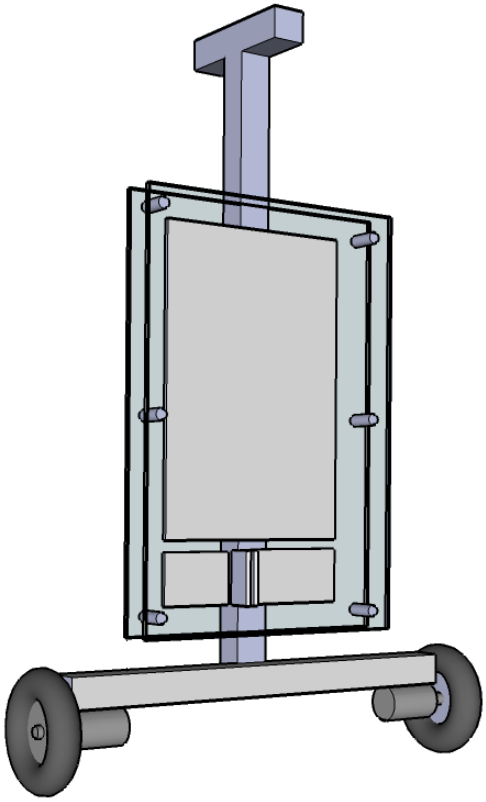
CHALMERS



Mobile Inverted Pendulum

Control Of An Unstable Process Using Open Source Real-Time Operating System

Master of Science Thesis by

BJÖRN CARLSSON
PER ÖRBÄCK

Department of Signals and Systems
Division of Automatic Control
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2009
Report No. EX055/2009

REPORT NO. EX055/2009

Mobile inverted pendulum

Control of an unstable process using open source real-time operating system

BJÖRN CARLSSON
PER ÖRBÄCK

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2009

Mobile inverted pendulum
Control of an unstable process using open source real-time operating system

BJÖRN CARLSSON
PER ÖRBÄCK

Technical report no EX055/2009
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-772 10 00

Cover:
Sketch of the mobile inverted pendulum.

Chalmers Reproservice
Göteborg, Sweden 2009

Abstract

The inverted pendulum is a classical problem in control theory. One fundamental case is when the pendulum is mounted on a cart which moves back and forth to make the pendulum balancing. In this master thesis an extension of this fundamental case is investigated. The goal of this master thesis work was to construct and build an inverted pendulum on wheels. The finished pendulum should be controlled with help of an open source real-time operating system and have a wireless connection. The total cost of the used parts had to be below 10000 SEK. The main focus was put into the control system. The mechanical and electrical parts were chosen for easy functionality and robustness. These goals were fulfilled.

First a mathematical model of the system was developed representing the system in state space form. The model was then implemented in Simulink. Two kinds of controllers were evaluated, PID and LQR controller, both together with Kalman filtering of the sensor outputs. After the evaluation in Simulink the controllers were implemented on the real system.

The controller algorithm runs on a development board from Analog Devices featuring the Blackfin processor. An open source Linux distribution called uClinux is used together with Xenomai to achieve the real-time property. The Blackfin board use peripheral devices for motor control, angle sensing and Bluetooth connection. Everything is power supplied from an 11,1V battery.

The final choice of controller was the LQR controller, not because the balancing ability but the advantages when it comes to movement control. The pendulum is able to balancing by itself and is controlled wirelessly via Bluetooth from a mobile phone or a laptop PC. The Linux operating system has been easier than expected to work with. However some real-time problems have occurred when using drivers for peripheral devices which not are written for real-time purpose.

Acknowledgements

We would like to thank the people at ÅF Infrastruktur especially our supervisor Kenneth Östberg for his help and interest in our work, Sten Byfeldt for his help with OrCad and all his other tips and tricks, Magnus Janemyr who gave us the chance in the first place to do our master thesis at ÅF. Last but not least we would like to thank Torsten Wik, our examiner at Chalmers, for his support and advice.

