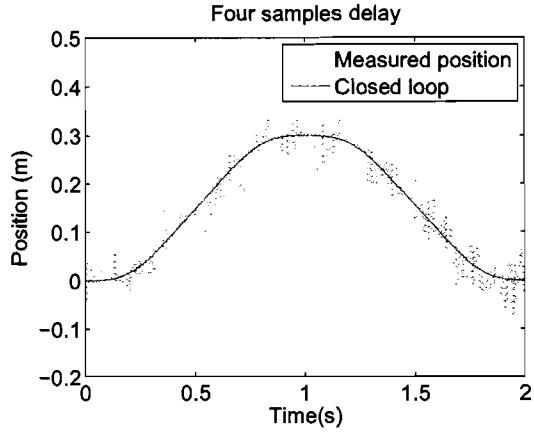
(b) Noise reduction 1 delay



(c) Noise reduction 2 delay

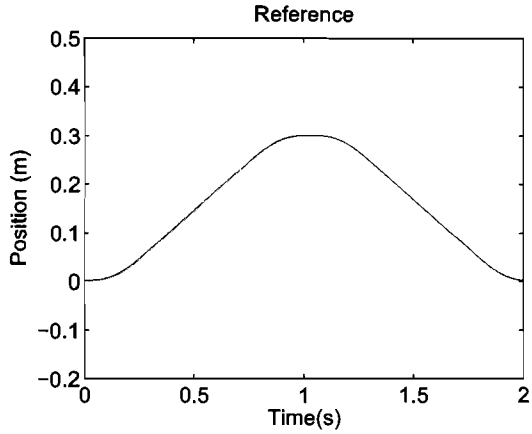


(d) Noise reduction 3 delay

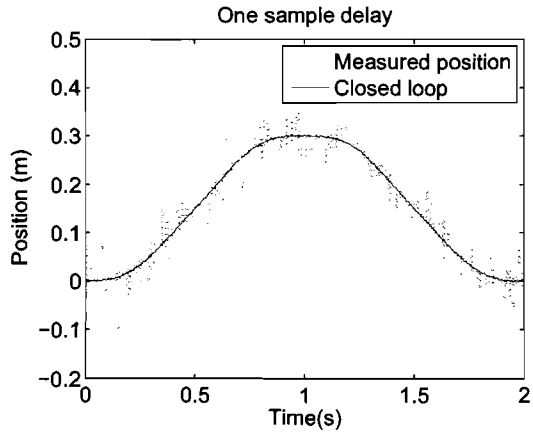


(e) Noise reduction 4 delay

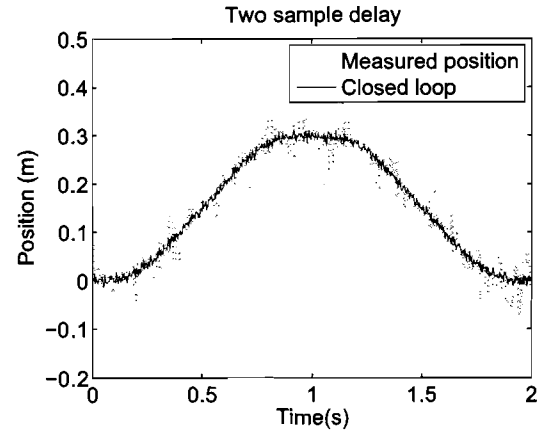
Figure 47: *Noise reduction X axis*



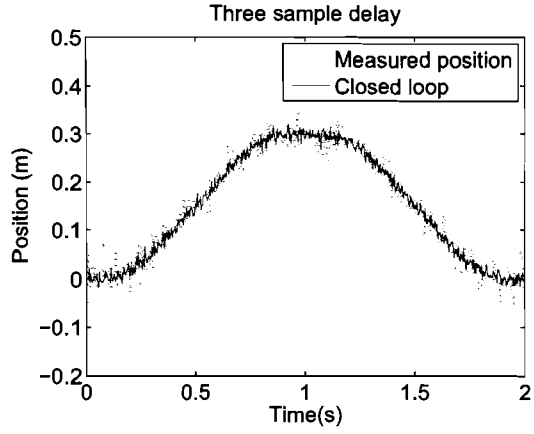
(a) Reference



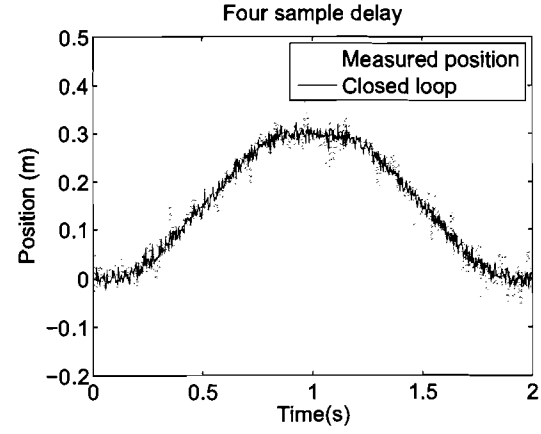
(b) Noise reduction 1 delay



(c) Noise reduction 2 delay



(d) Noise reduction 3 delay



(e) Noise reduction 4 delay

Figure 48: *Noise reduction Y axis*

7 Conclusions

In this thesis we investigated the feasibility for wireless control of a pick and place machine from Assembléon. The aspect that the position information is no longer real time available was singled out. A proper solution is brought up.
For this study Assembléon donated a pick and place machine for research. The first part of this thesis was to overrule this original system. We obtained full control of the pick and place machine in our own research environment.
Suitable hardware was ordered and the corresponding software was successfully implemented. The written alignment procedure finds the maximum current to force relation to control both axes. This is realized with minimum movement of the motors.
The controllers follow the position profiles accurately. An accuracy of 40 μm is obtained for the y-axis. This is the accuracy Assembléon guarantees on their site.
The models in combination with the controller give a nice representation of the dynamics of the plant. With this a good approximation of the position can be made. The average estimation error for the x-axis stays within $6\mu m$ en for the y-axis $18\mu m$.
This representation of the plants will be used in the design of the observers. A method is found to design a stable observer capable to improve the estimation delayed position information. The observer are designed for a fixed number of samples delay. The use of delayed data turns out to an improved estimation. Till now no numeric boundary is found on up to how many samples delayed data can be used to design a stable observer.
The observer will be used to estimate the position for placing a component. For every placement a position profile is determined. Such a position profile has an acceleration and deceleration part. The error of the estimation increases by acceleration and by deceleration. These errors are opposite proportional of each other. Because we want to estimate the position for placing or picking a component we have to deal with the deceleration part of the profile. So it isn't useful to correct the position at this point of time with an error due to acceleration. On the contrary it will deteriorate the estimation.
When using a position profile of 2 Hz a 20 samples delayed position error can still influence the estimation in a positive way.
To determine this observer 19 equations with 19 unknowns must be solved. The 19 unknown are polynomials of the 19^{th} order.
For both axes five observers are designed. The observers are capable to deal with an error from 0 to 4 samples ago. Every observer improves the estimation in a positive way.
The observers are designed for a specified delayed error. Till four delays the determined feedback gains are exchangeable. For all of then the poles of the error dynamics lie within the unit circle.

References

[1] Antoine Verweij, "Control of a permanent magnet linear motor", Master's thesis, 2000.

[2] M. Krawczak and K. Mizukami, "On the application of delayed sampled data observers to some interception problems", *International Journal of Control*, vol. 14, no. 9, pp. 1099–1113, 1983.

[3] A. Salama and V. Gourishankar, "Continuous-Time Observers with Delayed Sampled Data", *IEEE Transactions on automatic control*, vol. AC-23, no. 6, pp. 1107–1109, 1978.

[4] B. Gopinath, "On the control of linear multiple input-output systems", *The Bell System Journal*, vol. 50, no. 3, pp. 1063–1081, March 1971.

[5] Gene Franklin, J.David Powell and Abbas Emami-Naeini, *Feedback Control Of Dynamic Systems*, 2002.

