

# **AS-0.3200 Automation- and systems technology project work**

**S09-19 Exercise platform for the Wireless Automation course (AS-74.3199)**

**Report**

**29.4.2009**

Ossi Kaltiokallio 62854C

[ossi.kaltiokallio@tkk.fi](mailto:ossi.kaltiokallio@tkk.fi)

Päivikki Repo 53796J

[srepo@cc.hut.fi](mailto:srepo@cc.hut.fi)

Instructors Lasse Eriksson and Maurizio Bocca

## Table of Contents

1. Introduction .....	2
2. Planning the control.....	3
2.1 Optimal Control.....	7
2.2 Traditional PID-controller .....	8
2.3 Enhanced PID-controller, PIDplus .....	10
3. The systems simulation results .....	13
4. Modernization of Halvari .....	21
4.1 Renewed parts .....	21
4.2 Communication and connections in the system.....	22
4.3 Calibrating the system .....	25
4.4 Graphical interface and Simulink-model.....	26
5. From wired to wireless .....	28
5.1 Joystick control .....	28
5.2 PIDPlus control.....	30
6. The exercise for the 2 <sup>nd</sup> tutorial and the solution.....	32
References .....	33
Appendix A .....	34
Appendix B .....	36
Appendix D.....	38
Appendix E .....	39
Appendix F .....	40
Appendix G.....	42

## 1. Introduction

The goal of this project work is to create an exercise for the course AS-74.3199 Wireless Automation. The system used in the exercise is a ball balancing system called Halvari which has been used in the course AS-0.2230 Automaatio- ja systeemiteknikan laboratoriotyöt to demonstrate how optimal control works. Halvari has a cart that is set upon a rail and the cart can move from side to side. Upon the cart there is an arch and on that arch a ball as can be seen from fig. 1. The primary goal of Halvari is to balance the ball in the middle of the arch by moving the cart back and forward with the aid of an electric motor. Secondary goal of the system is to keep also the cart in the middle of the rail. The system measures the place of the ball and of the cart. From these it calculates the speed of the ball and of the cart as derivatives. Halvari can be controlled by a manual joystick or a computer.

The project work was divided into two parts: designing the control and creating the hardware interface. The control design includes creating a Simulink model of Halvari and testing it with optimal control. Based on optimal control a traditional PID controller is designed and tested. The controllers are also compared to each other. The main task is to replace PID with a PIDplus which is an enhanced version of the traditional one. In wireless automation failures in the communication between the controller and the actuator can happen easily. The disappearance of measurement or control data is called packet loss. When packet loss happens traditional PID gives a poor dynamic response. PIDplus is used because it can cope with packet loss to a certain point. The idea is that the controller gets the previous executed control signal from the actuator when a new measurement is not received. This means that a new control signal is calculated only when a data packet is transmitted successfully and if not the old signal is used.



**Figure 1.** The Halvari system.

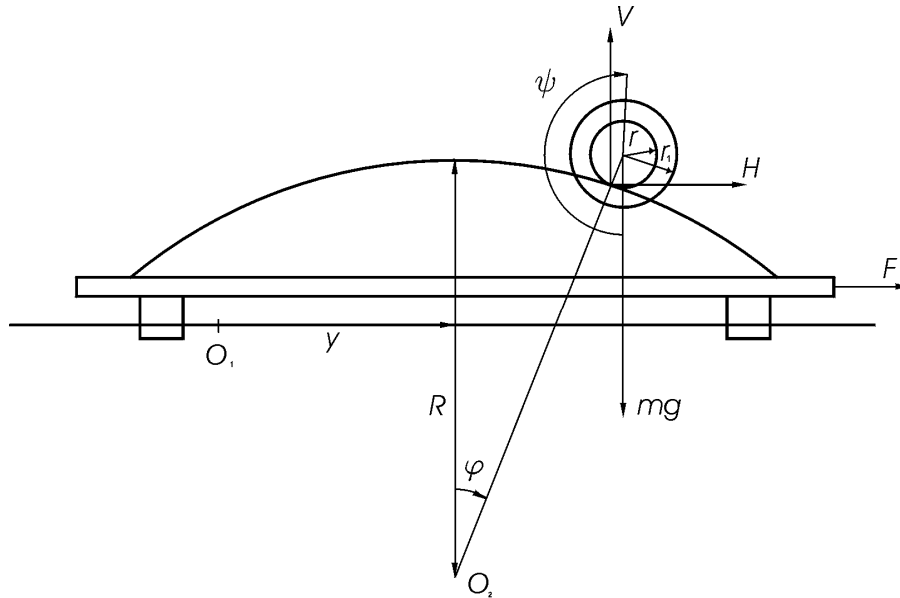
## 2. Planning the control

At first a Simulink model of Halvari was created. Halvari can be described with two nonlinear differential equations [1] and [2].

$$\ddot{y}(M + m) = -m(R + r)(\ddot{\varphi} \cos \varphi - \dot{\varphi}^2 \sin \varphi) + F \quad (1)$$

$$\ddot{\varphi} \frac{J(R + r)}{r} = mr(R + r)(-\ddot{\varphi} \sin^2 \varphi - \dot{\varphi}^2 \sin \varphi \cos \varphi) + mrg \sin \varphi + Mr\ddot{y} \cos \varphi - Fr \cos \varphi \quad (2)$$

In the equations  $y$  means the position of the cart and  $\varphi$  the angle of the ball as can be seen from figure 2. The angle gets values from -0.2 radians to 0.2 radians.



**Figure 2.** A chart representing Halvari.  $F$  is the force used for control,  $y$  is the position of the cart the zero point being in the middle of the rail and  $\varphi$  is the angle of the ball also from the middle. [1]

The equations have to be modified in order to do a Simulink model.

$$\ddot{y} = k_1 \dot{\varphi} \cos \varphi + k_2 \dot{\varphi}^2 \sin \varphi + k_3 F \quad (3)$$

$$\ddot{\varphi} = \frac{R + r}{(R + r) + k_4 \sin^2 \varphi} [k_5 \dot{\varphi}^2 \sin \varphi \cos \varphi + k_6 \sin \varphi + k_7 \ddot{y} \cos \varphi + k_8 \cos \varphi F] \quad (4)$$

Coefficients used in (3) and (4)

$$k_1 = -\frac{m(R+r)}{M+m} \quad (5)$$

$$k_2 = \frac{m(R+r)}{M+m} \quad (6)$$

$$k_3 = \frac{1}{M+m} \quad (7)$$

$$k_4 = \frac{mr^2}{J} \quad (8)$$

$$k_5 = -\frac{mr^2}{J} \quad (9)$$

$$k_6 = \frac{mr^2 g}{J(R+r)} \quad (10)$$

$$k_7 = \frac{Mr^2}{J(R+r)} \quad (11)$$

$$k_8 = -\frac{r^2}{J(R+r)} \quad (12)$$

Coefficients use these constants that are derived from the physical information of Halvari seen also in figure 2. [1]

$M = 4kg$ , mass of the cart without the ball.

$m = 0.7kg$ , mass of the ball.

$J = 0.175 \cdot 10^{-3} kgm^2$ , inertia of the ball.

$R = 0.5m$ , radius of the arch.

$r = 0.2m$ , rolling radius of the ball.

$g = 9.81m^2/s$ , acceleration of gravity.

The system has four states that can be chosen as follows

$$x_1 = y$$

$$x_2 = \dot{y} = \dot{x}_1$$

$$x_3 = \varphi$$

$$x_4 = \dot{\varphi} = \dot{x}_3$$

Now the equations can be written to their final form from which the Simulink model can be designed.

$$\dot{x}_2 = k_1 \dot{x}_4 \cos x_3 + k_2 x_4^2 \sin x_3 + k_3 F \quad (13)$$

$$\dot{x}_4 = \frac{R+r}{(R+r) + k_4 \sin^2 x_3} \left[ k_5 x_4^2 \sin x_3 \cos x_3 + k_6 \sin x_3 + k_7 \dot{x}_2 \cos x_3 + k_8 \cos x_3 F \right] \quad (14)$$

Force is used as the control variable. Optimal control is calculated by the following equation

$$F = K_1 y + K_2 \dot{y} + K_3 \phi + K_4 \dot{\phi}. \quad (15)$$

The gains of optimal control are calculated in Matlab. For that the system has to be linearized and a state-space model calculated. The following assumptions are valid when  $\varphi$  is small

$$\sin \varphi = \varphi.$$

$$\sin^2 \varphi = 0.$$

$$\cos \varphi = 1.$$

$$\dot{\varphi}^2 = 0.$$

With these assumptions (1) becomes

$$\ddot{y} = -\frac{m(R+r)}{(M+m)} \ddot{\varphi} + \frac{1}{(M+m)} F. \quad (16)$$

Respectively (2) becomes

$$\ddot{\varphi} \frac{J(R+r)}{r} = mrg\varphi + Mr\ddot{y} - Fr. \quad (17)$$

Now (16) can be placed to (17)

$$\ddot{\varphi} = \frac{mgr^2(M+m)}{(R+r)((M+m)J + Mmr^2)} \varphi - \frac{mr^2}{(R+r)((M+m)J + Mmr^2)} F \quad (18)$$

and (18) to (16)

$$\ddot{y} = -\frac{m^2 r^2 g}{(M+m)J + Mmr^2} \varphi + \frac{mr^2 + J}{(M+m)J + Mmr^2} F. \quad (19)$$

If the coefficients are marked as follows (18) and (19) can be written in a shorter way

$$a = -\frac{m^2 r^2 g}{(M + m)J + Mmr^2} \quad (20)$$

$$b = \frac{mr^2 + J}{(M + m)J + Mmr^2} \quad (21)$$

$$c = \frac{mgr^2(M + m)}{(R + r)((M + m)J + Mmr^2)} \quad (22)$$

$$d = \frac{mr^2}{(R + r)((M + m)J + Mmr^2)} \quad (23)$$

$$\ddot{y} = a\varphi + bF \quad (24)$$

$$\ddot{\varphi} = c\varphi + dF. \quad (25)$$

Now the system is linear and it can be converted into a state-space model. The states ( $x_1 - x_4$ ) have been listed earlier and from them and (24) and (25) the following can be derived

$$\dot{x}_1 = \dot{y} = x_2 \quad (26)$$

$$\dot{x}_2 = \ddot{y} = ax_3 + bF \quad (27)$$

$$\dot{x}_3 = \dot{\varphi} = x_4 \quad (28)$$

$$\dot{x}_4 = \ddot{\varphi} = cx_3 + dF. \quad (29)$$

The control vector and the output vector are

$$u = F \quad (30)$$

$$y_1 = y = x_1 \quad (31)$$

$$y_2 = \varphi = x_3. \quad (32)$$

A state-space model is determined by the following equations

$$\dot{x} = Ax + Bu \quad (33)$$

$$y = Cx + Du. \quad (34)$$

From (26) – (34) matrices A, B, C and D of the state-space model of Halvari can be determined

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & c & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ b \\ 0 \\ d \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

## 2.1 Optimal Control

In the m-file `paramtrit.m` the physical constants of Halvari are determined and the needed coefficients computed. `paramtrit.m` has to be run before using any Simulink model including Halvari. In `paramtrit.m` the gains  $K$  of optimal control (also seen in (15)) are computed from the discrete state-space model. Discrete model can be created from the continuous one in Matlab as follows [2]

```
sys = ss(A,B,C,D1);
sysd = c2d(sys,sample_t);
[Ad,Bd,Cd,Dd] = ssdata(sysd);
```

The weights  $K$  are calculated with the command

```
[K,s,e] = dlqr(Ad,Bd,Qs,Rs);
```

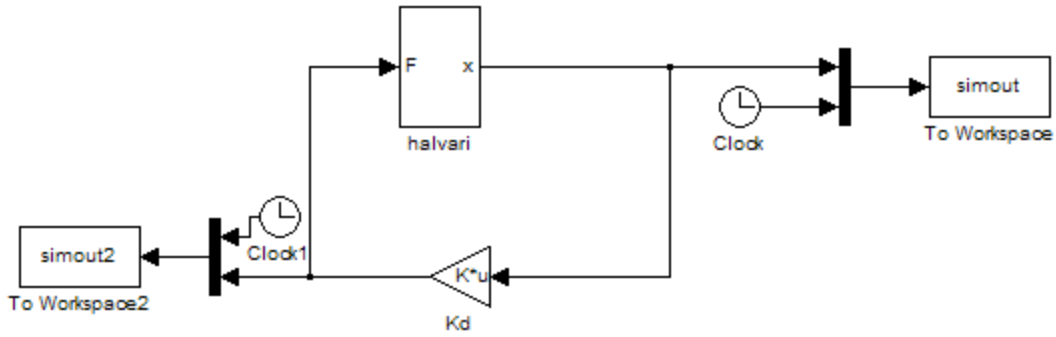
Sample time has an impact on the discretization results and thus on the vector  $K$ . The sample time is selected to be 0.05 s.  $Qs$  and  $Rs$  are given beforehand

$$Qs = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \quad \text{and} \quad Rs = 0.01.$$

The gains of the optimal control are

$$K = [-23.9663 \quad -27.9799 \quad -205.1571 \quad -58.6860]$$

A Simulink model of Halvari (`halvari.mdl`) can be seen in the Appendix A. In Appendix B the whole m-file `paramtrit.m` is represented. Optimal control is implemented in the Simulink model `halvari_optimi.mdl` that is presented in figure 3. Model includes `halvari.mdl` as a subsystem. By choosing the gains  $K$  the system can be adjusted but the best possible values are calculated above. The most important aim is to keep the ball and secondly the cart in the middle. That is why gain  $K_3$  is larger than  $K_1$ . Before simulating initial values for  $x_1$  and  $x_3$  have to be given. This represents that neither the ball nor the cart are in their reference value in the beginning and it is the controller's job to put them there. Initial values can be set in the integration blocks in `halvari` as initial condition. Initial condition for  $x_1$  is set to be 0.1 and for  $x_3$  0.05 and they are kept the same in all the simulations.