Nomenclature

$g - Acceleration\ of\ gravity$

$\theta - Angle\ between\ the\ pendulum\ and\ a\ vertical\ line$

$\phi - Angle\ between\ the\ wheel\ and\ a\ vertical\ line$

$T_m - Torque\ preduced\ between\ the\ pendulum\ and\ the\ wheel$

$m_p - Mass\ of\ the\ pendulum$

$r_p - Distance\ from\ the\ center\ of\ mass\ to\ the\ bottom\ the\ pendulum$

$I_p - Inertia\ of\ the\ pendulum$

$m_w - Mass\ of\ the\ wheel$

$r_w - Radius\ \ of\ the\ wheel$

$I_w - Inertia\ of\ the\ wheel$

$F_p - Vertical\ force$

$F_a - Horizontial\ force$

$F_f - Tangential\ force$

$F_{nw} - Force\ nw\ is\ the\ normal\ force\ which\ act\ on\ the\ wheel$

$s_x - Measurment\ value\ from\ the\ accelerometer\ x\ axis$

$s_y - Measurment\ value\ from\ the\ accelerometer\ y\ axis$

$r_s - Distance\ from\ the\ pendulums\ rotational\ center\ to\ center\ of\ the\ accelerometer$

$r_m - Resistance\ of\ the\ motor\ windings$

$k_v - Motor\ backemf\ constant$

$k_t - Motor\ torque\ constant$

Table of contents

Appendix D

1 Introduction

Nowadays computers play an important roll when it comes to construct controllers. By the usage of computers the controllers can be more advanced and more flexible then basic controllers. These controllers are often implemented in an embedded system that is using some sort of RTOS (Real-Time Operating System). There are several different types of RTOSs, commercial and open source ones [1]. Over time the price of computers decrease and the computational power increase. This has cleared the way for introducing computers and controllers in new products to reasonable prices.

A classical control theory problem is the inverted pendulum. The most fundamental case is when a pendulum is mounted on a cart which can move back and forth in one linear direction. The pendulum is then balanced in upright position by controlling the movement of the cart. The inverted pendulum principle can be varied in many ways to make the system more complex or more interesting. In 2001 the company Segway introduced an interesting construction of a transportation vehicle which was based on the physical dynamics from the inverted pendulum. On the vehicle the cart is replaced by two wheels. Between the wheels a platform is placed with room for one passenger. The Segway is able to balancing the passenger in upright position and at the same time drive forward or backwards [2].

1.1 Purpose and goal

The purpose of this Master of Science thesis work presented here was to construct a mechanical system which dynamics resembles an inverted pendulum. The system is to be controlled by a controller implemented in an open source real-time operating system. The mechanical system is modeled in a computer and different controllers are investigated. This thesis will describe the practical work and the ingoing parts of the system which includes mechanics, electronics and software.

Specific goals were set for this project:

- The pendulum should be able to balancing all by itself without any attachments.
- The vehicle should be able to go forwards, backwards and turn.
- The vehicle should have a wireless connection.
- An open source real-time operating system should be used.
- A controller software should be developed making it possible to control the movement of the pendulum from another device, such as a computer or a mobile phone.
- The cost of the final system should not exceed 10000 SEK.

1.2 Delimitation

The mechanical or electronically systems are not evaluated for optimality. These systems are constructed simple and robust to give good functionality. This does not mean that no effort has been put in to constructing good mechanical or electronically solutions, only that the main focus has not been put in to optimality.

The embedded system chosen for implementation of the controller is a development system delivered with an open source operating system having real-time support. This implies that little effort is required to getting the embedded system to run a real-time operating system. When it

comes to the embedded system the main focuses is on implementing the controller in the real-time operating system, and evaluate its suitability to run the controller.

1.3 Results

The sequence of work during this master thesis can be divided into three parts, first the design and building of the actual mobile inverted pendulum, the second result is the developed control system controlling the pendulum and the third is the functionality of the Open Source Real-Time Operating System.

First, the mobile inverted pendulum were designed and built. Aluminum frames, wheels, electronic components and other parts where put together to construct the pendulum. A simple hardware design were chosen, which made the building of the pendulum easier.