APPENDIX

A C-code

```
/* Alignment procedure for manipulator by Jeroen de Boeij    *\
   edit bij René Hommels for Pick and Place Machine Assembléon
\* September 2007                                            */

#define S_FUNCTION_NAME     align_rene_model_X
#define S_FUNCTION_LEVEL    2
#include "simstruc.h"
#include <math.h>

// define no of inputs of s-function
#define NINPUTS             4//10
// assign tags to inputs and outputs
#define u_pos(element)              (*uPtrs[0])
#define u_vel(element)              (*uPtrs[1])
#define u_index                     (*uPtrs[2])
#define u_enable                    (*uPtrs[3])

// define no of outputs of s-function
#define NOUTPUTS            8//16

// assign tags to outputs and outputs
#define y_ilm1(element)     (yPtrs[0+element])
#define y_enc_reset(element)(yPtrs[2])
#define y_amp_enable        (yPtrs[3])
#define y_controller_enable (yPtrs[4])
#define y_status            (yPtrs[5])
#define y_debug             (yPtrs[6])
#define y_phi(element)      (yPtrs[7])

// define number of states
#define NDSTATES            0
#define NCSTATES            3
#define vest(element)       (xc[0+element])

// define real variables workspace
#define NRWRK               16
#define Iref                prwrk[0]
#define iq                  prwrk[1]
#define PHI                 prwrk[2]
#define DPHI                prwrk[3]
#define RESULT              prwrk[4]
#define temp0               prwrk[5]
#define temp1               prwrk[6]
#define temp2               prwrk[7]
#define temp3               prwrk[8]
#define temp4               prwrk[9]
#define PREVIOUS_RESULT     prwrk[10]
#define STEP_SIZE           prwrk[11]
#define PHI_LM1             prwrk[12]
#define IQ_LM1              prwrk[13]
#define debug               prwrk[14]
#define pref_lm1            prwrk[15]

// define integer variables workspace
#define NIWRK               14
#define STATUS              piwrk[0]
```

```
#define AMP_ENABLED          piwrk[1]
#define CONTROLLER_ENABLED   piwrk[2]
#define ENC_RESET_LM1        piwrk[4]
#define HOMING_STATUS        piwrk[5]
#define TEST_COUNT           piwrk[6]
#define AMPLITUDE_COUNT      piwrk[7]
#define DISC_STEP            piwrk[8]
#define Direction            piwrk[9]
#define Once                 piwrk[10]
#define Last                 piwrk[11]
#define Index                piwrk[12]
#define StartV               piwrk[13]

// define status values
#define WAITING_FOR_START    0
#define TEST                 1
#define TEST_LM1             1
#define ZERO_SEARCH          4
#define ZERO_SEARCH_LM1      4
#define HOMING               7
#define READY                9
#define TRAJECT              10
#define EMERGENCY            13
#define TEST_FAILED          14
#define ZERO_SEARCH_FAILED   15
#define HOMING_FAILED        16
#define EMERGENCY_STOP       17

// define constants
#define pi                   3.14159265358979323846264338327795
#define Imax                 3
#define Iref0                0.35
#define DETECTION_LEVEL      0.000010          // 10 micrometers
#define VIBRATION_PULSE      0.002             // 2 miliseconds
#define HOMING_SPEED_LOW     0.01
#define HOMING_SPEED_HIGH    0.02
#define LM_VMAX              2
#define RM_VMAX              62.8
#define tau                  0.00825
#define Kgain                1000.0
#define eps                  0.0001 // precision to be used instead of == statements
with doubles
#define Plm                  20.0
#define Dlm                  5.0


static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S,0);                          //no parameters expected
    if(ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
        return;

    ssSetNumContStates(S, NCSTATES);
    ssSetNumDiscStates(S, NDSTATES);

    if(!ssSetNumInputPorts(S, 1)) return;
```

```
    ssSetInputPortWidth(S, 0, NINPUTS);
    ssSetInputPortDirectFeedThrough(S, 0, true);    //direct feedthrough

    if(!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, NOUTPUTS);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, NRWRK);
    ssSetNumIWork(S, NIWRK);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0);
}

# define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    int                i;
    int_T              *piwrk = ssGetIWork(S);
    real_T             *prwrk = ssGetRWork(S);
    real_T             *xc    = ssGetContStates(S);

    for(i=0;i<NRWRK;i++){
        prwrk[i]=0.0;
    }

    for(i=0;i<NIWRK;i++){
        piwrk[i]=0;
    }

    for(i=0;i<NCSTATES;i++){
        xc[i]=0.0;
    }

    STATUS = WAITING_FOR_START;
    STEP_SIZE = ssGetStepSize(S);
    DPHI = 0.5*pi;
    PHI = 0;

}

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T  pos;
    int_T   SWITCH_VALUE,vibration_step;

    int_T              *piwrk = ssGetIWork(S);
    real_T             *prwrk = ssGetRWork(S);
    real_T             *xc    = ssGetContStates(S);
    real_T             *yPtrs = ssGetOutputPortRealSignal(S,0);
```

```
    InputRealPtrsType   uPtrs  = ssGetInputPortRealSignalPtrs(S,0);

    vibration_step = (int_T) (VIBRATION_PULSE/STEP_SIZE); // = 0.002/125e-6 =
    pos=0.0;

    //safety
    if(fabs(vest(0)) > LM_VMAX){                          //if speed exceeds 0.3 m/s, go↙
to EMERGENCY
        STATUS = EMERGENCY;
    }

    // set initial values when waiting for start
    if(u_enable < eps ){
        STATUS = WAITING_FOR_START;
        AMP_ENABLED = 0;
        CONTROLLER_ENABLED=0;
        Iref=Iref0;
        HOMING_STATUS = 0;
        Index =0;
        PHI = 0;
        DPHI = 0.5*pi;
        IQ_LM1=0.0;
        debug = 1;
    }

    //get switch_case
    SWITCH_VALUE=STATUS;

//=============================================================================
// WAITING FOR START
//=============================================================================

    switch(SWITCH_VALUE){
        case WAITING_FOR_START:
            IQ_LM1=0.0;
            //IQ_LM2=0.0;
            //IQ_RM=0.0;
            ENC_RESET_LM1=0;
            AMP_ENABLED = 0;
            CONTROLLER_ENABLED=0;
            DISC_STEP=0;
            TEST_COUNT=0;
            AMPLITUDE_COUNT=0;
            Iref=Iref0;
            if(u_enable > 1.0-eps){                       //start switch is enabled
                AMP_ENABLED = 1;
                STATUS=TEST;
                ENC_RESET_LM1=1;
                Iref = Iref0;
                TEST_COUNT = 0;
                ENC_RESET_LM1=1;
                debug = 0;
            }
        break;

//=============================================================================
```

```
// TEST
//=============================================================================

        case TEST:
            // read position of motor to be tested
            if(STATUS == TEST_LM1){
                pos = u_pos(0);
            }
            // generate test pulses
            if(DISC_STEP == (0*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (1*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (2*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP == (2*vibration_step)){
                temp1 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (3*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (4*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP == (4*vibration_step)){
                temp2 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (5*vibration_step)){
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (6*vibration_step)){
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP == (6*vibration_step)){
                temp3 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (7*vibration_step)){
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (8*vibration_step)){
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP == (8*vibration_step)){
                temp4 = pos;
                RESULT = -(temp0-temp1)+(temp1-temp2)+(temp2-temp3)-(temp3-temp4);
                iq = 0;
            }
            else if(DISC_STEP < (10*vibration_step)){
                iq = 0;
            }
```

```
            else if(DISC_STEP == (10*vibration_step)){
                iq = 0;
                temp0 = pos;
                DISC_STEP = 0;
                if((RESULT >= -DETECTION_LEVEL)&&(RESULT <= DETECTION_LEVEL))
                {
                    Iref = 1.2*Iref;            //no movement is detected
                    PHI = PHI + (0.5*pi);
                    TEST_COUNT = 0;
                    if(Iref > Imax)
                    {
                        Iref = Iref0;
                        STATUS = EMERGENCY_STOP;//TEST_FAILED;
                        PHI = 0.0;
                        iq = 0.0;
                    }
                }
                else
                {
                    TEST_COUNT++;               //increase test_count
                    if(TEST_COUNT == 3)         //movement is detected three times
                    {
                        if(STATUS == TEST_LM1){
                            STATUS = ZERO_SEARCH;//EMERGENCY_STOP;
                            Iref = Iref0;
                            DISC_STEP=0;
                        }

                        PREVIOUS_RESULT = 0;    // reinitialize variables
                        TEST_COUNT = 0;
                        DPHI = 0.5*pi;
                        Iref = Iref0;
                    }
                }
            }
            DISC_STEP++;
            // assign q-phi for motor
            IQ_LM1=iq;
            PHI_LM1=PHI;

            // assign outputs
            ENC_RESET_LM1=0;
            AMP_ENABLED = 1;
            CONTROLLER_ENABLED=0;
        break;

//=============================================================================
// ZERO SEARCH LM1