**Figure 3.** halvari\_optimi.mdl. The Simulink model of optimal control. Halvari is included as a subsystem and  $F$  is calculated in  $K_d$  as in (15).

## 2.2 Traditional PID-controller

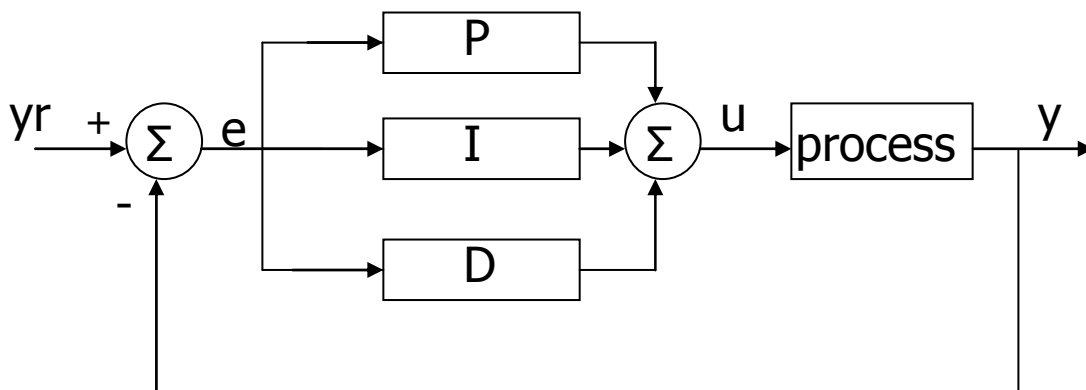
Traditional PID (proportional-integral-derivative) is described by (35) or (36). The schematic of the controller is presented in figure 4.

$$u(t) = K_p \left( e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) \quad (35)$$

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (36)$$

where

$$K_i = \frac{K_p}{T_i} \text{ and } K_d = K_p T_d.$$



**Figure 4.** Schematic of a traditional PID-controller

The input of the controller is an error signal that is calculated from a reference signal and the measured output of the system.

$$e(t) = y_r(t) - y(t) \tag{37}$$

PID controller can be tuned by choosing the gain values ( $K_p, K_i, K_d$ ).

For the Simulink model `halvari_pid.mdl` blocks PID Controller (with Approximate Derivative) are used. The schematic of `halvari_pid.mdl` is presented in fig. 5. Naturally Halvari stays the same and `paramtrit.m` has to be run before simulation. The model has two controllers: one for controlling the position of the cart and one for controlling the angle of the ball. The position or more precisely the angle of the ball is again the most important variable to be controlled. The outputs of the controllers are summed in order to get  $F$ . The reference signal of both is zero. The controller parameters are found by experimenting and they are listed in table 1. It can be seen that there is no need for integration so there has to be some in the system itself.

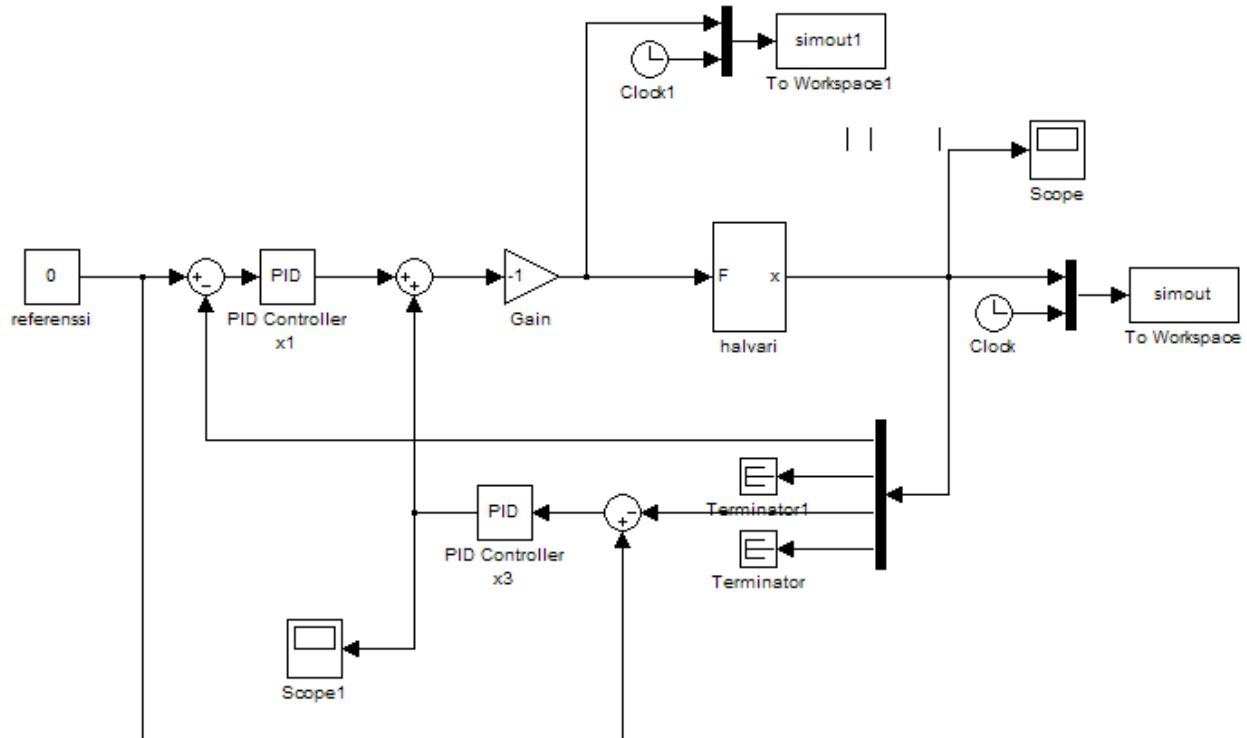


Figure 5. `halvari_pid.mdl`. The Simulink model of PID control of Halvari.

Table 1. Parameters for the PID controllers.

parameter	controller x1	controller x3
P = $K_p$	0,1	100
I = $K_i$	0	0
D = $K_d$	5	20

## 2.3 Enhanced PID-controller, PIDplus

Because of packet loss, traditional PID gives a poor dynamic response. If the control is executed only when a measurement is received there can be a delay in control response to setpoint changes. To avoid problems a PIDplus controller is implemented. The idea of PIDplus is that it keeps the previous control signal if a new measurement does not arrive. So control is executed periodically no matter how many packets get lost on the way. Fig. 6 represents the idea of PIDplus. [3]

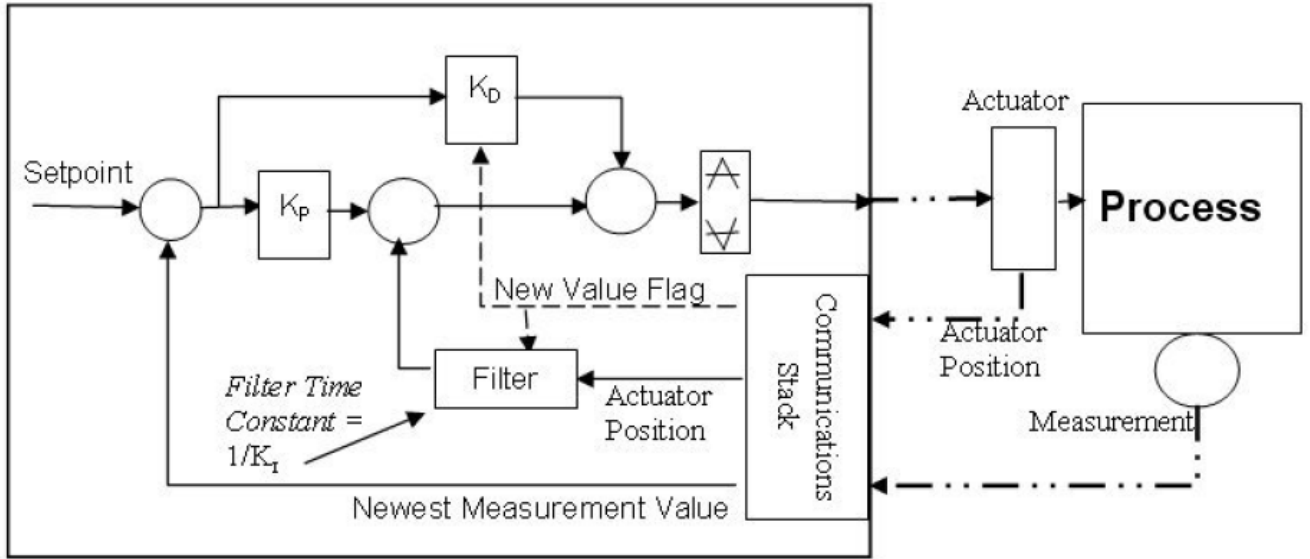


Figure 6. A chart of a PIDplus where the integration part is replaced by a filter. [3]

The main difference of PID and PIDplus is in the integration part that will be replaced by a filter. The filter output is calculated as follows [3]

$$F_N = F_{N-1} + (O_{N-1} - F_{N-1}) \left(1 - e^{-\frac{\Delta T}{T_{reset}}}\right) \quad (38)$$

$F_N$  = new filter output

$F_{N-1}$  = filter output for last execution

$O_{N-1}$  = controller output for last execution

$\Delta T$  = elapsed time since a new value was communicated

$$T_{reset} = \frac{P}{I}$$

The derivative part is also replaced as follows [3]

$$O_D = K_D \frac{e_N - e_{N-1}}{\Delta T}, \quad (39)$$

where

$e_N$  = current error

$e_{N-1}$  = last error

$\Delta T$  = elapsed time since a new measurement was communicated

$O_D$  = controller derivative term

Because the reference signal is zero there is a connection

$$e_N = -y_N \quad (40)$$

$y_N$  is the controlled variable meaning the position of the ball (angle  $\varphi$ ) or of the cart (coordinate  $y$ ). A filter is added to the derivative part. The equation for the filter is

$$e_{f,N} = \frac{T_f}{\Delta T + T_f} e_{f,N-1} + \frac{\Delta T}{\Delta T + T_f} e_N \quad (41)$$

where

$e_{f,N}$  = new filter output

$e_{f,N-1}$  = filter output last execution

$T_f$  = filter time-constant

PIDplus is implemented in `halvari_pidplus_packetloss.mdl` which can be seen in figure 7 and the schematic of PIDplus is presented in Appendix C. `halvari_pidplus_packetloss.mdl` corresponds to `halvari_pid.mdl` but traditional PID is replaced by PIDplus. The parameters used are the same as in optimal control and are presented in table 2. Also a little bit of integration is added just to see how the filter works.

Table 2. Parameters for PIDplus.

parameter	controller x1	controller x3
P	23,97	205,16
I	0,1	0,1
D	27,98	58,69

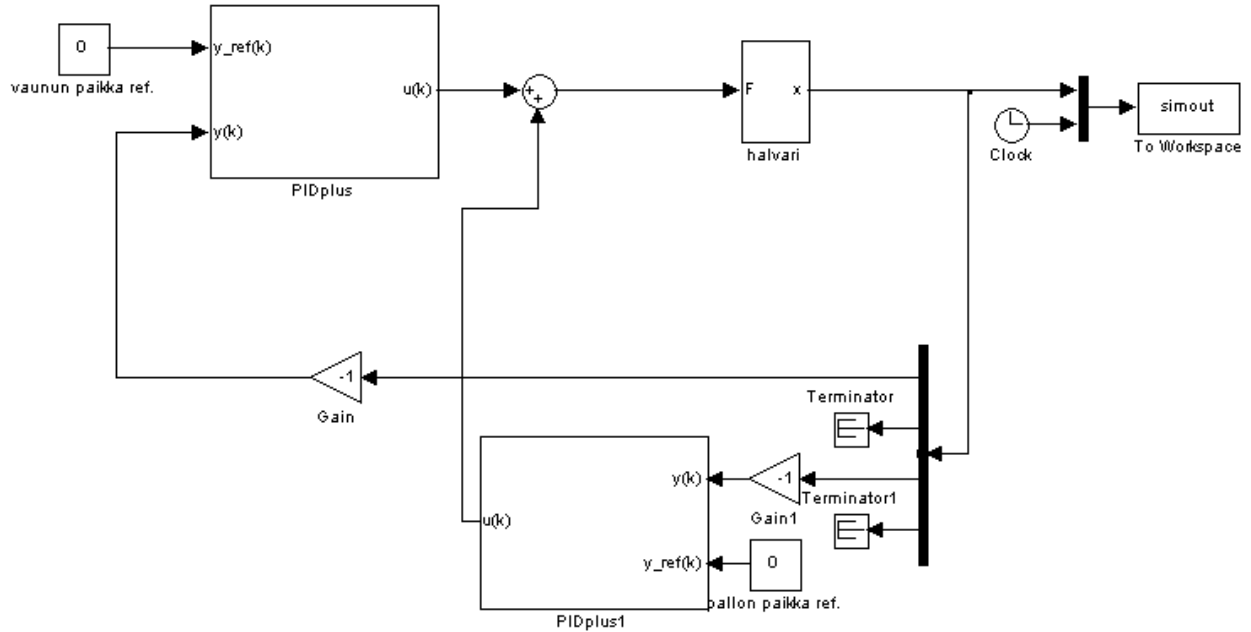


Figure 7. halvari\_pidplus\_packetloss.mdl. The Simulink model of PIDplus control of Halvari.

The reason why the same parameters work in optimal control and PIDplus can be found by comparing (15) and (36) and remembering that the reference value is always zero. In optimal control  $K_1$  and  $K_3$  multiply  $\varphi$  and  $y$  and so does the P-terms of the PIDplus controllers. Correspondingly  $K_2$  and  $K_4$  are gains for the derivatives of the signals. In PID control (and also in PIDplus) D-term is a coefficient for the derivative of the error. If the reference is zero the error corresponds the measured signal itself and the parameters in optimal and PID control are the same.