The control system developed to control the pendulum work in a satisfying way, mathematical models and simulations in Matlab and Simulink were used in order to calculate control parameters used in the control system. Two different controller types where tried for balancing and driving, one PID controller and one LQR controller. The LQR controller gave the most satisfying performance.

The open source real-time OS has been working very well. It was easy to get into and get it started. The real-time support trough Xenomai has been satisfying when following the guidelines on how to develop real-time applications.

2 Methods

This chapter will present the theory behind the methods used during the work.

2.1 Mathematical description

To be able to simulate the system in Matlab/Simulink a mathematical model which describes the dynamics of the system has to be developed. Two different models are developed. One which describes the dynamics of mechanical system and another one which describes the dynamics of the accelerometer used for estimation of the pendulum angle. Friction and wind resistance is neglected in the mechanical model.

2.1.1 Inverted pendulum mathematical model

A mathematical description of the inverted pendulum is needed to be able to simulate the behavior of the system in a correct way. The mechanical construction is divided into two parts: The wheels and the pendulum. On each part the affecting forces and torques are determined. Table 2-1 contains a description of the variables used in the mathematical description.

g	m/s^2	$Acceleration\ of\ gravity$
θ	rad	$Angle\ between\ the\ pendulum\ and\ a\ vertical\ line$
ϕ	rad	$Angle\ between\ the\ wheel\ and\ a\ vertical\ line$
T_m	Nm	$Torque\ preduced\ between\ the\ pendulum\ and\ the\ wheel$
m_p	kg	$Mass\ of\ the\ pendulum$
r_p	m	$Distance\ from\ the\ center\ of\ mass\ to\ the\ bottom\ the\ pendulum$
I_p	kgm^2	$Inertia\ of\ the\ pendulum$
m_w	kg	$Mass\ of\ the\ wheel$
r_w	m	$Radius\ \ of\ the\ wheel$
I_w	kgm^2	$Inertia\ of\ the\ wheel$
F_p	N	$Vertical\ force\ (see\ Figure\ 2-1)$
F_a	N	$Horisontal\ force\ (see\ Figure\ 2-1)$
F_f	N	$Tangential\ force\ (see\ Figure\ 2-2)$
F_{nw}	N	$Normal\ force\ which\ act\ on\ the\ wheel$
r_m	Ohm	$Resistance\ of\ the\ motor\ windings$
k_v	Vs/rad	$Motor\ backemf\ constant$
k_t	Nm/A	$Motor\ torque\ constant$

Table 2-1 Description of variables.

The fundamental equation used to derive the models is Newton's second law, which says that the force applied to a particle is proportional to the change of the particles velocity in the force direction [3]. This law is also possible to apply to rotational particles were its results is which gives that the applied torque is proportional to the change in rotational velocity. The equations are

$$\sum F = m\ddot{x} \tag{2.1}$$

$$\sum T = J\ddot{\theta} \tag{2.2}$$

First, the forces and torques on the pendulum are investigated, see Figure 2-1.