//=============================================================================

        case ZERO_SEARCH:
            if(STATUS == ZERO_SEARCH_LM1){
                pos = u_pos(0);
            }
            // generate pulses
            if(DISC_STEP < (1*vibration_step))
```

```
    iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
else if(DISC_STEP < (2*vibration_step))
    iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
else if(DISC_STEP == (2*vibration_step))
{
    temp1 = pos;
    iq = 0;
}
else if(DISC_STEP < (3*vibration_step))
    iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
else if(DISC_STEP < (4*vibration_step))
    iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
else if(DISC_STEP == (4*vibration_step))
{
    temp2 = pos;
    iq = 0;
}
else if(DISC_STEP < (5*vibration_step))
    iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
else if(DISC_STEP < (6*vibration_step))
    iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
else if(DISC_STEP == (6*vibration_step))
{
    temp3 = pos;
    iq = 0;
}
else if(DISC_STEP < (7*vibration_step))
    iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
else if(DISC_STEP < (8*vibration_step))
    iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
else if(DISC_STEP == (8*vibration_step))
{
    temp4 = pos;
    RESULT = -(temp0-temp1)+(temp1-temp2)+(temp2-temp3)-(temp3-temp4);
    iq = 0;
}
else if(DISC_STEP < (20*vibration_step))
    iq = 0.0;
else if(DISC_STEP == (20*vibration_step))
{
    temp0 = pos;
    if((RESULT >= -DETECTION_LEVEL)&&(RESULT <= DETECTION_LEVEL))
    {
        Iref = 1.2*Iref;
            if(Iref > Imax)
            {
                iq = 0.0;
                PHI=PHI+0.5*pi;
                if(STATUS == ZERO_SEARCH_LM1)
                {
                    PHI_LM1=PHI;
                    ENC_RESET_LM1=1;
                    STATUS = HOMING;
                    StartV=0;
                }
```

```
                            PREVIOUS_RESULT = 0.0;    // reinitialize variables
                            TEST_COUNT = 0;
                            DPHI = 0.5*pi;
                            Iref = Iref0;
                            DISC_STEP=1;
                        }
                    AMPLITUDE_COUNT = 0;
                }
                else
                {
                    AMPLITUDE_COUNT++;          //increase amplitude_count
                    if(((PREVIOUS_RESULT > 0) && (RESULT < 0)) || ((PREVIOUS_RESULT <↙
0) && (RESULT > 0)))
                    {
                        DPHI = DPHI*0.5;
                    }
                    PREVIOUS_RESULT = RESULT;
                    if(RESULT > 0)
                    {
                        PHI = PHI - DPHI;
                    }
                    else
                    {
                        PHI = PHI + DPHI;
                    }
                    if(AMPLITUDE_COUNT>25)
                    {
                        STATUS=ZERO_SEARCH_FAILED;
                        iq = 0.0;
                    }
                }
                iq = 0;
                DISC_STEP = 0;
            }
            DISC_STEP++;
            // assign q-phi for each motor
            if(STATUS == ZERO_SEARCH_LM1){
                IQ_LM1=iq;
                PHI_LM1=PHI;
            }

            ENC_RESET_LM1=0;
            AMP_ENABLED = 1;
            CONTROLLER_ENABLED=0;

        break;

//=============================================================================
// HOMING
//=============================================================================

        case HOMING:
            if(HOMING_STATUS==0){                    // positive current positive↙
movement
                if(DISC_STEP==1){
                    temp0=u_pos(0);
```

```
                    IQ_LM1=Iref0;
                }
                else if(DISC_STEP < (10*vibration_step)){
                    IQ_LM1=Iref0;
                }
                else if(DISC_STEP == (10*vibration_step)){
                    IQ_LM1 = 0;
                    temp1=u_pos(0)-temp0;
//                      // if positive current leads to negative displacement
                    // change PHI with pi to reverse direction
                     if(temp1<-DETECTION_LEVEL){
                         PHI_LM1+=pi;
                     }
                     if(fabs(temp1)<DETECTION_LEVEL){
                         STATUS=HOMING_FAILED;
                     }
                    Direction = -1;
                    HOMING_STATUS++;
                    DISC_STEP=0;
                    IQ_LM1 = 0;
                }
                DISC_STEP++;
                ENC_RESET_LM1=0;
            }

            //wait for LM1 to stand still after movement of LM1
            if(HOMING_STATUS==1){
                if( fabs(vest(0))<= 0.0001){
                    HOMING_STATUS++;
                    DISC_STEP=1;
                }
                IQ_LM1=0.0;
                ENC_RESET_LM1=0;
                temp1 = 100;
                Once = 0;
            }
            if(HOMING_STATUS==2){
                if(DISC_STEP==1){
                    temp0=u_pos(0);
                    IQ_LM1= 0.47*Iref0*Direction;       //(0.25*(1-vest(0)))*Direction;
                }
                else if(DISC_STEP < (10*vibration_step)){
                    IQ_LM1= 0.47*Iref0*Direction;       //(0.25*(1-vest(0)))*Direction;
                    if (temp0 == temp1){                // het bereik bereikt,↙
rechterhoek
                        //IQ_LM1 = 0;
                        HOMING_STATUS = 6;              //geen index zoeken
                        Direction = Direction *(-1);
                        }
                }

                else if(DISC_STEP == (10*vibration_step)){
                    IQ_LM1= 0.47*Iref0*Direction;       //(0.24*(1-vest(0)))*Direction;
                    temp1=temp0;
                    DISC_STEP=0;
                }
```

```
DISC_STEP++;
}

if(HOMING_STATUS==3){              //find index
   IQ_LM1 = 0.35*Iref0*Direction; //(0.12*(1-vest(0)))*Direction;
   if (u_index > 0.99 && Once == 0){
        Index++;
        Once = 1;
   }
   if (u_index < 0.01){
        Once = 0;
   }
}
if(Index == 2){
       IQ_LM1 = 0;
       HOMING_STATUS++;
       Direction = Direction*(-1);
       Index = 0;
}


if(HOMING_STATUS==4){       //wacht  tot stilstand
    if( fabs(vest(0))<= 0.1*(DETECTION_LEVEL/STEP_SIZE) ){
            HOMING_STATUS++;
            DISC_STEP=0;
    }
}

if(HOMING_STATUS==5){         //index low speed
    IQ_LM1= 0.25*Direction;   //(0.65*(0.25-vest(0)))*Direction;
    if (u_index > 0.9){
        PHI_LM1+=pi/tau*u_pos(0);
        ENC_RESET_LM1=1;
        HOMING_STATUS++;
        IQ_LM1 = 0;
        ENC_RESET_LM1=0;
    }
    DISC_STEP++;
    if (DISC_STEP == 10000){   //index not found
        IQ_LM1 = 0;
        HOMING_STATUS = 0;
        DISC_STEP=1;
    }
}

if(HOMING_STATUS==6){               //wacht  tot stilstand
    if( fabs(vest(0))<= 0.1*(DETECTION_LEVEL/STEP_SIZE) ){
            //HOMING_STATUS++;
            PHI_LM1+=pi/tau*u_pos(0);
            ENC_RESET_LM1=1;
            DISC_STEP++;
               if (DISC_STEP == 10){  //wait reset
                 DISC_STEP=1;
               }
            STATUS = READY;
            DISC_STEP=1;
```

```
                }
            }

        break;

//=============================================================================
// READY
//=============================================================================

        case READY:
            IQ_LM1=0;//-vest(0);
            ENC_RESET_LM1=0;
            AMP_ENABLED = 1;
            DISC_STEP++;
               if (DISC_STEP == 10000){
                   CONTROLLER_ENABLED=1;
                   DISC_STEP=1;
               }

        break;


//=============================================================================
// EMERGENCY
//=============================================================================

case EMERGENCY:
            ENC_RESET_LM1=0;
            AMP_ENABLED = 1;
            CONTROLLER_ENABLED=0;
            IQ_LM1=-10*vest(0);
            if (AMP_ENABLED==1){
                if((fabs(vest(0))<DETECTION_LEVEL/STEP_SIZE)){//&& (fabs(vest(1))
<DETECTION_LEVEL/STEP_SIZE) ){
                    IQ_LM1=0.0;
                    STATUS=EMERGENCY_STOP;
                }
            }
        break;
//=============================================================================
// EMERGENCY_STOP
//=============================================================================

        case EMERGENCY_STOP:
            IQ_LM1=0.0;
            ENC_RESET_LM1=0;
            AMP_ENABLED = 0;
            CONTROLLER_ENABLED=0;
        break;

    }


    //check current saturation
    if(IQ_LM1>Imax){
```

```
        IQ_LM1=Imax;
    }
    else if(IQ_LM1<-Imax){
        IQ_LM1=-Imax;
    }

    //generate three phase outputs

    y_ilm1(0)=IQ_LM1*sin(pi/tau*u_pos(0)+PHI_LM1);
    y_ilm1(1)=IQ_LM1*sin(pi/tau*u_pos(0)+PHI_LM1+2.0/3.0*pi);


    // other outputs
    y_enc_reset(0)=ENC_RESET_LM1;

    y_controller_enable = CONTROLLER_ENABLED;
    y_status = STATUS;
    y_amp_enable = AMP_ENABLED;
    y_debug = debug;

    // phase angle
    y_phi(0)=PHI_LM1;
}

#define MDL_UPDATE
#if defined(MDL_UPDATE)
static void mdlUpdate(SimStruct *S, int_T tid)
{
}
#endif

#define MDL_DERIVATIVES
#if defined(MDL_DERIVATIVES)
static void mdlDerivatives(SimStruct *S)
{
    real_T            *dx = ssGetdX(S);
    real_T            *xc = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    int i;

    for(i=0;i<NCSTATES;i++){
        dx[i]=Kgain*(u_vel(i)-vest(i));
    }
}
#endif

static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