### 3. The systems simulation results

In figure 8 the results of optimal control are presented. Signals and their derivatives are drawn in the upper plot and  $F$  in the lower. It is seen that optimal control works quite fast and the system reaches the desired values under 3 seconds.

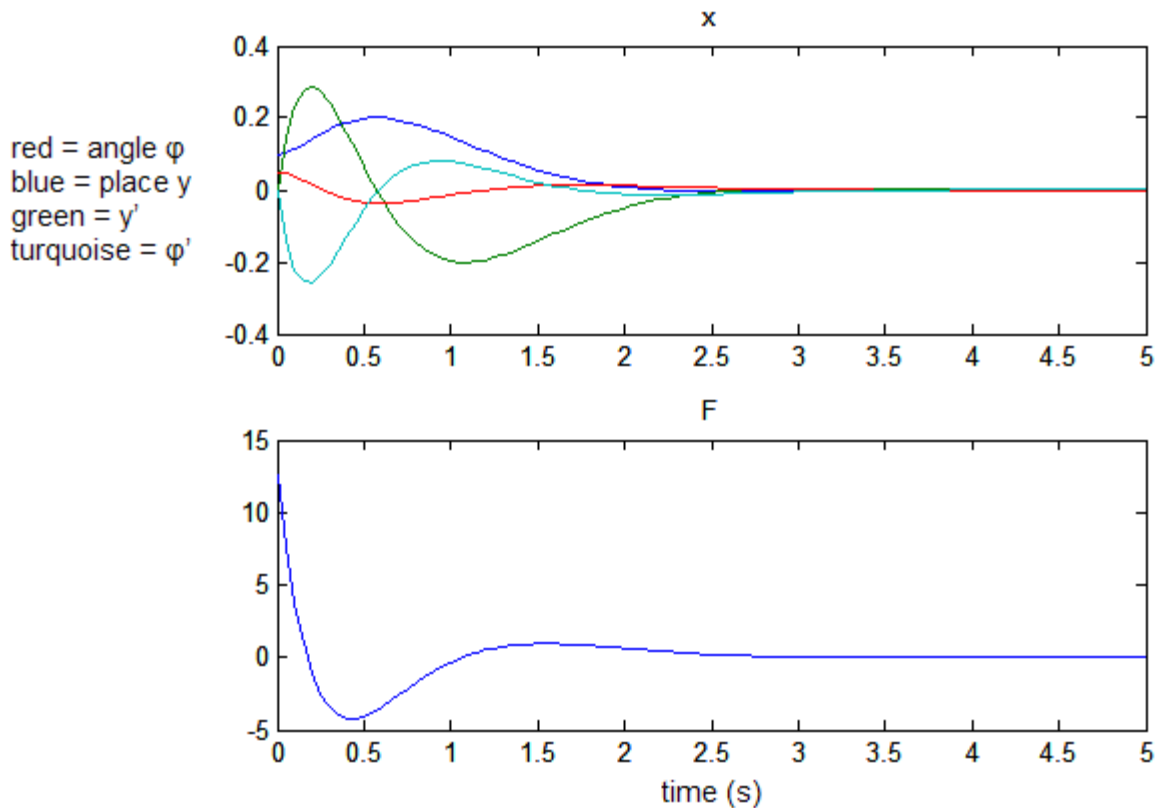


Figure 8. Signals in optimal control.

In figure 9 the results of PID control with the parameters in table 1 are presented. From the figure it can be seen that PID with these parameters is slower than optimal control and it takes about 3,5 seconds for the signals to settle down to the reference values. PID is tested also with the parameters from optimal control. This can be seen in figure 10. With these parameters the signals settle down in less than 3 seconds.

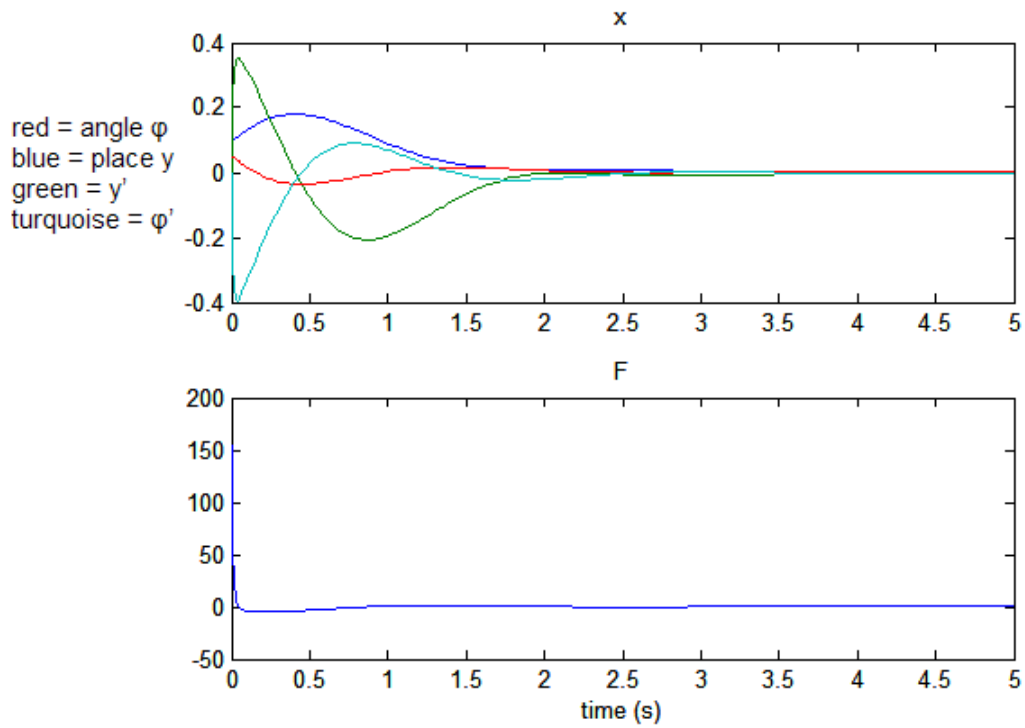


Figure 9. Signals in PID control when the parameters in table 1 are used.

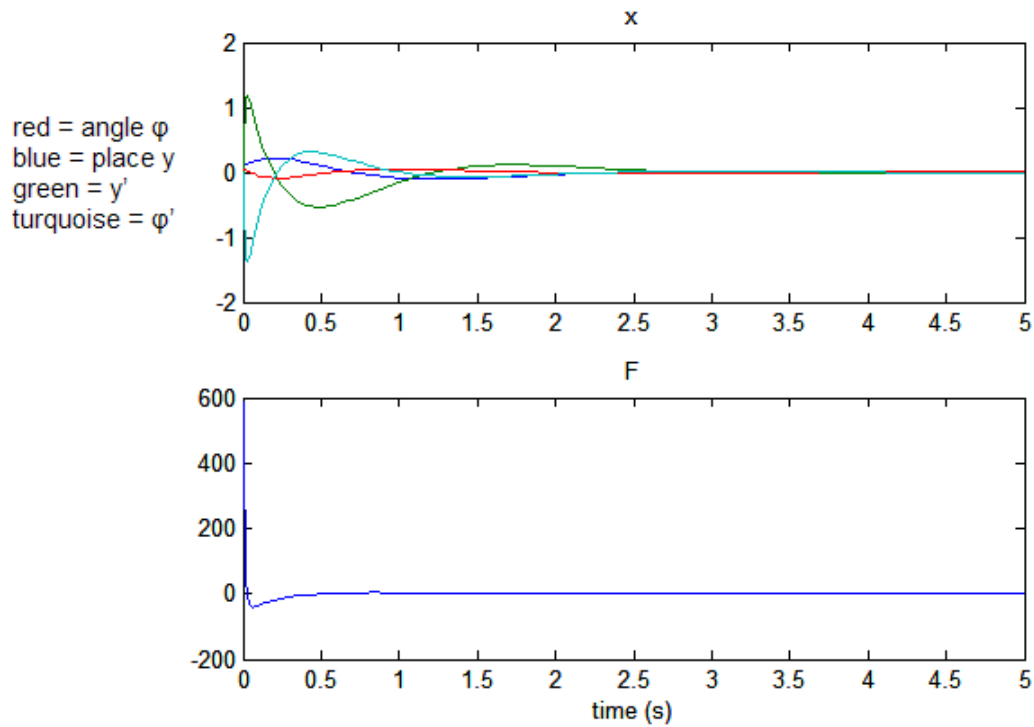
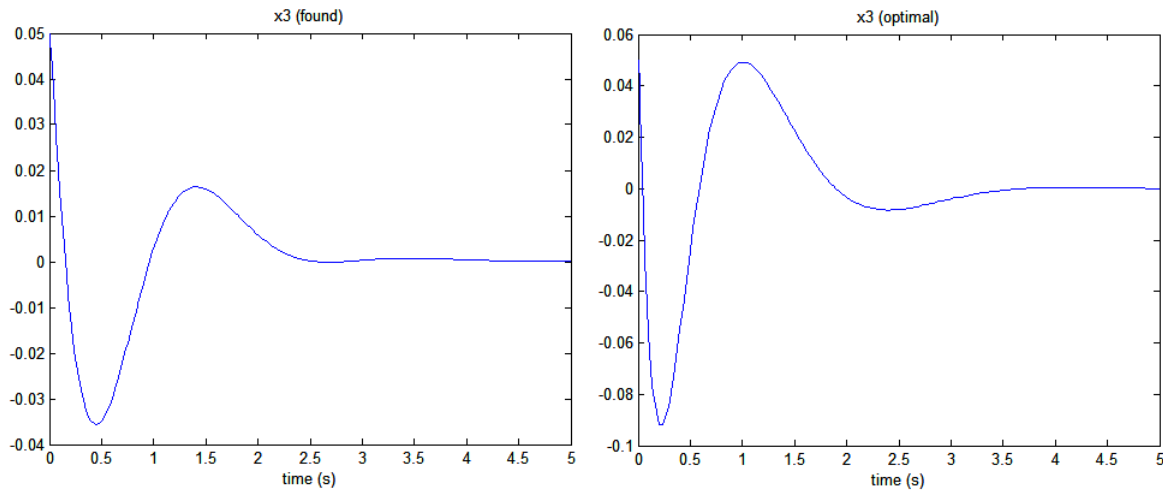


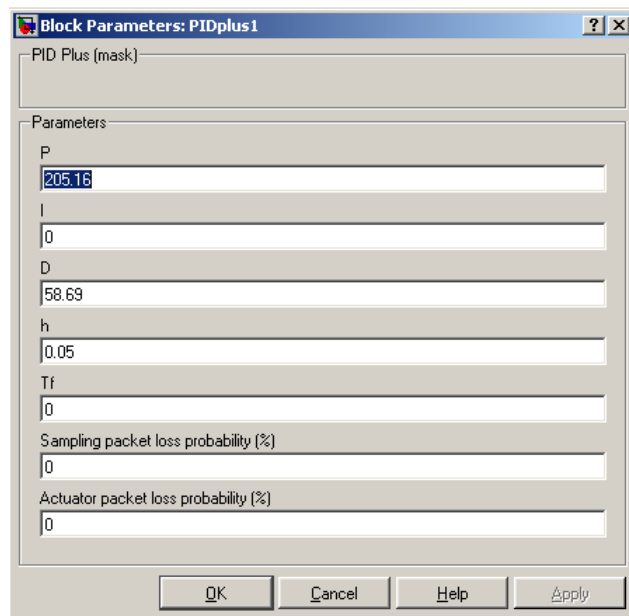
Figure 10. Signals in PID control when parameters from optimal control are used. Parameters are the same as in table 2 except there is no integration.

To compare PID with the parameters found and the one's taken from optimal control a closer look at  $x_3$  is in order. In figure 11  $x_3$  in both cases is drawn. It can be seen that the overshoots are much larger and the changes faster with the optimal parameters. Differences in settling times are not remarkable.



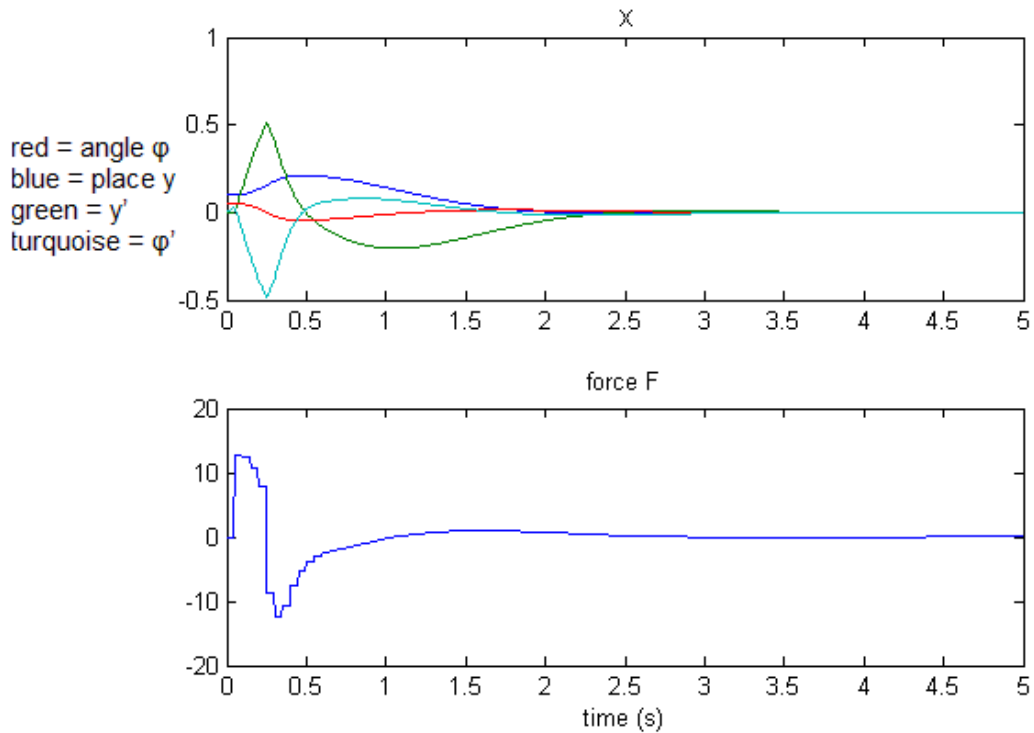
**Figure 11.** Signal  $x_3$  in PID control. On the right results with the parameters from optimal control (table 2) and on the left with the parameters found (table 1).