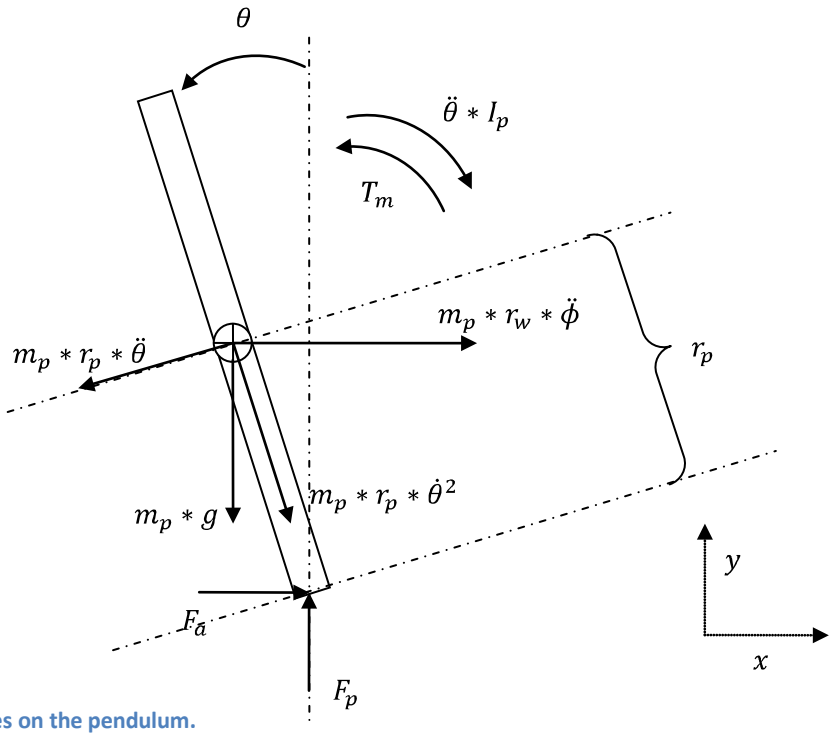


Figure 2-1 Forces and torques on the pendulum.

Forces acting on the pendulum seen from a plane perpendicular to the pendulum are

$$m_p\ddot{x}\cos\theta = F_a\cos\theta + F_P\sin\theta - m_p g\sin\theta - m_p r_p\ddot{\theta} \tag{2.3}$$

The moments acting about the center of gravity of the pendulum are

$$I_p\ddot{\theta} = -F_a\cos\theta\, r_p - F_P\sin\theta\, r_p - T_m \tag{2.4}$$

The linear acceleration of the pendulum can be determined from the wheels angular acceleration

$$\ddot{x} = \ddot{\emptyset}\, r_w \tag{2.5}$$

The linear relation which transfer applied terminal voltage to torque is a model of an ideal DC-motor. Two motors are used, this means that the torque is multiplied with two to get the total torque.

$$T_m = \frac{2k_t V}{r_m} - \frac{2k_t k_v\dot{\emptyset}}{r_m} \tag{2.6}$$

When combining the equations (2.3-2.6) an expression for the pendulums angular acceleration can be derived:

$$\ddot{\theta} = \frac{m_p g r_p r_m\sin\theta - m_p\ddot{\emptyset} r_w r_p r_m\cos\theta + 2k_t k_v\dot{\emptyset} - 2k_t V}{r_m(m_p {r_p}^2 + I_p)} \tag{2.7}$$

If the forces acting in the x direction on the pendulum are added up the following expression is given

$$m_p\ddot{x} = F_a + m_p r_p \dot{\theta}^2 \sin\theta - m_P r_P \ddot{\theta}\cos\theta \tag{2.8}$$

Next the forces and torques on the wheels are investigated, see Figure 2-2.

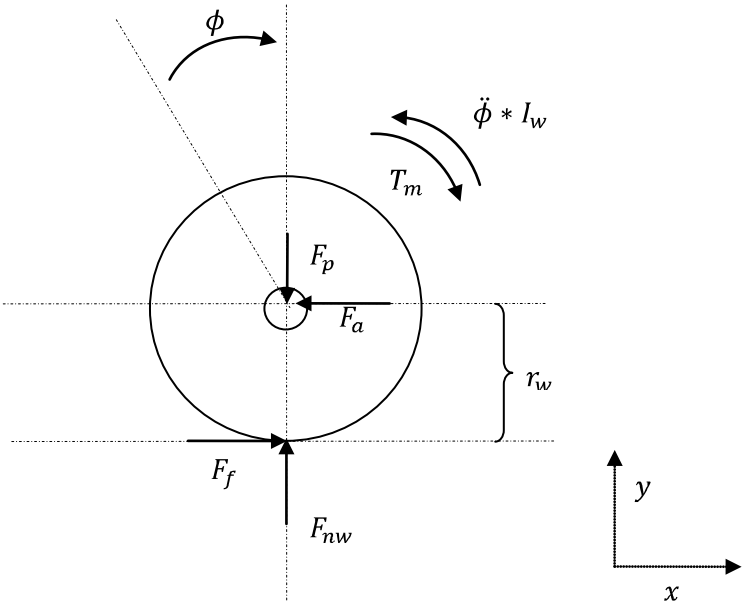


Figure 2-2 Forces and torques on the wheels.