```
/* Alignment procedure for manipulator by Jeroen de Boeij    *\
   edit bij René Hommels for Pick and Place Machine Assembléon
\* September 2007                                            */

#define S_FUNCTION_NAME     align_rene_model_Y
#define S_FUNCTION_LEVEL    2
#include "simstruc.h"
#include <math.h>

// define no of inputs of s-function
#define NINPUTS             4//10

// assign tags to inputs and outputs
#define u_pos(element)              (*uPtrs[0]) //(*uPtrs[0+element])
#define u_vel(element)              (*uPtrs[1]) //(*uPtrs[3+element])
#define u_index                     (*uPtrs[2]) //(*uPtrs[6+element])
#define u_enable                    (*uPtrs[3]) //(*uPtrs[9])

// define no of outputs of s-function
#define NOUTPUTS            8//16
// assign tags to outputs and outputs
#define y_irm(element)      (yPtrs[0+element])
#define y_enc_reset(element)(yPtrs[2])
#define y_amp_enable        (yPtrs[3])
#define y_controller_enable (yPtrs[4])
#define y_status            (yPtrs[5])
#define y_debug             (yPtrs[6])
#define y_phi(element)      (yPtrs[7])

// define number of states
#define NDSTATES            0
#define NCSTATES            3
#define vest(element)       (xc[0+element])

// define real variables workspace
#define NRWRK               16
#define Iref                prwrk[0]
#define Iq                  prwrk[1]
#define PHI                 prwrk[2]
#define DPHI                prwrk[3]
#define RESULT              prwrk[4]
#define temp0               prwrk[5]
#define temp1               prwrk[6]
#define temp2               prwrk[7]
#define temp3               prwrk[8]
#define temp4               prwrk[9]
#define PREVIOUS_RESULT     prwrk[10]
#define STEP_SIZE           prwrk[11]
#define PHI_RM             prwrk[12]
#define IQ_RM              prwrk[13]
#define debug               prwrk[14]
#define pref_RM            prwrk[15]

// define integer variables workspace
#define NIWRK               15
#define STATUS              piwrk[0]
```

```
#define AMP_ENABLED          piwrk[1
#define CONTROLLER_ENABLED   piwrk[2
#define ENC_RESET_RM         piwrk[4
#define HOMING_STATUS        piwrk[5
#define TEST_COUNT           piwrk[6
#define AMPLITUDE_COUNT      piwrk[7
#define DISC_STEP            piwrk[8
#define Direction            piwrk[9
#define Once                 piwrk[10
#define Last                 piwrk[11
#define Index                piwrk[12
#define StartV               piwrk[13
#define laststatus           piwrk[14

// define status values
#define WAITING_FOR_START    0
#define TEST                 1
#define TEST_RM              1
#define WAIT                 3
#define ZERO_SEARCH          4
#define ZERO_SEARCH_RM       4
#define HOMING               7

#define READY                9
#define TRAJECT              10
#define EMERGENCY            13
#define TEST_FAILED          14
#define ZERO_SEARCH_FAILED   15
#define HOMING_FAILED        16
#define EMERGENCY_STOP       17

// define constants
#define pi                   3.14159265358979323846264338327 95
#define Imax                 3
#define Iref0                0.1
#define DETECTION_LEVEL      0.000010          // 10 micrometers
#define VIBRATION_PULSE      0.004             // 2 miliseconds
#define HOMING_SPEED_LOW     0.01
#define HOMING_SPEED_HIGH    0.02
#define LM_VMAX              2
#define RM_VMAX              0.7 // m/s
#define DIFF_MAX             0.005
#define tau                  0.00375
#define Kgain               1000.0
#define eps                  0.0001 // precision to be used instead of == statements
with doubles
#define Plm                  20.0
#define Dlm                  5.0


static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S,0);                          //no parameters expected
    if(ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))
        return;
```

```
    ssSetNumContStates(S, NCSTATES);
    ssSetNumDiscStates(S, NDSTATES);

    if(!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, NINPUTS);
    ssSetInputPortDirectFeedThrough(S, 0, true);    //direct feedthrough

    if(!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, NOUTPUTS);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, NRWRK);
    ssSetNumIWork(S, NIWRK);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0);
}

# define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    int                i;
    int_T              *piwrk = ssGetIWork(S);
    real_T             *prwrk = ssGetRWork(S);
    real_T             *xc    = ssGetContStates(S);

    for(i=0;i<NRWRK;i++){
        prwrk[i]=0.0;
    }

    for(i=0;i<NIWRK;i++){
        piwrk[i]=0;
    }

    for(i=0;i<NCSTATES;i++){
        xc[i]=0.0;
    }

    STATUS = WAITING_FOR_START;
    STEP_SIZE = ssGetStepSize(S);
    DPHI = 0.5*pi;
    PHI = 0;

}

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T  pos;
    int_T   SWITCH_VALUE,vibration_step;
```

```
    int_T              *piwrk = ssGetIWork(S);
    real_T             *prwrk = ssGetRWork(S);
    real_T             *xc    = ssGetContStates(S);
    real_T             *yPtrs = ssGetOutputPortRealSignal(S,0);
    InputRealPtrsType  uPtrs  = ssGetInputPortRealSignalPtrs(S,0);

    vibration_step = (int_T) (VIBRATION_PULSE/STEP_SIZE); // = 0.002/125e-6 =
    pos=0.0;

    //safety
    if(fabs(vest(0)) > RM_VMAX){                          //if speed exceeds 0.3 m/s, go↙
to EMERGENCY
        STATUS = EMERGENCY;
    }

    // set initial values when waiting for start
    if(u_enable < eps ){
        STATUS = WAITING_FOR_START;
        AMP_ENABLED = 0;
        CONTROLLER_ENABLED=0;
        Iref=Iref0;
        HOMING_STATUS = 0;
        Index =0;
        PHI = 0;
        DPHI = 0.5*pi;
        IQ_RM=0.0;
        debug = 1;
    }

    //get switch_case
    SWITCH_VALUE=STATUS;

//=============================================================================
// WAITING FOR START
//=============================================================================

    switch(SWITCH_VALUE){
        case WAITING_FOR_START:
            IQ_RM=0.0;
            ENC_RESET_RM=0;
            AMP_ENABLED = 0;
            CONTROLLER_ENABLED=0;
            DISC_STEP=0;
            TEST_COUNT=0;
            AMPLITUDE_COUNT=0;
            Iref=Iref0;
            if(u_enable > 1.0-eps){                       //start switch is enabled
                AMP_ENABLED = 1;
                STATUS=TEST;
                Iref = Iref0;
                TEST_COUNT = 0;
                ENC_RESET_RM=1;
                debug = 0;
            }
        break;
```

```
//=============================================================================
// TEST
//=============================================================================

        case TEST:
            // read position of motor to be tested
            if(STATUS == TEST_RM){
                pos = u_pos(0);
            }

            // generate test pulses
            if(DISC_STEP == (0*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (1*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (2*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP == (2*vibration_step)){
                temp1 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (3*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (4*vibration_step)){
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP == (4*vibration_step)){
                temp2 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (5*vibration_step)){
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (6*vibration_step)){
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP == (6*vibration_step)){
                temp3 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (7*vibration_step)){
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP < (8*vibration_step)){
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            }
            else if(DISC_STEP == (8*vibration_step)){
                temp4 = pos;
                RESULT = -(temp0-temp1)+(temp1-temp2)+(temp2-temp3)-(temp3-temp4);
                iq = 0;
            }
            else if(DISC_STEP < (10*vibration_step)){
```

```
            iq = 0;
        }
        else if(DISC_STEP == (10*vibration_step)){
            iq = 0;
            temp0 = pos;
            DISC_STEP = 0;
            if((RESULT >= -DETECTION_LEVEL)&&(RESULT <= DETECTION_LEVEL))
            {
                Iref = 1.2*Iref;            //no movement is detected
                PHI = PHI + (0.5*pi);
                TEST_COUNT = 0;
                if(Iref > Imax)
                {
                    Iref = Iref0;
                    STATUS = EMERGENCY_STOP;//TEST_FAILED;
                    PHI = 0.0;
                    iq = 0.0;
                }
            }
            else
            {
                TEST_COUNT++;               //increase test_count
                if(TEST_COUNT == 3)         //movement is detected three times
                {
                    if(STATUS == TEST_RM){
                        STATUS = ZERO_SEARCH;//EMERGENCY_STOP;
                        Iref = Iref0;
                        DISC_STEP=0;
                    }

                    PREVIOUS_RESULT = 0;    // reinitialize variables
                    TEST_COUNT = 0;
                    DPHI = 0.5*pi;
                    Iref = Iref0;
                }
            }
        }
        DISC_STEP++;
        // assign q-phi for motor
        IQ_RM=iq;
        PHI_RM=PHI;

        // assign outputs
        ENC_RESET_RM=0;
        AMP_ENABLED = 1;
        CONTROLLER_ENABLED=0;
      break;