The interface of PIDplus can be seen in figure 12. The user can choose parameters  $P$ ,  $I$ ,  $D$ ,  $h$  and  $T_f$ .  $T_f$  is the time constant of the filter that is added to the derivative part (equation 41) and  $h$  is the sampling time. The percentage of how many packets get lost on the way can be chosen in Sampling packet loss probability and Actuator packet loss probability. Regarding the last two parameters the controllers have to be identical.



**Figure 12.** PIDplus as a mask.

$T_f$  is kept zero for further notice so the derivative part is the same as in traditional PID. The parameters from table 2 are used in all the simulations. In figure 13 all the signals are presented when there is no packet loss. It can be seen that it takes less than 3 seconds for the signals to reach zero.



**Figure 13.** Signals in PIDplus control when the parameters from table 2 are used and there is no packet loss.

The next figure (figure 14) presents a situation where both packet losses are set to 20%. In figure 15 the value is 40%. There is not much difference between the situations when there is no packet loss and when both packet losses are set to be 20% so PIDplus still handles the packet loss well if one fifth of all packets get lost on the way. With 20% packet loss the changes are not as fast as without packet loss and overshoots are a bit larger. For 40% of packet loss the time span of the figure is 10 seconds instead of 5 as before. It can be seen that the signals have not settled after 8 seconds. It can be predicted that PIDplus does not cope with greater packet losses than 40%. That is why 41% of packet loss is tested and the result is presented in figure 16. In figure 17 a closer look at the signals is taken when both packet losses are set to 41%.

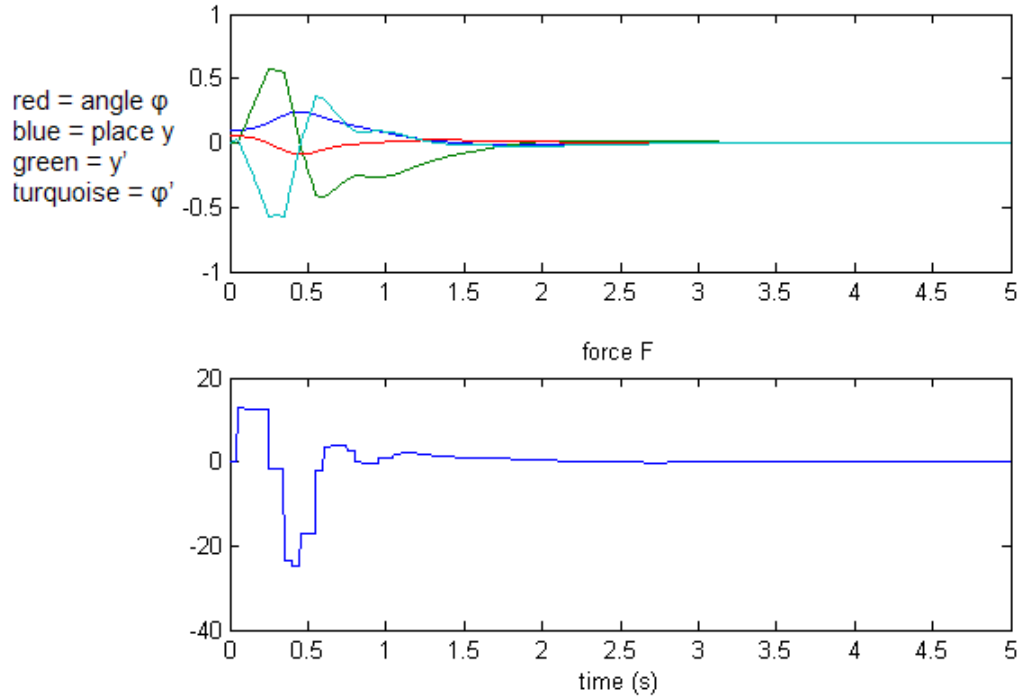


Figure 14. PIDplus when sampling packet loss and actuator packet loss are both set to 20 %.

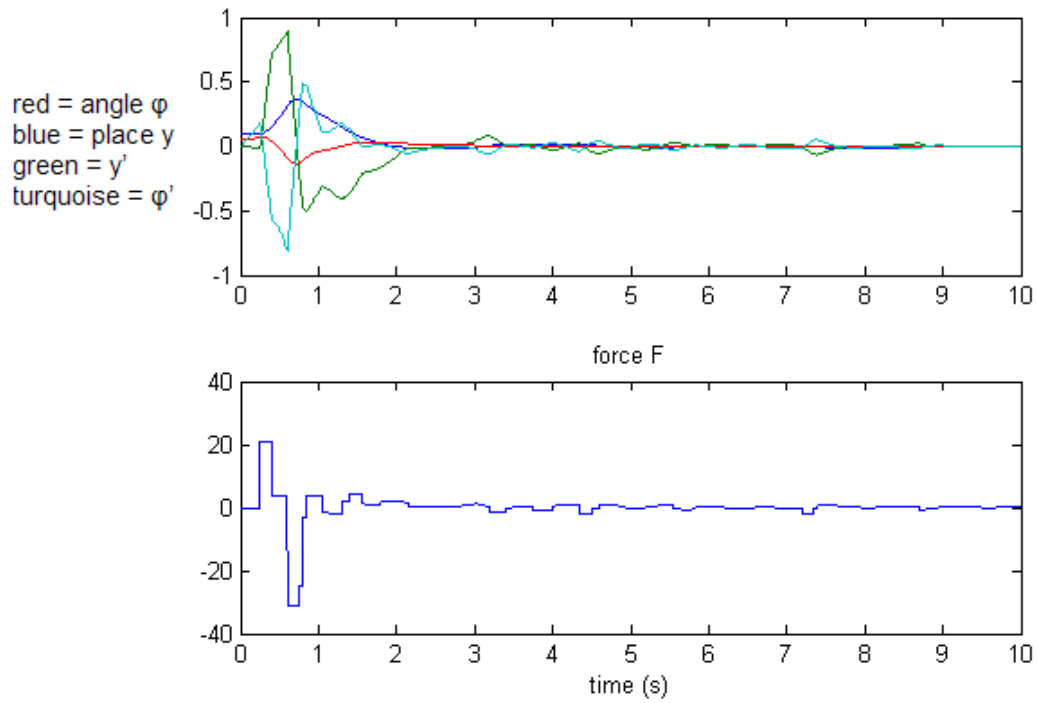
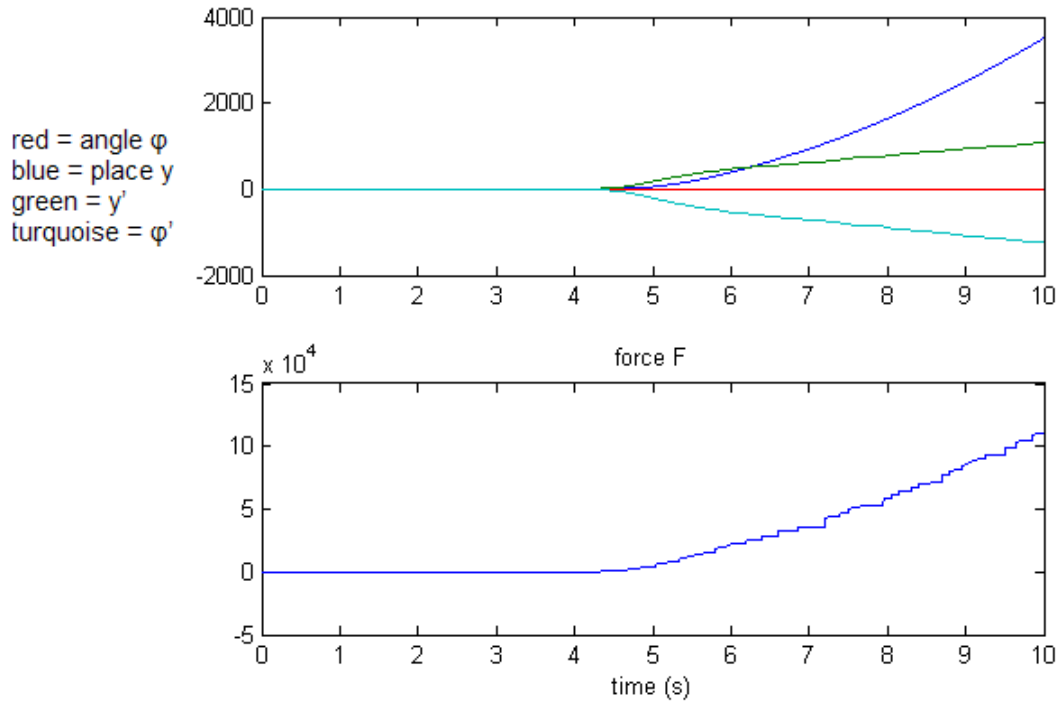
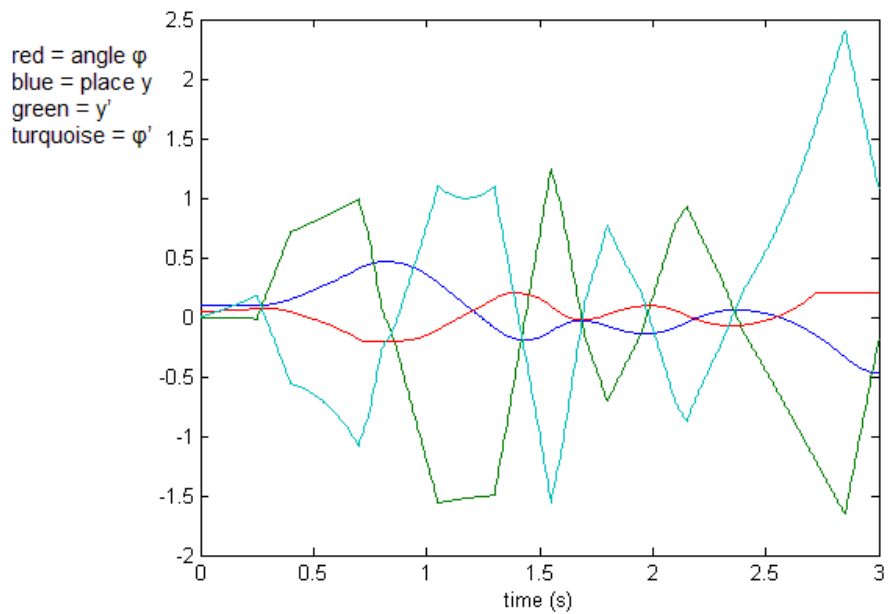


Figure 15. PIDplus when sampling packet loss and actuator packet loss are both set to 40%.



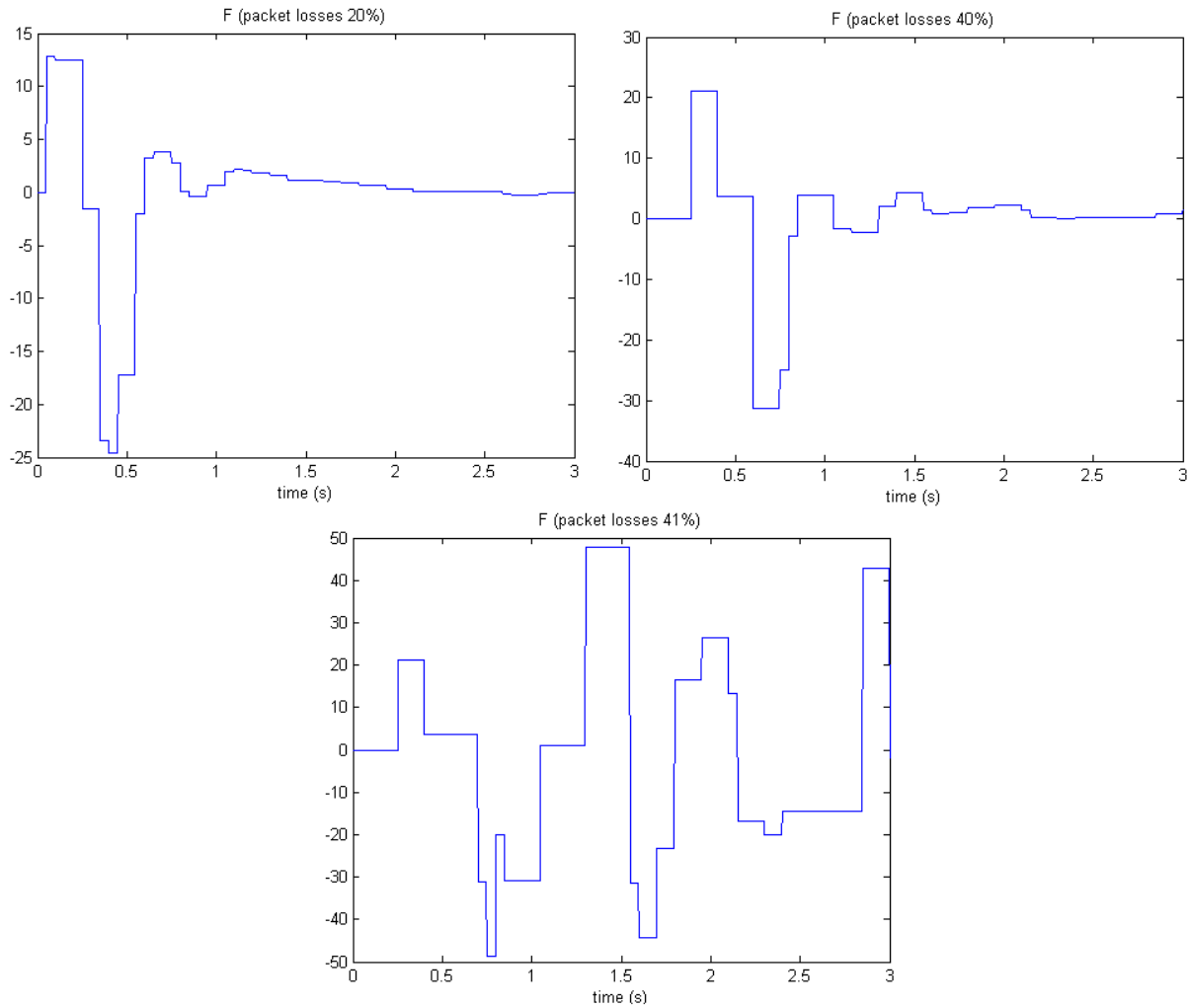
**Figure 16.** PIDplus with packet losses set to 41%.