//=============================================================================
// WAIT
//=============================================================================
    case WAIT:
      StartV++;
      if(StartV == 10)              // delay encoder reset
      {
          STATUS = HOMING; //EMERGENCY_STOP; reset encoder
```

```
            ENC_RESET_RM=0;
            StartV=0;
        }
      break;

//=============================================================================
// ZERO SEARCH LM1
//=============================================================================

        case ZERO_SEARCH:
            // get position of motor
            if(STATUS == ZERO_SEARCH_RM){
                pos = u_pos(0);
            }
            // generate pulses
            if(DISC_STEP < (1*vibration_step))
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            else if(DISC_STEP < (2*vibration_step))
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            else if(DISC_STEP == (2*vibration_step))
            {
                temp1 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (3*vibration_step))
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            else if(DISC_STEP < (4*vibration_step))
                iq = Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            else if(DISC_STEP == (4*vibration_step))
            {
                temp2 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (5*vibration_step))
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            else if(DISC_STEP < (6*vibration_step))
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            else if(DISC_STEP == (6*vibration_step))
            {
                temp3 = pos;
                iq = 0;
            }
            else if(DISC_STEP < (7*vibration_step))
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            else if(DISC_STEP < (8*vibration_step))
                iq = -Iref*sin(0.5*pi/vibration_step*DISC_STEP);
            else if(DISC_STEP == (8*vibration_step))
            {
                temp4 = pos;
                RESULT = -(temp0-temp1)+(temp1-temp2)+(temp2-temp3)-(temp3-temp4);
                iq = 0;
            }
            else if(DISC_STEP < (20*vibration_step))
                iq = 0.0;
            else if(DISC_STEP == (20*vibration_step))
            {
```

```
                temp0 = pos;
                if((RESULT >= -DETECTION_LEVEL)&&(RESULT <= DETECTION_LEVEL))
                {
                    Iref = 1.2*Iref;
                        if(Iref > Imax)
                        {
                            iq = 0.0;
                            PHI=PHI+0.5*pi;
                            if(STATUS == ZERO_SEARCH_RM)
                            {
                                PHI_RM=PHI;
                                ENC_RESET_RM=1;
                                STATUS = WAIT; // reset encoder
                            }

                            PREVIOUS_RESULT = 0.0;    // reinitialize variables
                            TEST_COUNT = 0;
                            DPHI = 0.5*pi;
                            Iref = Iref0;
                            DISC_STEP=1;
                        }
                    AMPLITUDE_COUNT = 0;
                }
                else
                {
                    AMPLITUDE_COUNT++;          //increase amplitude_count
                    if(((PREVIOUS_RESULT > 0) && (RESULT < 0)) || ((PREVIOUS_RESULT <↙
0) && (RESULT > 0)))
                    {
                        DPHI = DPHI*0.5;
                    }
                    PREVIOUS_RESULT = RESULT;
                    if(RESULT > 0)
                    {
                        PHI = PHI - DPHI;
                    }
                    else
                    {
                        PHI = PHI + DPHI;
                    }
                    if(AMPLITUDE_COUNT>30)
                    {
                        STATUS=ZERO_SEARCH_FAILED;
                        iq = 0.0;
                    }
                }
                iq = 0;
                DISC_STEP = 0;
            }
            DISC_STEP++;
            // assign q-phi for each motor
            if(STATUS == ZERO_SEARCH_RM){
                IQ_RM=iq;
                PHI_RM=PHI;
            }
```

```
            //ENC_RESET_RM=0;
            AMP_ENABLED = 1;
            CONTROLLER_ENABLED=0;

        break;

//=============================================================================
// HOMING
//=============================================================================

        case HOMING:
            if(HOMING_STATUS==0){                    // positive current positive movement
                if(DISC_STEP==1){
                    temp0=u_pos(0);
                    IQ_RM= Iref0;
                }
                else if(DISC_STEP < (10*vibration_step)){
                    IQ_RM= Iref0;
                }
                else if(DISC_STEP == (10*vibration_step)){
                    IQ_RM = 0;
                    temp1=u_pos(0)-temp0;
                    // if positive current leads to negative displacement
                    // change PHI with pi to reverse direction
                        if(temp1<-DETECTION_LEVEL){
                            PHI_RM+=pi;
                        }
                        if(fabs(temp1)<DETECTION_LEVEL){
                            STATUS=50;//HOMING_FAILED;
                        }
                    HOMING_STATUS++;
                    Direction = -1;
                    DISC_STEP=0;
                    IQ_RM = 0;
                }
                DISC_STEP++;
                ENC_RESET_RM=0;
            }

            //wait for LM1 to stand still after movement of LM1
            if(HOMING_STATUS==1){
                if( fabs(vest(0))<= 0.0001){
                    HOMING_STATUS++;
                    DISC_STEP=1;
                }
                IQ_RM=0.0;
                ENC_RESET_RM=0;
                temp1=101;
                temp2=100;
                Once = 0;
                DISC_STEP=1;
            }
            if(HOMING_STATUS==2){
                if(DISC_STEP==1){
                    temp0=u_pos(0);
```

```
                    IQ_RM= 0.8*Iref0*Direction;              //(0.25*(1-vest(0)))↙
*DirectionX;
                }
                else if(DISC_STEP < (10*vibration_step)){
                    IQ_RM= 0.72*Iref0*Direction;             //(0.25*(1-vest(0)))↙
*DirectionX;
                    if ((temp0 == temp1) && (temp2!=temp1)){ // het bereik bereikt,↙
rechterhoek
                        //PHI_RM+=pi/4*tau*u_pos(0);
                        ENC_RESET_RM=0;
                        //STATUS = READY;
                        IQ_RM = 0;
                        //ENC_RESET_RM=1;
                        temp2=temp1;
                        DISC_STEP=0;
                        HOMING_STATUS++;
                    }
                }
                else if(DISC_STEP == (10*vibration_step)){
                    IQ_RM= 0.72*Iref0*Direction;              //(0.24*(1-vest(0)))↙
*DirectionX;
                    temp1=temp0;
                    DISC_STEP=0;
                }
            DISC_STEP++;
            }
            //wait for LM1 to stand still after movement of LM1
            if(HOMING_STATUS==3){
                if( fabs(vest(0))<= 0.0001){
                    PHI_RM+=pi/tau*u_pos(0);
                    ENC_RESET_RM=1;
                    STATUS = READY;
                    DISC_STEP=0;
                }
                IQ_RM=0.0;
                //ENC_RESET_RM=0;
            }



            break;

//=============================================================================
// READY
//=============================================================================

        case READY:
            IQ_RM=0;//-vest(0);
            PHI_RM = PHI_RM;
            AMP_ENABLED = 1;
            DISC_STEP++;
                if (DISC_STEP == 10000){
                    ENC_RESET_RM=0;
                    CONTROLLER_ENABLED=1;
                    DISC_STEP=1;
                }
```

```
        break;

//=============================================================================
// EMERGENCY
//=============================================================================

case EMERGENCY:
            ENC_RESET_RM=0;
            AMP_ENABLED = 1;
            CONTROLLER_ENABLED=0;
            IQ_RM=-10*vest(0);
            if (AMP_ENABLED==1){
                if((fabs(vest(0))<DETECTION_LEVEL/STEP_SIZE)){//&& (fabs(vest(1))
<DETECTION_LEVEL/STEP_SIZE) ){
                    IQ_RM=0.0;
                    STATUS=EMERGENCY_STOP;
                }
            }
        break;

//=============================================================================
// EMERGENCY_STOP
//=============================================================================

        case EMERGENCY_STOP:
            IQ_RM=0.0;
            ENC_RESET_RM=0;
            AMP_ENABLED = 0;
            CONTROLLER_ENABLED=0;
        break;

    }

    //check current saturation
    if(IQ_RM>Imax){
        IQ_RM=Imax;
    }
    else if(IQ_RM<-Imax){
        IQ_RM=-Imax;
    }

    //generate three phase outputs

    y_irm(0)=IQ_RM*sin(pi/tau*u_pos(0)+PHI_RM);
    y_irm(1)=IQ_RM*sin(pi/tau*u_pos(0)+PHI_RM+2.0/3.0*pi);



    // other outputs
    y_enc_reset(0)=ENC_RESET_RM;

    y_controller_enable = CONTROLLER_ENABLED;
    y_status = STATUS;
    y_amp_enable = AMP_ENABLED;
    y_debug = debug;
```

```
    // phase angle
    y_phi(0)=PHI_RM;
}

#define MDL_UPDATE
#if defined(MDL_UPDATE)
static void mdlUpdate(SimStruct *S, int_T tid)
{
}
#endif

#define MDL_DERIVATIVES
#if defined(MDL_DERIVATIVES)
static void mdlDerivatives(SimStruct *S)
{
    real_T            *dx = ssGetdX(S);
    real_T            *xc = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

    int i;

    for(i=0;i<NCSTATES;i++){
        dx[i]=Kgain*(u_vel(i)-vest(i));
    }
}
#endif

static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

APPENDIX

B Test data Assembléon

Assembléon

1. Test Configuration

1.1 CPR configuration

Item	ID	
XY Robot	PA1118/00	DC 1675
PHS	PA2800/00	DC 7515
Placement Controller Radisys (Reference) Prodrive	 PA1800/00 PA1800/01	 DC pr2-016

ACS Servo Parameters

	X	Y	Phi	Z
SLPKP	202	346	350	300
SLVKP	565	500	320	694
SLVKI	265	200	569	247
SLVSOF	600	600	1500	508
SLVSOFD	0.707	0.707	0.5	0.9
SLVNFRQ	590	881	2200	n.a.
SLVNWD	51	50	300	n.a
SLVNATT	2.15	9.00	13	n.a.

1.2 SPR configuration

Item	ID	
XY Robot	PA1118/15	DC 0856
PHS	PA2800/00	DC 1931
Placement Controller Radisys (Reference) Prodrive	 PA1800/00 PA1800/01	 DC pr2-016

ACS Servo Parameters

	X	Y	Phi	Z
SLPKP	202	346	350	300
SLVKP	565	500	320	694
SLVKI	265	200	569	247
SLVSOF	1000	900	1500	508
SLVSOFD	0.55	0.44	0.5	0.9
SLVNFRQ	590	881	2200	n.a.
SLVNWD	51	50	300	n.a
SLVNATT	2.15	9	13	n.a.

CLASS NO.					
NAME: Martijn Opheij	SUPERS.	14 SH	SH - 2	10	A4
DATA FMT	PROGR. MS-WORD 97		PLATF WINDOWS NT		
DC CHECK	DATE: 06-06-20		© Assembléon Netherlands B.V.		

Assembléon

2. Procedure

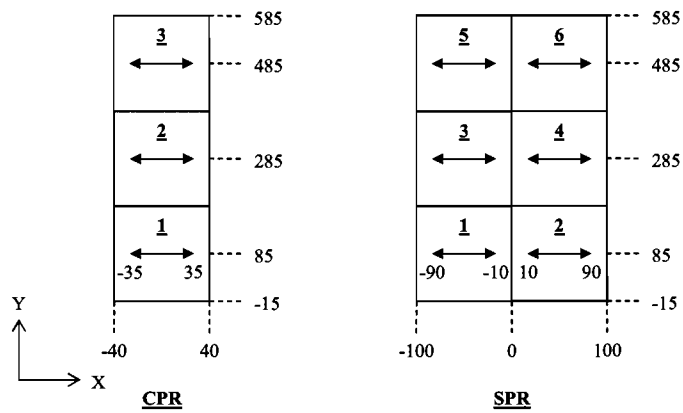
2.1 FRF Measurement

1) Reference FRF Measurement (ACS Controller)
- Update FW version of controller to ACS FW5.0 (in order to use FRF tool)
- Load duurtest.prg (loading parameters, homing scripts)
- Start Buffer#4 (initialise parameters and home axis).
- Perform FRF measurements as described below, using ACS FRF Tool.

2) Prodrive FRF Measurement
3) Compare results.

2.1.1 X-Axis

Measurements X



CPR SPR

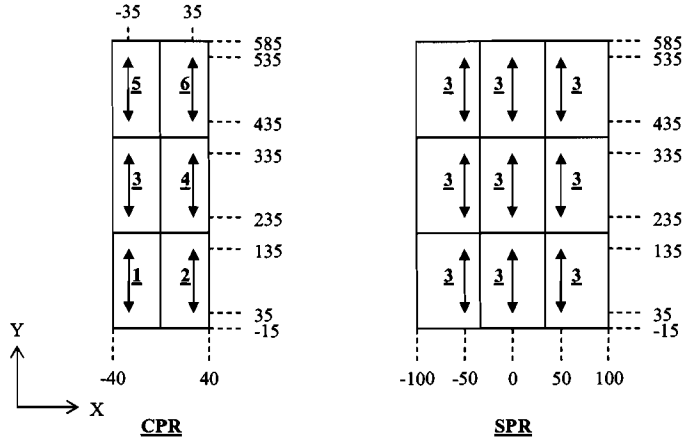
Motion Settings		
axis	**Item**	**Value**
X	V [mm/s]	10
	A [mm/s^2]	5000
	J [mm/s^3]	250E3
	Dwell [msec]	0
Y	Loop	Closed
Phi	Loop	Open
Z	Loop	Open

ACS Measurement Settings	
Item	**Value**
Min. freq. [Hz]	10
Max. freq. [Hz]	4000
Freq. / Decade [-]	50
Excitation Current [% of max]	10

CLASS NO.								
NAME: Martijn Opheij		SUPERS.		14 SH	SH - 3	10		A4
DATA FMT		PROGR. MS-WORD 97			PLATF WINDOWS NT			
DC	CHECK	DATE: 06-06-20			© Assembléon Netherlands B.V.			

2.1.2 Y-Axis

Measurements Y



Motion Settings		
axis	**Item**	**Value**
X	Loop	Closed
Y	V [mm/s]	10
	A [mm/s^2]	16000
	J [mm/s^3]	1.25E6
	Dwell [msec]	0
Phi	Loop	Open
Z	Loop	Open

ACS Measurement Settings	
Item	**Value**
Min. freq. [Hz]	10
Max. freq. [Hz]	4000
Freq. / Decade [-]	50
Excitation Current [% of max]	8

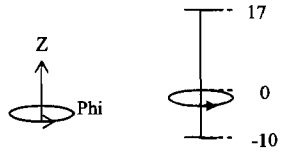
CLASS NO.							
NAME: Martijn Opheij	SUPERS.	14 SH	SH - 4	10		A4
DATA FMT	PROGR. MS-WORD 97		PLATF WINDOWS NT			
DC CHECK DATE: 06-06-20	© Assembléon Netherlands B.V.					

4022 599 00151

Assembléon

2.1.3 Phi-Axis

Measurement Phi

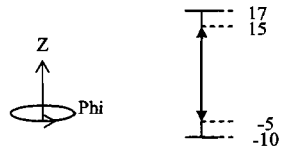


Motion Settings		
axis	**Item**	**Value**
X	Loop	Open
Y	Loop	Open
Phi	V jog [rad/s]	+6 CPR +12 SPR[1]
Z	Loop	Closed

ACS Measurement Settings	
Item	**Value**
Min. freq. [Hz]	10
Max. freq. [Hz]	4000
Freq. / Decade [-]	50
Excitation Current [% of max]	15

2.1.4 Z-Axis

Measurement Z



Motion Settings		
axis	**Item**	**Value**
X	Loop	Open
Y	Loop	Open
Phi	Loop	Closed
Z	V [mm/s]	20
	A [mm/s^2]	40E3
	J [mm/s^3]	15E6
	Dwell [msec]	0

ACS Measurement Settings	
Item	**Value**
Min. freq. [Hz]	10
Max. freq. [Hz]	4000
Freq. / Decade [-]	50
Excitation Current [% of max]	12

[1] In order to perform a reliable FRF on the SPR configuration the jog speed had to be increased. Most likely the PHS involved is suffering more friction than the one on the CPR. Friction causes unreliable FRF measurements at lower frequencies.

CLASS NO.					
NAME: Martijn Opheij	SUPERS.	14 SH	SH - 5	10	A4
DATA FMT	PROGR. MS-WORD 97		PLATF WINDOWS NT		
DC CHECK DATE: 06-06-20	© Assembléon Netherlands B.V.				

Assembléon

2.2 Settle behaviour

1) Perform trajectory as described below.
2) Measure settle behaviour on encoder level and controller output for both ACS and Prodrive controller.
3) Compare results.

Motion parameters				
	X	**Y**	**Phi**	**Z**
Velocity	500 [mm/s]	1400 [mm/s]	62.8 [rad/s]	500 [mm/s]
Acc. = Dec.	5000 [mm/s^2]	16E3 [mm/s^2]	1885 [rad/s^2]	40E3 [mm/s^2]
Jerk	250E3 [mm/s^3]	1.25E6 [mm/s^3]	314E3 [rad/s^3]	15E6 [mm/s^3]

Test trajectories												
	X [mm]			**Y [mm]**			**Phi [rad]**			**Z [mm]**		
	Pstart	Pend	Stroke	Pstart	Pend	Stroke	Pstart	Pend	Stroke	Pstart	Pend	Stroke
Traject 1	-35	-34	+1	0	+1	+1	0	+1	+1	+15	+10	-5
Traject 2	-35	-25	+10	0	+10	+10	0	+6.28	+6.28	+15	-5	-20
Traject 3	-35	+35	+70	0	+50	+50	0					
Traject 4	-90	+90	+180	0	+400	+400	0					
Precond.	Y enabled at Y=0			X enabled at X=0			None			None		

ACS Scope Settings		
Channel	**Signal**	**Description**
1	PE	Position Error
2	DOUT	DAC Output
3	RVEL	Reference Velocity
4	GPHASE	Generator Phase
Trigger	RVEL>1 @ 25% of time base	

CLASS NO.							
NAME: Martijn Opheij	SUPERS.	14 SH	SH - 6	10		A4
DATA FMT	PROGR. MS-WORD 97		PLATF WINDOWS NT			
DC CHECK	DATE: 06-06-20		© Assembléon Netherlands B.V.			