**Figure 17.** A closer look at what happens to signals when both packet losses are 41%.

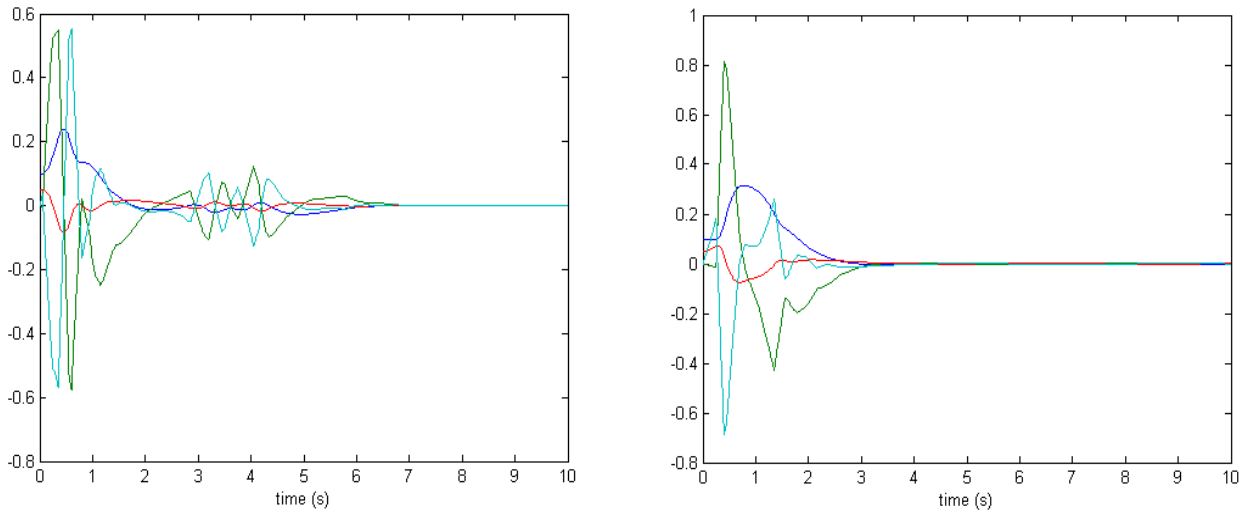
To have more precise idea what happens when the percentage of packet loss is increased a shorter time span of the signals  $F$  are drawn in figure 18. From the figures one can read how many packets get lost on the way. In the first figure the longest time span when  $F$  is not updated is three times the sampling time

(0.05s) meaning that two packets disappear. When packet losses are increased to 40% or 41% the gaps can be as long as nine times the sampling time which means that eight packets disappear.



**Figure 18.** Signal  $F$  when both packet losses are set to be 20%, 40% and 41%.

In figure 19 the signals are drawn when the other packet loss is set to be zero. On the left sampling packet loss is 50 % and actuator packet loss 0 %. On the right the sampling packet loss is 0 % and actuator packet loss 50 %. It can be seen from the figure that when all the sampling packets are received PIDplus works quite well despite the fact that half of the actuator packets are not received. When half of the sampling packets disappear PIDplus does not work as fast or as smoothly.



**Figure 19.** On the left the signals when sampling packet loss is 50 % and actuator packet loss is 0 %. On the right sampling packet loss is 0 % and actuator packet loss is 50 %.

## 4. Modernization of Halvari

The Halvari ball balancing system was renewed in the fall of 2008. The system had problems due to the output power of the existing dc-motor and it wasn't able to provide enough torque to balance the ball in the upright position. The motor was able to shift the cart adequately in the other direction but lacked the torque to move it in the opposite direction. This made it almost impossible to demonstrate the systems functionality to students. The system was tested alternately with a different power supply, motor and motor controller and the problems were derived to locate in the old motor. The conclusion was made that the motor had to be replaced by a new one. At the same time a decision was made to replace the old power supply and motor controller by new ones and bring the system to the 21<sup>st</sup> century. The old analog unit was meant to be kept in place and to be used in measuring the position of the cart and the balls angle deviation from equilibrium. However during test measurements some electrical unit broke and we decided it was safer to continue without the old analog unit.

### 4.1 Renewed parts

The backbone of the whole device is the dc-motor. The motor had to operate on 24 volts and to produce enough output power that was enough to move the heavy cart. Also the motors rotational speed was taken in notice. An adequate motor was found from Dunkermotoren products. The model GR 80x80 has an output power of 240 W, operating on 24 V and 10 A [4]. The motor is able to rotate 3200 revolutions per minute. To make the process even more precise, a built in incremental encoder was also chosen. The motor is displayed in figure 20.



**Figure 20.** Dunkermotoren GR 80x80 on the left, Sunpower-S-240-24 in the middle, and Electromen EM-176 DC-motor controller on the right

Without an appropriate power supply that is able to produce the needed output power, the motor is useless. The new power supply was ordered from Farnell and the model is Sunpower-S-240-24 [5]. The power supply is able to produce 240 W using 24 volts.

A motor controller is a device that forms the control voltage for the motor. It controls the rotation direction by the polarity of the voltage signal and the rotation speed by the signals amplitude. The motor controller was taken from the Rotary Inverted Pendulum (RIP) device due to the lack of time and a new one was ordered for that device. The motor controller was suitable to be used with the new

power supply and motor, because it uses a 12-35 Vdc supply voltage, and produces a 0-29 V and 0-15 A signal for the controlled motor. The motor speed is controlled by using 0-5 V voltage signal, and the rotation direction is set with a voltage signal from 0-1 V to clockwise and with 4-30 V to counterclockwise. The motor controller was originally ordered from Electromen and the model is EM-176 DC-motor controller [6].

After renewing the power supply and motor controller the system was tested using the old motor. The old motor started to work properly and it was decided not to replace the motor at that time. The new motor is installed if the old motor brakes down or is starting to show signals that it is wearing out.

## 4.2 Communication and connections in the system

There are two different possibilities to interact with the process. The cart can be moved manually using a joystick or an optimal controller can be used to determine the control signal to the system. The control mode can be chosen from the systems graphical interface. There is more information about the Simulink-model and the interface of the system in chapter 4.4.

The supply voltage of the joysticks potentiometer is provided by the computers I/O-card and it is 5 volts. When using the joystick control the computer receives a value from the joysticks potentiometer. The computer determines from this measurement the control direction and speed. If the optimal controller is used the control speed and direction is determined using the cart and ball position measurements.

The control speed signal is between 0 and 2.5 volts and the control direction signal is either 0 or 2.5 volts. The I/O-card is able to provide output signals up to 5 volts, but the scale was chosen lower so the system is adaptable with the wireless nodes which are able to only provide an output voltage of 2.5 V. There is more information of the wireless configuration and the nodes in chapter 5. The control speed and direction signals are amplified using a Texas Instruments TLC2272 rail-to-rail operational amplifier. [7] The amplified signals go to the motor controller which creates the signal for the dc-motor. The signal values and communication between different parts of the system are shown in figure 21.

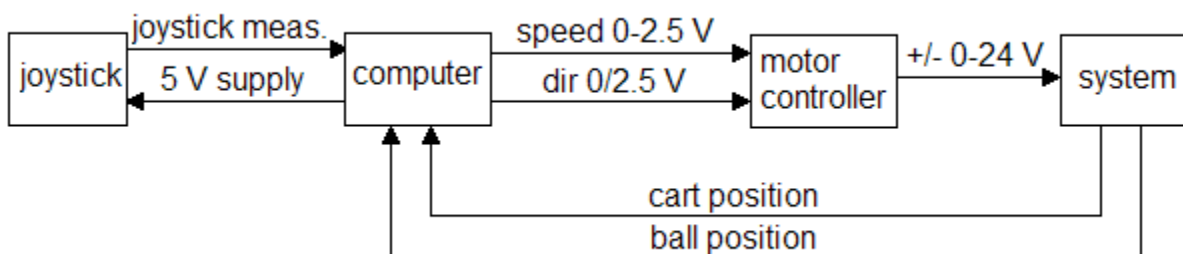
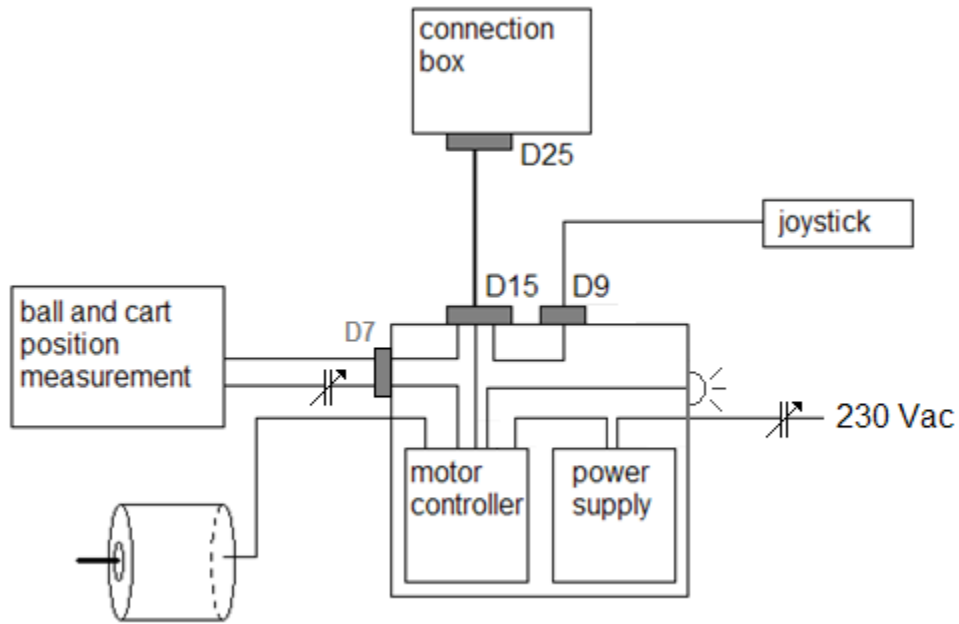


Figure 11. Signal values between different components of the system

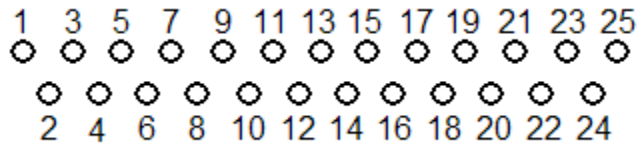
To make the system more modular, it was designed so each part of the system can be replaced easily by just unplugging it from the system. The connections between each component in the system are shown in figure 22. The supply voltage to the system is controlled by a switch; the led-light beside the switch

indicates is the system turned on or off. There is also a switch to control the supply power to the measurements.



**Figure 22.** Connections between different components of the system

In the following, different connection plugs are explained thoroughly. The plugs are described so that they are observed from the female side. In figure 23 the D25 connection is shown and in table 3 the description of each connection is explained.

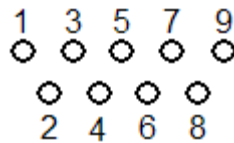


**Figure 23.** Numbering of the D25-connection plug

**Table 3.** Used pins and connection box ports

D25 pin number	Color of wire	Connection box port	Explanation
1	-	Not connected	Joystick on/off
2	-	Not connected	-
3	-	Not connected	Measurement on/off
5	Yellow	57 ACH7	Cart position measurement
9	Green	47 DI03	Motor direction control
10	Black	24 AIGND	Virtual ground
13	Red	65 ACH2	Ball position measurement
16	Purple	Not connected	Incremental encoder
17	Black	14 +5V	Joystick 5V
21	Blue	21 DAC1OUT	Motor speed control
24	Green	68 ACHO	Joystick measurement
25	Blue	56 AIGND	Ground

In figure 24 the D9 connection is shown and in table 4 the description of each connection is explained.

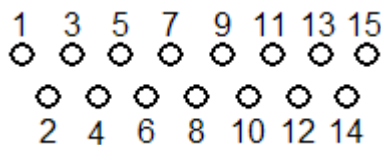


**Figure 24.** Numbering of the D9-joystick plug

**Table 4.** Used pins and connection box ports

D9 pin number	Color of wire	D15 pin number	Explanation
1	Black	2	Joystick on/off
7	Yellow	15	Ground
8	Red	1	Joystick 5V
9	Blue	14	Joystick measurement

In figure 25 the D15 connection is shown and in table 3 the description of each connection is explained.