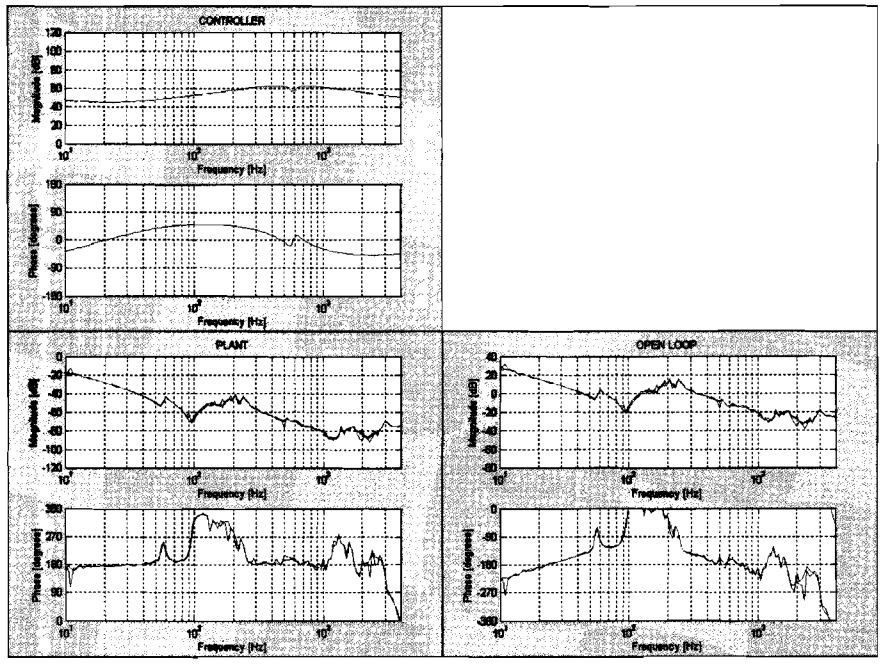
Assembléon

3. Results

3.1 FRF Measurement

3.1.1 X-Axis

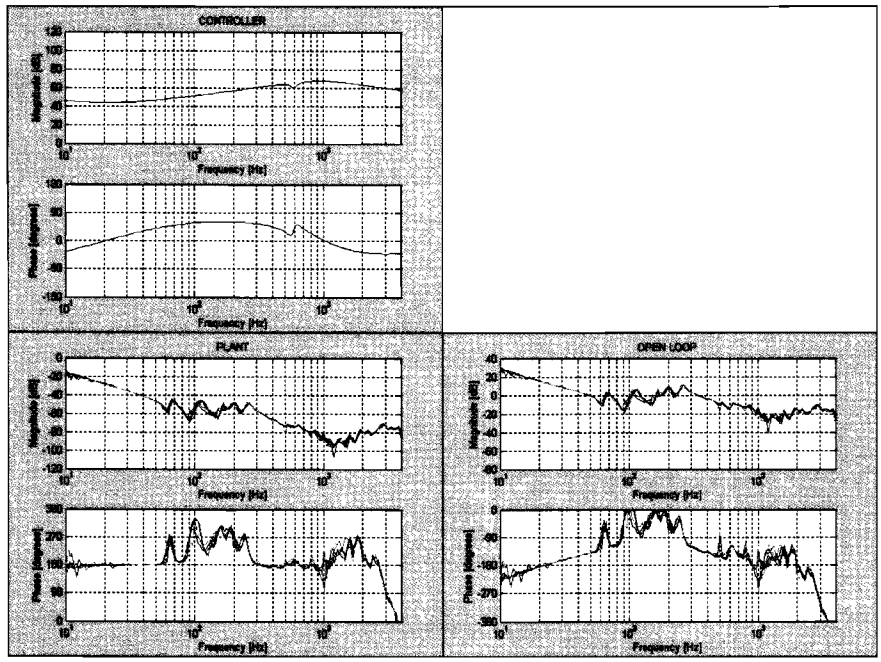
3.1.1.1 CPR X



CLASS NO.							
NAME: Martijn Opheij	SUPERS.	14 SH	SH - 7	10		A4	
DATA FMT	PROGR. MS-WORD 97		PLATF WINDOWS NT				
DC	CHECK	DATE: 06-06-20	© Assembléon Netherlands B.V.				

4022 599 00151

3.1.1.2 SPR X



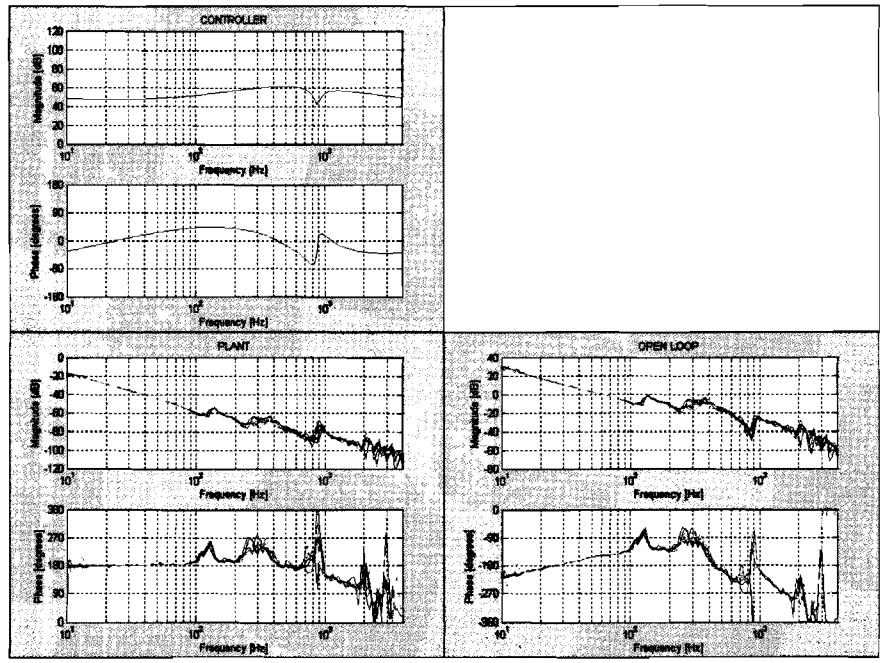
CLASS NO.					
NAME: Martijn Opheij	SUPERS.	14 SH	SH - 8	10	A4
DATA FMT	PROGR. MS-WORD 97		PLATF WINDOWS NT		
DC CHECK	DATE: 06-06-20		© Assembléon Netherlands B.V.		

4022 599 00151

Assembléon

3.1.2 Y-Axis

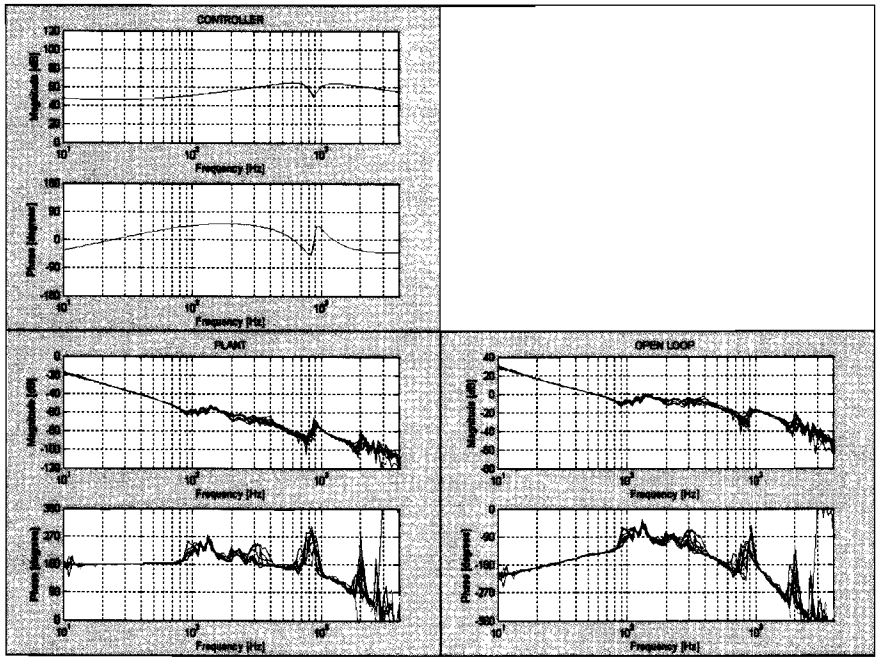
3.1.2.1 CPR Y



CLASS NO.									
NAME: Martijn Ophelj	SUPERS.		14 SH	SH	- 9	10		A4
DATA FMT	PROGR. MS-WORD 97			PLATF WINDOWS NT				
DC	CHECK	DATE: 06-06-20	© Assembléon Netherlands B.V.					