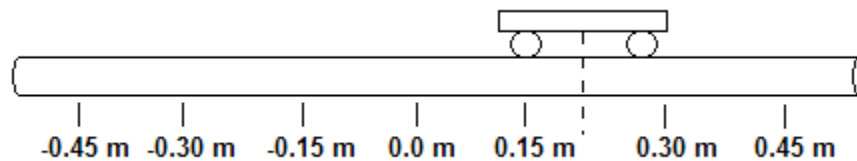
**Figure 25.** Numbering of the D9-joystick plug

**Table 5.** Used pins and connection box ports

D15 pin number	Color of wire	Connected to	Explanation
1	Red	Joystick	Joystick 5V
2	Black	Joystick	Joystick on/off
3	Green	Motor controller	Motor speed control
4	Black	Motor controller	Ground
5	Red	Measurements	Measurements on/off
7	Red	Measurements	Ball Position
8	Blue	Measurements	Ground
11	Yellow	Measurements	Cart Position
12	Blue	Motor controller	Motor direction control
14	Blue	Joystick	Joystick measurement
15	Yellow	Joystick	Ground

### 4.3 Calibrating the system

The LQR-control uses position of the cart in meters and angle of the ball in radians. Because the measurements (position and angle) from the Halvari are in volts they have to be calibrated. The position of the cart is calibrated by measuring the actual position of the cart and comparing it to the measured voltage signal. The center point of the rails is marked as the carts position  $0\text{ m}$ . To the left of the center point the cart can travel  $-0.45\text{ m}$ , and to the right  $0.45\text{ m}$ . In figure 26 the carts position is about  $0.22\text{ m}$  and at that position the measured voltage is about  $805\text{ mV}$ . The carts position and voltage was measured from 20 positions that were distributed evenly thru the whole traveling space. The measured values are shown in appendix D. The actual calibration is done in the Simulink-models block *Cart position*.



**Figure 26.** Determining the position of the cart

The angle of the ball was calibrated with the same method. The equilibrium point was set as the zero position. To the left the ball can travel  $-0.105\text{ m}$  and to the right  $0.105\text{ m}$ . The balls position and voltage was measured from 23 positions that were distributed evenly. In figure 27 the balls position is about  $-0.03\text{ m}$  and at that position the voltage measurement is  $1020\text{ mV}$ . Once the position of the ball is known the angle can be determined. The measured values are shown in appendix D.

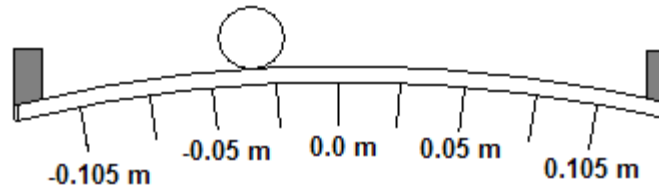


Figure 27. Determining the position of the cart

#### 4.4 Graphical interface and Simulink-model

The Simulink-model of the system is shown in appendix E. The model receives three values as input to the model: the position measurements of the ball and cart and the joysticks potentiometer value. The used control, joystick or optimal controller, is determined using the graphical interfaces radio buttons *joystick* and *control*. The interface is displayed in figure 28. The interface updates the Simulink-models parameter *control\_source*; which is an input for a switch which determines is the control source the joystick or the optimal controller.

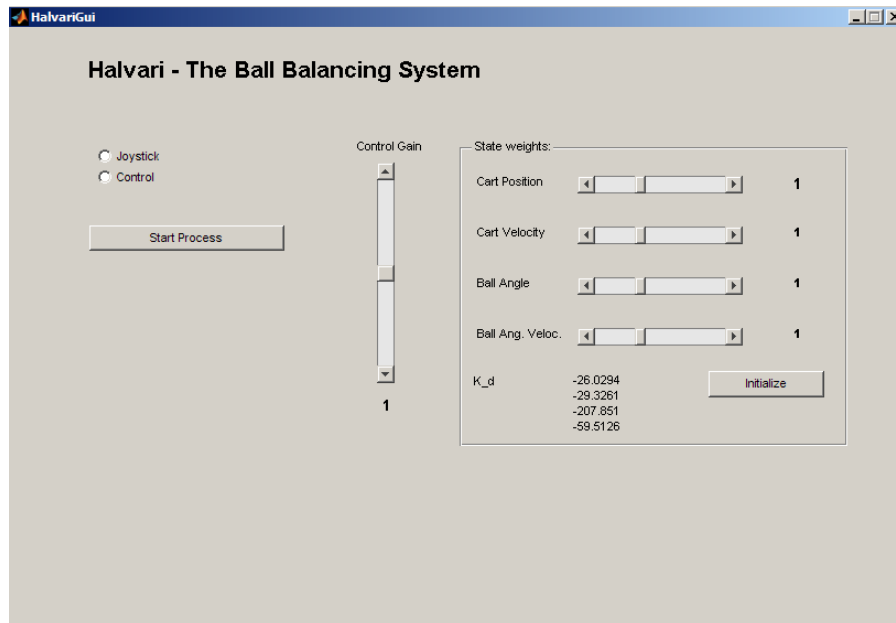
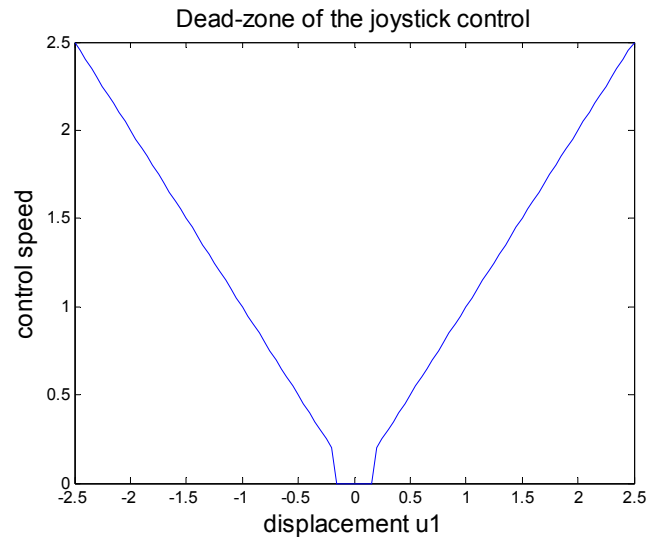


Figure 28. Graphical interface of the Ball Balancing System

Using the joystick, the control value is determined from the displacement of the joystick shaft. The received values are between zero and five volts, and the model scales these values between -2.5 and +2.5 volts by subtracting 2.5 from the measured value; this value is denoted as  $u1$ . To eliminate the users very small shaft displacements and noise while the joystick is in the center position a dead-zone is

introduced in the joystick control. Figure 29 shows the scales of the dead-zone. In the Simulink-model there are two logical operations which compare is the variable  $u1$  between the values -0.15 and 0.15. If the operation is true it returns one and this value is subtracted by one. So the equation returns one if the displacement is out of the dead-zone and zero if the displacement is outside the dead-zone. After this, the scaled parameter  $u1$  is multiplied by the result of the equation and it returns either zero or  $u1$ . From the sign of  $u1$  the control direction is determined using a switch and the control speed is retrieved taking the absolute value of  $u1$ .



**Figure 29.** Dead-zone of the joystick control. The dead-zone is  $\pm 6.0\%$  of the total scale.

The optimal control uses measurements of the cart and ball position to calculate the control. The cart position in meters and the balls angular displacement from the equilibrium are retrieved from the measured cart and ball position voltages. This is done by interpolating the table which is shown in appendix D. Derivate of the cart and ball positions return the speeds of the cart and ball. These values are multiplied by the calculated gain  $K$  introduced in part 2.1. The effect of each term can be set by setting the state weights from the graphical interface, which sets the values of the vector *weights* in the Simulink-model. From the result the control direction and speed are set for the process, using the National Instruments analog output block.

## 5. From wired to wireless

Once going from a wired system to a wireless application, some changes needed to be made in the physical system. First the signal levels had to be lowered to a desired level, since the wireless nodes are able to produce only 2.5 V to DAC. Second some logic needed to be added since the wireless system doesn't have an interface from where the control mode can be selected. Also the measurements needed to be lowered to 3.3 V since that is the maximum value the wireless nodes ADC can withstand. In the following chapters the basic structure of the system is explained and also the communications between each node is described.

### 5.1 Joystick control

The aim was to make the system as flexible as possible, so the wireless joystick was implemented so it can also be used with the computer control. It functions like the wired joystick except the measured values from the joystick's potentiometer are not the same. A new Simulink-model was created where this was taken into account.

The implementation and the basic functions of the nodes are explained in the following. To help picture the communication between nodes see figure 30. Node 1 is the wireless joystick node which reads the value of the joystick potentiometer using the external pins. After reading the value from the ADC the program compares if the joystick is in the center position. If the joystick enters the center area but a *moving buffer* array is not empty, the program sends the last value stored in the array to N2 and sets the array to zero. If the joystick shaft is kept in the middle the program repetitively stores the latest five readings from the ADC and calculates the median value of these. This value is sent to node N2. In the center position the packets are sent every 100 ms to the node N2.

If the joystick isn't in the center position the same procedures are done as in the center position mode but the transmission rate is 25 ms. The first byte of the transmitted packet includes information if the joystick is in the center position or not. The next two bytes include the horizontal displacement of the joystick so that the least significant byte is in the packet first. The following two bytes include the vertical displacement of the joystick. The last two bytes of the packet include information of the execution round. With this information we can calculate how many packets are dropped in the transmission phase. The used radio channel is 12.

The node N2 is used to receive the potentiometer values from the wireless joystick and then it sets its DAC's so that they correspond to the potentiometer values. Node N2 listens to its defined radio channel and it reads its radio socket in time instances defined by *waiting\_time*. The *waiting\_time* is 300 ms if the joystick isn't in the middle position and 500 ms if it is. Port 6.0 is used to determine if the joystick is on or off. If N2 is receiving packets from node N1 the port 6.0 is set up.

If the node doesn't receive packets three times the length of the waiting time it sets its DAC's to 1.25 V which corresponds the middle position of the joystick. Also port 6.0 is set down which corresponds to the fact that the joystick is off. If the node is receiving packets it reconstructs the variables and sets its DAC's to correspond these values.

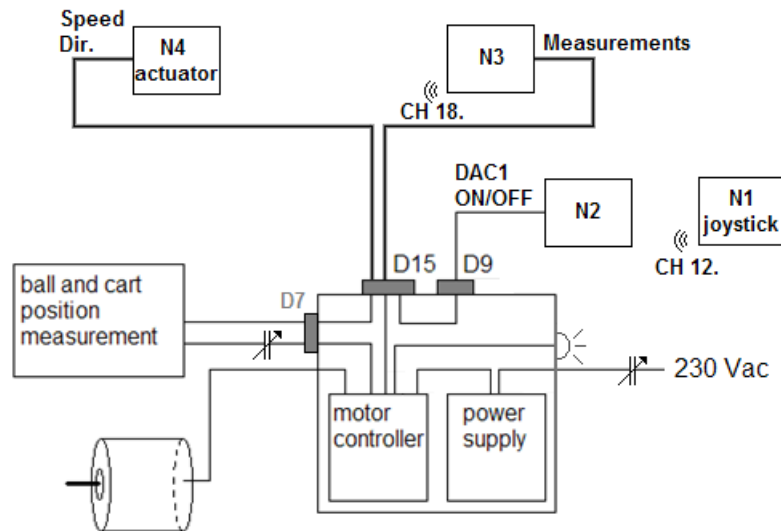


Figure 30. The configuration of the nodes when joystick control is used

The node N3 is used to monitor are we using the joystick, or do we want to use the PIDplus-controller to control the cart. The node reads from its ADC four measurements: joystick on-off information, the value of the joystick potentiometer, position of the cart and position of the ball. In this chapter we are only interested in the joystick on-off information and the value of the joystick's potentiometer. If the joystick is on we determine if the joystick is in the dead zone or not. If it is in the dead zone we set the control speed and direction to zero because we don't want any control then. If the joystick is outside of the dead zone we compare the measurement to a predefined center position value. From this comparison we can define the control direction. The control speed is then scaled so that the maximum joystick displacement responds to the maximum output value which is 4095 (2.5 V). The control speed is smoothed with a second order function so that small displacements outside the dead zone create a fairly low control speed. As the joystick is displaced further away from the center position, the larger the change in control speed becomes. The control speed and direction are sent to node N4 every 20 ms. The used radio channel is channel number 18.

Node N4 is the actuator of the system and it sets its DAC's to correspond to the calculated control. N4 listens to the defined radio channel (channel 18) and it reads its radio socket. The first byte of the received packet includes information if the packet is from the joystick or the PIDplus-controller. If the

packet is from the joystick the program reads the next six bytes of the packet. The first two bytes include the information of the control speed and the third byte is the control direction. The last three bytes in joystick control mode are zeros but they are used if the control comes from the PIDplus-controller. The program reconstructs the control speed variable and sets its external pin 7 to correspond that value. If a received value is greater than 4095, it is set to be 4095 because that is the maximum value what the DAC can output.

## 5.2 PIDPlus control

Using the PIDPlus control mode with the wireless nodes the joystick has to be turned off and the measurement switch turned on. The nodes N1 and N2 have no function in the system while the control is on.

The node N3 is used to monitor are we using the joystick, or do we want to use the PIDplus-controller to control the cart. The node reads from its ADC four measurements: measurement on-off information, the value of the joystick potentiometer, position of the cart and position of the ball. In this chapter we are only interested in the measurement on-off information and the measurements of the ball and cart position. If the measurement switch is on we change the radio channel to channel number 16 because now we are sending the packets to N3.5 and not N4. The measured values are transmission buffer and they are sent every 60 ms. Figure 31 illustrates the communications between the nodes when PIDPlus-control is used.

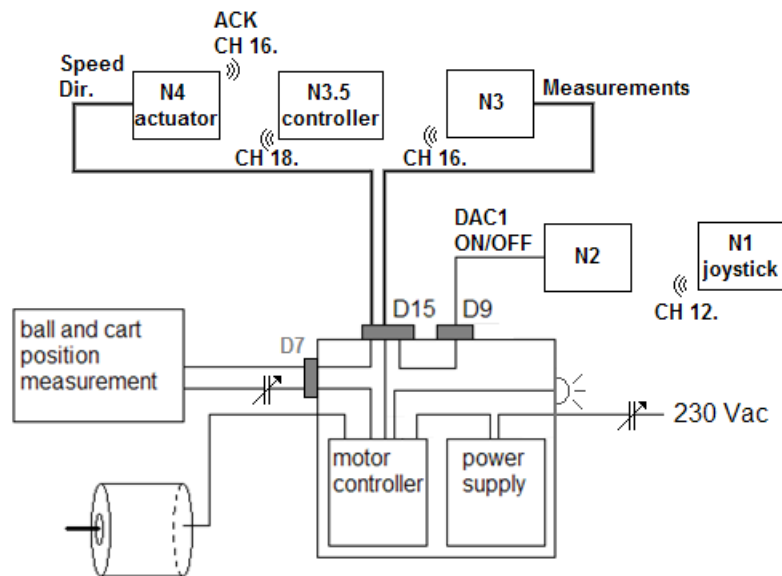


Figure 31. The configuration of the nodes when PIDPlus-control is used

The node N3.5 receives the measurements from the node N3. Since the measurements are in integer format they need to be first computed to voltage values and after this they can be transformed to the actual position of the cart and angle of the balls displacement. The integer value of the measurement can be scaled to volts with the equation (42).

$$Pos\_volt = ((float)(Pos)) * MAX\_ADC / MAX\_SCALE; \quad (42)$$

Variable *Pos* is the reconstructed measurement from N3. *MAX\_ADC* is the maximum value of the ADC and it is 3.3 V and *MAX\_SCALE* refers to the maximum integer scale of the ADC and has a value of 4095. The voltage value of the cart can be transformed to position in meters with the equation (43).

$$cartPosM = 0.5360 - 0.4188 * Pos\_volt + 0.0743 * Pos\_volt^2 - 0.0343 * Pos\_volt^3 \quad (43)$$

The voltage value of the ball can be transformed to the angle displacement in radians with the equation (44).

$$ballPosR = -0.2667 + 0.2906 * Pos\_volt - 0.1152 * Pos\_volt^2 + 0.0264 * Pos\_volt^3 \quad (44)$$

The code for the PIDPlus controller is introduced in appendix F. After the control values have been calculated the sign of the control variables is stored and an absolute value of the variables is taken. Before transmission the variables need to be casted from float to integers. The node sends the flag of the packet, absolute values of the control variables and two variables which contain the sign of the control variables. After transmission the node waits for 40 ms for an acknowledgement packet from N4. If it is received the parameters *O\_b* and *O\_c* is updated. These two variables are used when calculating the filter term for the control.

Node N4 is the actuator of the system and it sets its DAC's to correspond to the calculated control. N4 listens to the defined radio channel (channel 18) and it reads its radio socket. The first byte of the received packet includes information is the packet from the joystick or the PIDplus-controller. If the packet is from the PIDplus-controller the program reads the next six bytes of the packet. The first two bytes include the information of the control speed associated to the ball control and the third byte is the sign of the control direction. The next two bytes include the information of the control speed associated to the cart control and the third byte is the sign of the control direction. The program reconstructs the control speed variables and sets its external pin 7 to correspond that value. If a control speed value is greater than 4095, it is set to be 4095 because that is the maximum value what the DAC can output. The originally received variables are sent back to N3.5 with the acknowledgement packet because the filter term uses the last successfully sent control variables to calculate the filter term.

## 6. The exercise for the 2<sup>nd</sup> tutorial and the solution

The aim of the project was to develop a tutorial exercise for the course Wireless Automation AS-74.3199. In the first tutorial students had to implement the code to N1. In this tutorial the main objectives of learning were how to read from the nodes ADC's and how to send the data forward to another node. It was natural that the second tutorial would concentrate on receiving data packets from another node and assign values for the nodes DAC's.

The second tutorials task is to implement a node that receives data from N3 and assigns the correct DAC's. To simplify the exercise only the joystick control mode is considered. The node N4 receives data packets from N3 which contain a control flag, control speed and control direction. The first byte of the data packet contains the flag of the packet. This flag contains information is the control coming from the joystick or the controller. The next two bytes contain the control speed, so that the first byte is the least significant byte and the second is the most significant byte of the control speed variable. The last byte contains the control direction information. The student has to read these variables in correct order and reconstruct the control speed variable by shifting the most significant byte to the left by 8 spaces and adding the two bytes together. The information is stored in an unsigned integer of size 16 bits. After the variables are read a threshold has to be added for the control speed variable since the DAC's are capable of only outputting values up to 4095. After this the external pins can be assigned with the correct values. If the flag equals the PIDPlus control mode an acknowledgement packet has to be sent back to node N3. A functional code is presented in appendix G and slides for the tutorial can be found from the courses home pages. [8]

## References

- [1] Instructions for the laboratory exercise 10 of the course AS-0.2230 Automaatio- ja systeemitekniikan laboratoriotyöt
- [2] Halvari: tila- ja optimisäädön laboratoriotyön uudistaminen, Sami Kiviluoto, 27.8.2004
- [3] Addressing Control Applications Using Wireless Devices, Emerson Global Users Exchange, a powerpoint show, Terry Blevins and Mark Nixon
- [4] Dunkermotoren, Permanent Magnet DC-Motor GR 80x80  
[http://www.dunkermotoren.de/data/technical\\_data/motors/pdf/GR%2080x80.pdf#page=1](http://www.dunkermotoren.de/data/technical_data/motors/pdf/GR%2080x80.pdf#page=1)
- [5] Sunpower, S-240 Series-240W Single Output Switching Power Supply  
<http://www.farnell.com/datasheets/39069.pdf>
- [6] Electromen, EM-176 DC-moottorin nopeussäädin, 12/24Vdc 10A  
<http://www.electromen.com/>
- [7] Texas Instruments, TLC2272 Advanced LinCMOS rail-to-rail operational amplifier  
<http://www.datasheetcatalog.org/datasheet/texasinstruments/tlc2272-q1.pdf>
- [8] Wireless Automation, Weekly Exercises, Tut2  
[https://noppa.tkk.fi/noppa/kurssi/as-74.3199/viikkoharjoitukset/slides\\_3.pdf](https://noppa.tkk.fi/noppa/kurssi/as-74.3199/viikkoharjoitukset/slides_3.pdf)

## Appendix A

```
m-file parametrit.m

% Constants describing Halvari
M = 4;      % mass of the cart without the ball(kg)
m = 0.7;    % mass of the ball (kg)
J = 0.000175; % ball's moment of inertia (kgm2)
r1 = 0.025; % radius of the ball (m)
r = 0.02;   % ball's radius of rotation (m)
R = 0.5;    % radius of the arch (m)
g = 9.81;   % acceleration of gravity (m/s2)

% Coefficients used in the Simulink-model
k1 = -m*(R+r)/(M+m);
k2 = m*(R+r)/(M+m);
k3 = 1/(M+m);
k4 = (m*r^2)/J;
k5 = -(m*r^2)/J;
k6 = (g*m*r^2)/(J*(R+r));
k7 = (M*r^2)/(J*(R+r));
k8 = -(r^2)/(J*(R+r));

% Coefficients of the linearized equations
a = -(g*m^2*r^2)/((M+m)*J+M*m*r^2);
b = (J+m*r^2)/((M+m)*J+M*m*r^2);
c = (g*(M+m)*m*r^2)/((R+r)*((M+m)*J+M*m*r^2));
d = -(m*r^2)/((R+r)*((M+m)*J+M*m*r^2));

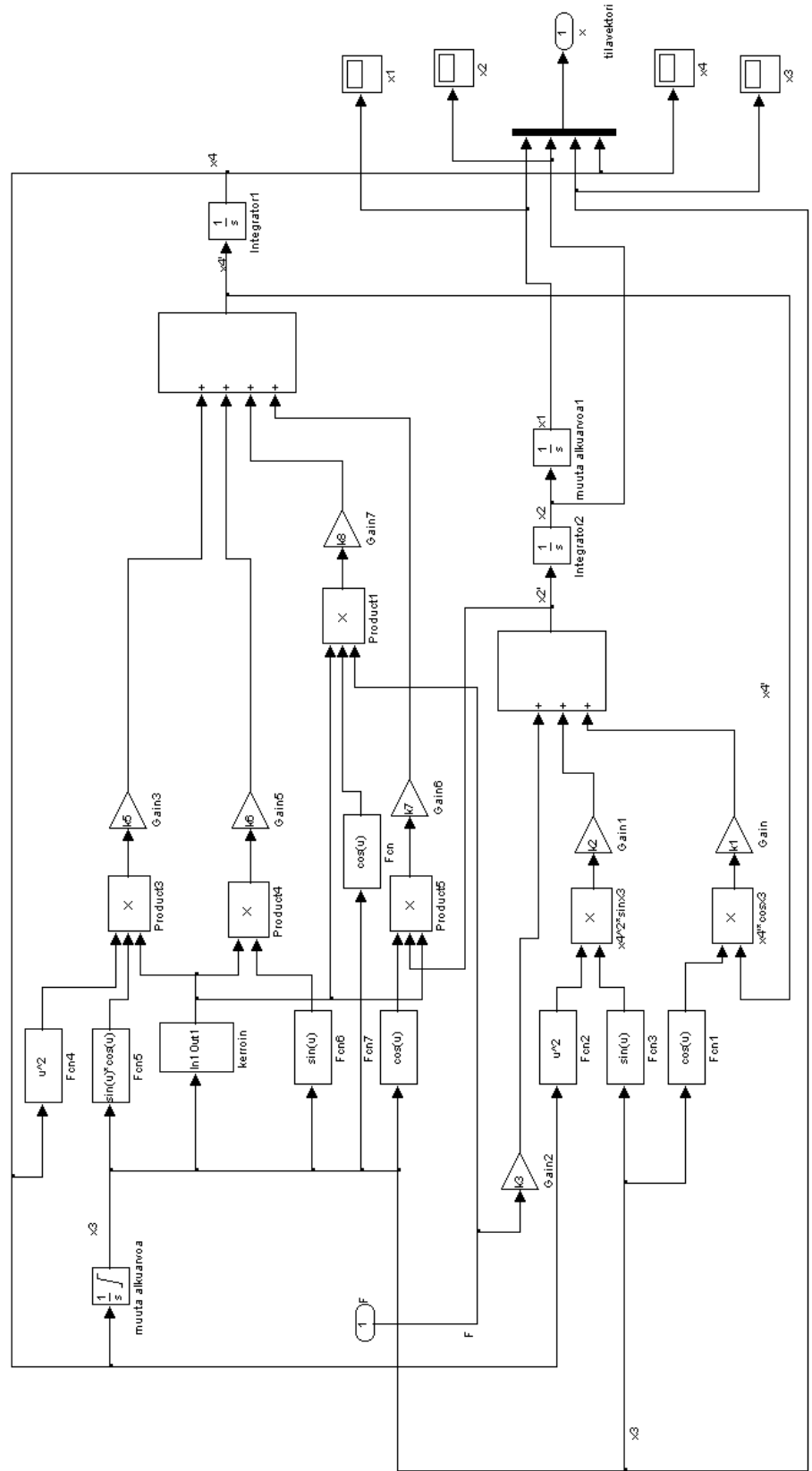
% Continuous state-space model
A = [0 1 0 0;0 0 a 0;0 0 0 1;0 0 c 0];
B = [0;b;0;d];
C = [1 0 0 0;0 0 1 0;0 0 0 0;0 0 0 0];
D = [0;0;0;0];

% Discretisation of the state-space model
sample_t = 0.05; % sampling time
sys = ss(A,B,C,D);
sysd = c2d(sys,sample_t);
```

```
[Ad,Bd,Cd,Dd] = ssdata(sysd);  
% Calculation of K for optimal control  
Qs = [10 0 0 0;0 0.1 0 0;0 0 10 0;0 0 0 0.1];  
Rs = 0.01;  
[K,s,e] = dlqr(Ad,Bd,Qs,Rs);
```