Assembléon

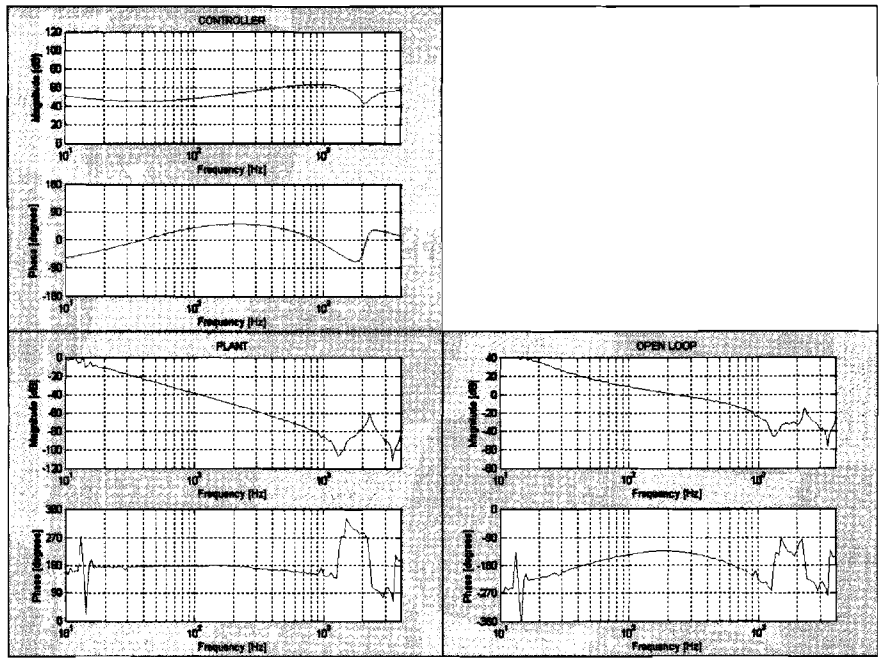
3.1.2.2 SPR Y



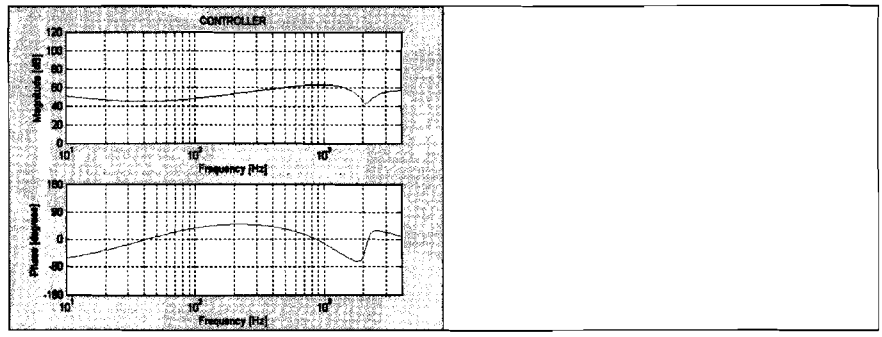
3.1.3 PHI-Axis

3.1.3.1 CPR PHI

CLASS NO.									
NAME: Martijn Opheij		SUPERS.		14 SH	SH - 10	10		A4	
DATA FMT		PROGR. MS-WORD 97			PLATF WINDOWS NT				
DC	CHECK	DATE: 06-06-20	© Assembléon Netherlands B.V.						

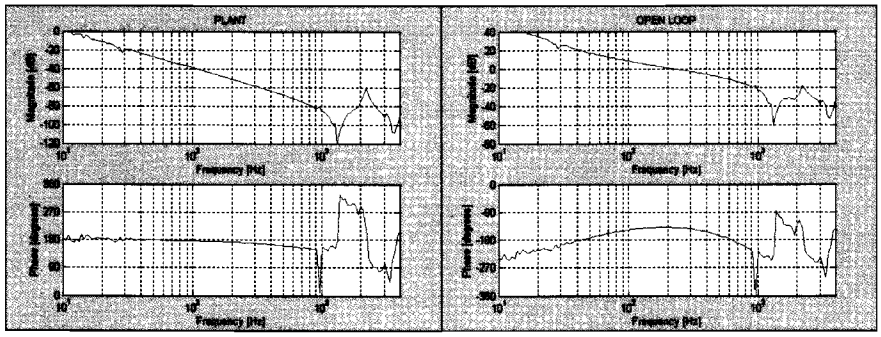


3.1.3.2 SPR PHI



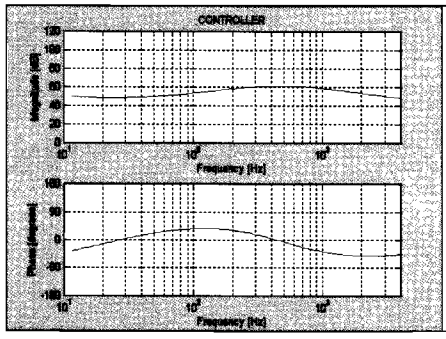
CLASS NO					
NAME: Martijn Opheij	SUPERS	14 SH	SH - 11	10	A4
DATA FMT	PROGR MS-WORD 97		PLATF WINDOWS NT		
DC CHECK	DATE: 06-06-20		© Assembléon Netherlands B.V.		

4022 599 00151

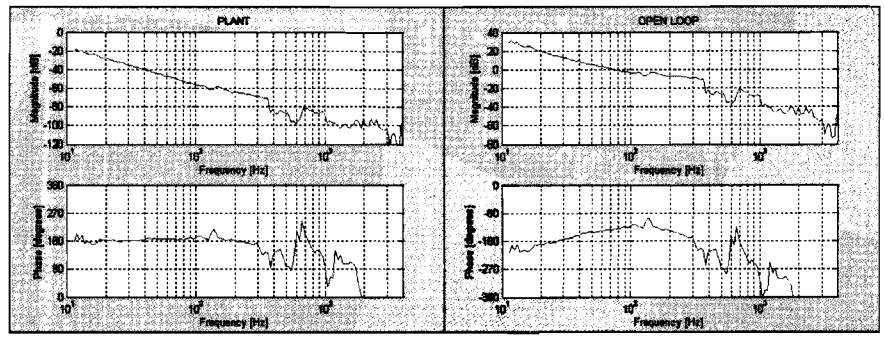


3.1.4 Z-Axis

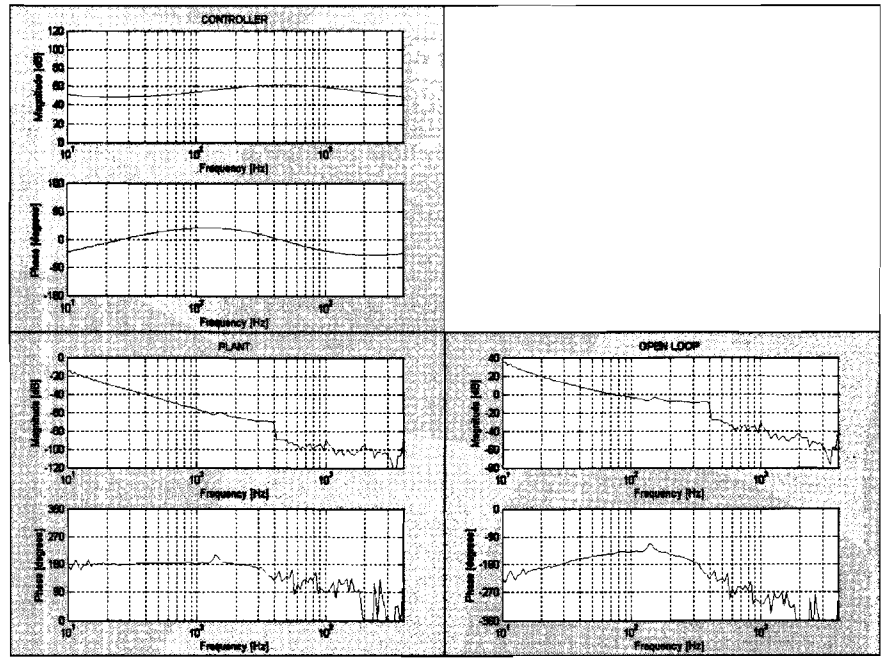
3.1.4.1 CPR Z



CLASS NO.									
NAME: Martijn Opheij		SUPERS.		14 SH	SH - 12	10		A4	
DATA FMT		PROGR. MS-WORD 97			PLATF WINDOWS NT				
DC	CHECK	DATE: 06-06-20			© Assembléon Netherlands B.V.				



3.1.4.2 SPR Z



Assembléon

CLASS NO							
NAME: Martijn Opheij	SUPERS	14 SH	SH - 13	10		A4
DATA FMT	PROGR MS-WORD 97		PLATF WINDOWS NT			
DC	CHECK	DATE: 06-06-20	© Assembléon Netherlands B.V.			

4022 599 00151