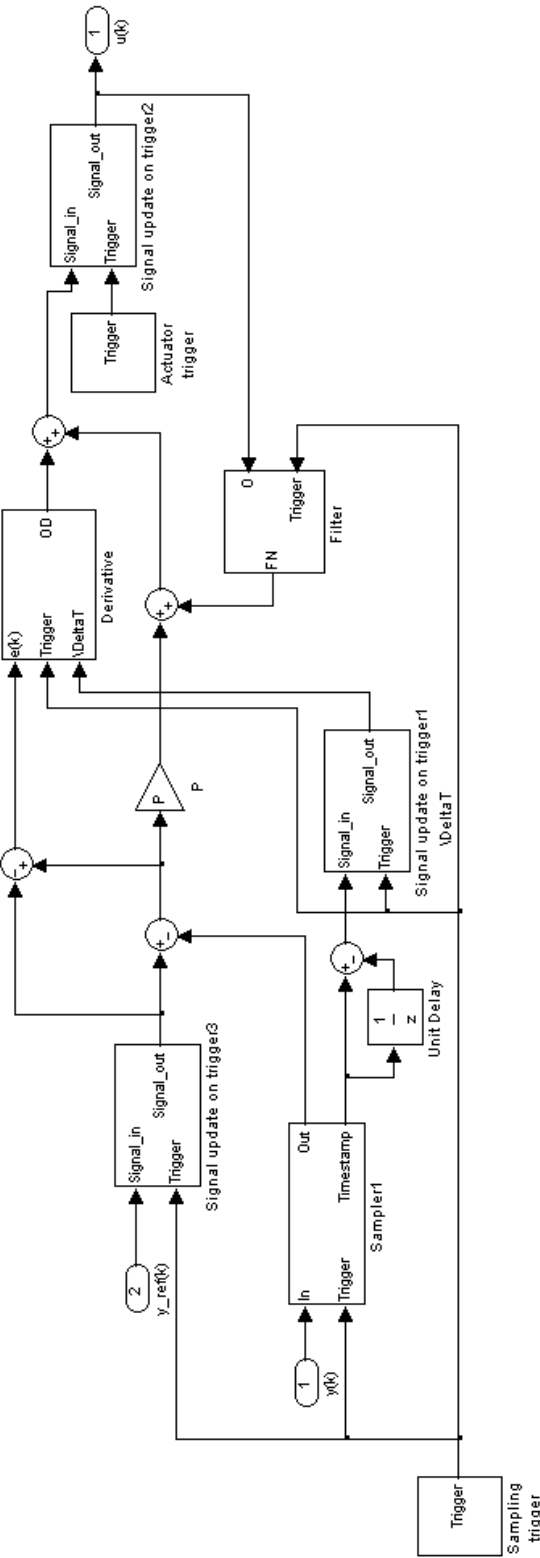
# Appendix B

Simulink model of Halvari



# Appendix C

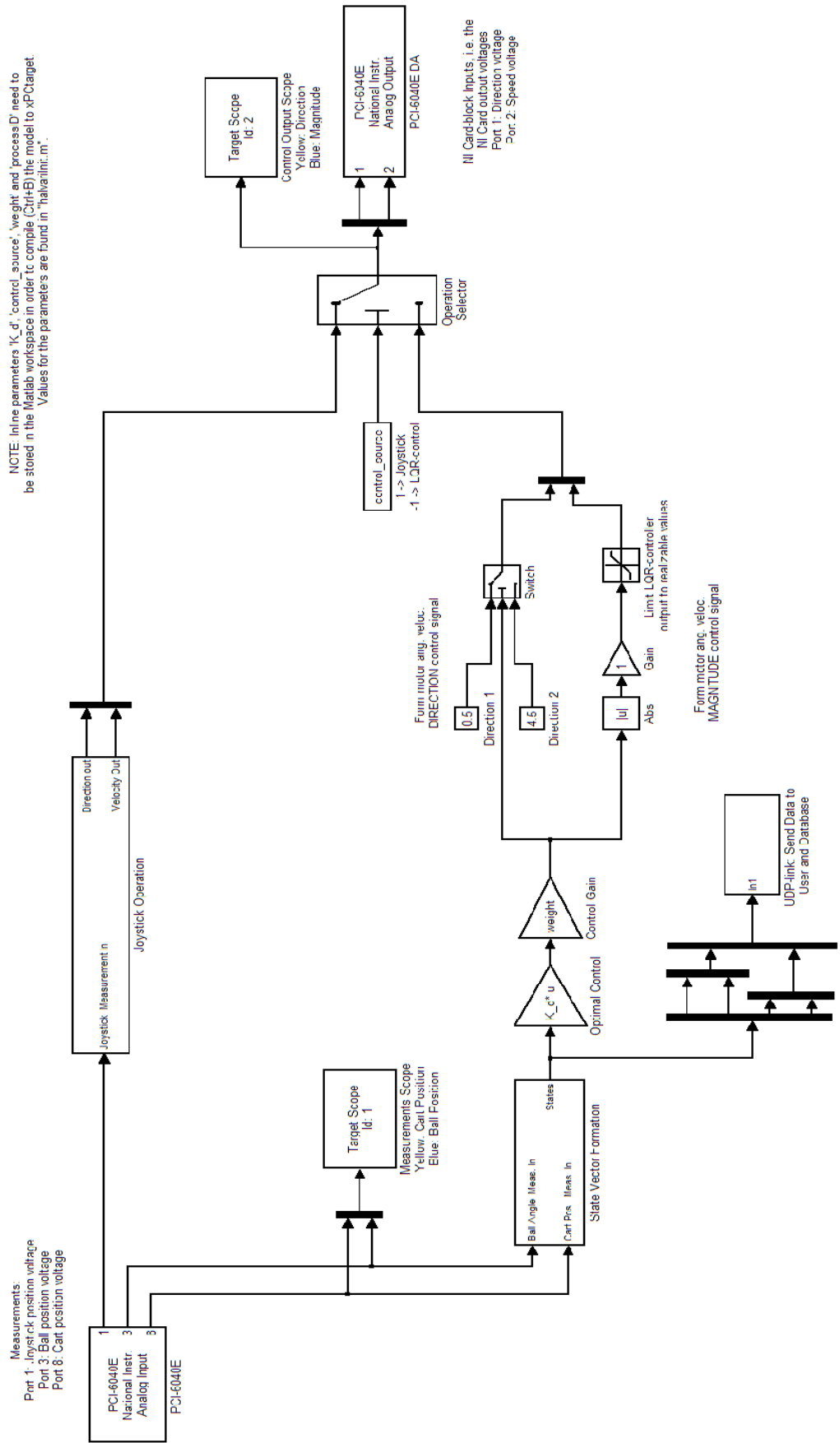
Simulink model of PIDplus



## Appendix D

Cart position [m]	Measured voltage [mV]	Ball position [m]	Measured voltage [mV]
0.475	154	-0.105	191
0.425	265	-0.1	271
0.375	401	-0.09	325
0.325	539	-0.08	430
0.275	677	-0.07	530
0.225	805	-0.06	620
0.175	948	-0.05	750
0.125	1090	-0.04	870
0.075	1218	-0.03	1020
0.025	1351	-0.02	1200
-0.025	1477	-0.01	1350
-0.075	1589	0	1514
-0.125	1703	0.01	1670
-0.175	1812	0.02	1830
-0.225	1922	0.03	1985
-0.275	1998	0.04	2147
-0.325	2097	0.05	2230
-0.375	2194	0.06	2370
-0.425	2293	0.07	2460
-0.475	2359	0.08	2561
		0.09	2650
		0.10	2681
		0.105	2730

# Appendix E



## Appendix F

```
if(num_samples>254)           //num_samples is used to determine the variable
                               //deltaT, which is the time interval between received
                               //packets.
{
    num_samples=2;
}
if(num_samples==1)
{
    deltaT=0.06;
}
else
{
    dT=newT-prevT;
    deltaT=(float)(dT/1000.0);
}

//PID1 updated ONLY when a new packet has arrived. PID1 is used to control
//the ball
//Error of the ball measurement
e_b = ball_ref-ballPosR;

//Filter term of the PIDPlus. If integral term is zero filter isn't used.
if(Ki_b == 0.0)
{
    F_b = 0.0;
}
else
{
    F_b = F_b_prev+(O_b-F_b_prev)*(1-exp(-deltaT*Ki_b/Kp_b));
}

//Derivative. Calculated if more then two packets have been received
if(num_samples > 1)
{
    ef_b=Tf_b/(deltaT+Tf_b)*ef_b_prev+deltaT/(deltaT+Tf_b)*e_b;
}
else
{
    ef_b=e_b;
    ef_b_prev=e_b;
}
D_b=Kd_b*(ef_b-ef_b_prev)/deltaT;

//Controller Output
u_b=(Kp_b*e_b+F_b+D_b)*scale;

//Update states
ef_b_prev=ef_b;
F_b_prev=F_b;
```

```

//PID2 updated ONLY when a new packet has arrived. PID2 is used to control
//the cart
//Error
e_c = cart_ref-cartPosM;

//Filter term of the PIDPlus. If integral term is zero filter isn't used.
if(Ki_c == 0.0)
{
    F_c = 0.0;
}
else
{
    F_c = F_c_prev+(O_c-F_c_prev)*(1-exp(-deltaT*Ki_c/Kp_c));
}

//Derivative. Calculated if more then two packets have been received.
if(num_samples > 1)
{
    ef_c=Tf_c/(deltaT+Tf_c)*ef_c_prev+deltaT/(deltaT+Tf_c)*e_c;
}
else
{
    ef_c=e_c;
    ef_c_prev=e_c;
}

D_c=Kd_c*(ef_c-ef_c_prev)/(deltaT);

//Controller Output
u_c=(Kp_c*e_c+F_c+D_c)*scale;

//Update states
ef_c_prev=ef_c;
F_c_prev=F_c;

```

## Appendix G

```
/* **** file main.c **** */

/* Standard includes. */
#include <stdlib.h>
#include <signal.h>
#include <string.h>

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "bus.h"
#include "gpio.h"
#include "debug.h"
#include "socket.h"
#include "rf.h"
#include "adc.h"
#include "buffer.h"

/*TASKS declaration*/
static void Task_Node4 (void *pvParameters);

/* Variables declaration */
uint8_t flag;
uint8_t control_dir;
uint16_t control_speed;
uint8_t control_speedLS;
uint8_t control_speedMS;

/*Constants declaration*/
uint8_t ACK_PACKET=10;
uint8_t channel1=12;
uint8_t channel3=16;
uint8_t JOYSTICK=1;
uint8_t CONTROL=22;

/* Sockets declaration */
socket_t *Radio_Socket = 0;

/* Buffers declaration */
buffer_t *R_Buffer; //Buffer to receive packets
buffer_t *T_Buffer; //Buffer to send packets

/* Ports definition */
#define PORT_NUM 10

/* Addresses declaration */
sockaddr_t Broadcast_Add =
{
```

```

        ADDR_802_15_4_PAN_LONG,
        { 0xFF, 0xFF, 0xFF, 0xFF,
          0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF },
        PORT_NUM
};

/* main */
int main( void )
{
    /* Initializes the leds */
    LED_INIT();
    if (bus_init() == pdFALSE)
    {
    }
    /* Initializes the debug window */
    debug_init(115200);
    stack_init();
    /* Creates the tasks */
    xTaskCreate(Task_Node4, "NODE4", 256, NULL, (tskIDLE_PRIORITY +
1), NULL);

    P6SEL &= 0x03; //0000 0011
    P6DIR |= 0xCF; //1100 1111

    /* Starts the scheduler */
    vTaskStartScheduler();
    return 0;
}

/* Task_Node1 */
static void Task_Node4 (void *pvParameters)
{
    LED1_ON();
    vTaskDelay(1000/portTICK_RATE_MS); /*Wait 1 sec before starting
activity*/
    LED1_OFF();

    //DAC configuration
    DAC12_0CTL |= 0xE0; //DAC12_0 amplifier setting
    DAC12_1CTL |= 0xE0; //DAC12_1 amplifier setting
    DAC12_0CTL |= 0x200; //DAC12_0 offset voltage calibration
    DAC12_1CTL |= 0x200; //DAC12_1 offset voltage calibration
    while ((DAC12_0CTL != 0x0E0) || (DAC12_1CTL != 0x0E0)) //Wait for the
end of the calibration
    DAC12_0CTL |= 0x100; //DAC12_0 input range (1x);
    DAC12_1CTL |= 0x100; //DAC12_1 input range (1x);

    DAC12_0CTL &= 0xFE; //DAC12_0 and DAC_1 are grouped; 1111 1110
    DAC12_1CTL &= 0x3FF; //The DAC registers are decoupled; 0011 1111 1111

    DAC12_0CTL |= 0x400; //Both the DAC12 latch bits must be bigger than 0;
    DAC12_0CTL |= 0x02; // DAC12_0 conversion enabled;
    DAC12_1CTL |= 0x02; // DAC12_1 conversion enabled;
    ADC12CTL0 |= 0x60; //ADC internal reference = 2.5V

```

```

Radio_Socket = socket(MODULE_CUDP,0); /* Socket creation */
if (socket_bind(Radio_Socket,&Broadcast_Add) == pdFAIL)
{
    debug("Error binding socket\r\n");
}

for ( ;; )
{
    flag=0;
    R_Buffer = socket_read(Radio_Socket,300);
    if (R_Buffer==0)
    {
        DAC12_0DAT = 0; //Speed, External pin 7
        P6OUT &= 0xFE; /*Port 6.0 of the MCU (= external pin 3)
DOWN*/
    }
    else
    {
        flag=buffer_pull_uint8(R_Buffer);
        control_speedLS=buffer_pull_uint8(R_Buffer);
        control_speedMS=buffer_pull_uint8(R_Buffer);
        control_dir=buffer_pull_uint8(R_Buffer);
        control_speed=((control_speedMS<<8)+control_speedLS);
        socket_buffer_free(R_Buffer); /*Free the buffer*/
        R_Buffer=0;

        if (control_speed >= 4095)
        {
            control_speed=4095;
        }

        DAC12_0DAT = control_speed; //Speed, External pin 7

        if (control_dir==1)
        {
            P6OUT |= 0x01; /*Port 6.0 MCU (= external pin 3)UP*/
        }
        else
        {
            P6OUT &= 0xFE; /*Port 6.0 MCU (= external pin 3)DOWN*/
        }

        if (flag==CONTROL) //If in CONTROL mode, N4 transmits back
an ACK packet to N3.5
        {
            T_Buffer=socket_buffer_get(Radio_Socket);
            if (T_Buffer)
            {
                buffer_push_uint8(T_Buffer, ACK_PACKET);
                buffer_push_uint8(T_Buffer, control_speedLS);
                buffer_push_uint8(T_Buffer, control_speedMS);
                buffer_push_uint8(T_Buffer, control_dir);

                if(socket_sendto(Radio_Socket,
&Broadcast_Add,T_Buffer) == pdTRUE)

```

```
    {
        debug_printf("ACK PACKET: %d %d\r\n",
                    control_dir,control_speed);
    }
    else
    {
        debug("ERROR: ACK Packet not
            transmitted!\r\n");

        socket_buffer_free(T_Buffer); /* Free the
            buffer */
        T_Buffer=0;
    }
}
}
}
}
```