The sum of the torques around the center of the wheel is

$$I_w\ddot{\emptyset} = T_m - F_f r_w \tag{2.9}$$

Forces acting in x direction on the wheel

$$m_w\ddot{x} = F_f - F_a \tag{2.10}$$

By combining equations (2.5) and (2.8-2.10) we get the following expression for the wheels angular acceleration:

$$\ddot{\emptyset} = \frac{r_m r_w m_p r_p \dot{\theta}^2 \sin\theta - r_m r_w m_p r_p \ddot{\theta}\cos\theta - 2k_t k_v \dot{\emptyset} + 2k_t V}{r_m(I_p + m_p {r_w}^2 + m_w {r_w}^2)} \tag{2.11}$$

When combining equations (2.7) and (2.11) two new expressions for $\ddot{\theta}$and $\ddot{\emptyset}$ can be formed

$$\ddot{\theta} = \frac{\beta\gamma g \sin\theta + \gamma r_w \cos\theta \left(2k_t\left(k_v\dot{\phi} - V\right) - \gamma r_w \sin\theta\, \dot{\theta}^2\right) + 2\beta k_t\left(k_v\dot{\phi} - V\right)}{(\alpha\beta - \gamma^2 r_w \cos\theta)} \quad (2.12)$$

$$\ddot{\emptyset} = \frac{\gamma r_w \sin\theta \left(\alpha\dot{\theta}^2 - \gamma g \cos\theta\right) + 2\gamma r_w \cos\theta\, k_t\left(V - k_v\dot{\phi}\right) + 2\alpha k_t\left(V - k_v\dot{\phi}\right)}{(\alpha\beta - \gamma^2 r_w \cos\theta)} \quad (2.13)$$

where

$$\alpha = r_m(m_p {r_p}^2 + I_p) \quad (2.14)$$

$$\beta = r_m(I_p + m_p {r_w}^2 + m_w {r_w}^2) \quad (2.15)$$

$$\gamma = m_p r_p r_m \quad (2.16)$$

To be able to design a linear quadratic controller for the pendulum, equation (2.12) and (2.13) must be linearized around the operating point which is $\theta, \dot{\theta}, \emptyset, \dot{\emptyset} = 0$. The linearized system on state space form is:

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\emptyset} \\ \ddot{\emptyset} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\beta\gamma g - {r_w}^2\gamma^2}{\beta\alpha - \gamma^2 {r_w}^2} & 0 & 0 & \frac{2k_t k_v(\gamma r_w + \beta)}{\beta\alpha - \gamma^2 {r_w}^2} \\ 0 & 0 & 0 & 1 \\ \frac{\gamma r_w \alpha - \gamma^2 r_w g}{\beta\alpha - \gamma^2 {r_w}^2} & 0 & 0 & \frac{2k_t k_v(-\gamma r_w - \alpha)}{\beta\alpha - \gamma^2 {r_w}^2} \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ \emptyset \\ \dot{\emptyset} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2k_t k_v(-\gamma r_w - \beta)}{\beta\alpha - \gamma^2 {r_w}^2} \\ 0 \\ \frac{2k_t k_v(\gamma r_w + \alpha)}{\beta\alpha - \gamma^2 {r_w}^2} \end{bmatrix} V \quad (2.17)$$

2.1.2 Sensor model

The accelerometer used to estimate the angle of the pendulum can measure two perpendicular accelerations, both these measurements are effected when the pendulum changes angle. Table 2-2 describes the additional variables and parameter used in the sensor model.

s_x	m/s^2	$Measurement\ value\ from\ the\ accelerometer\ x\ axle$
s_y	m/s^2	$Measurement\ value\ from\ the\ accelerometer\ y\ axle$
r_s	m	$Distance\ from\ the\ pendulums\ rotational\ center\ to\ center\ of\ the\ accelerometer$

Table 2-2 Description of variables.

We get the following model for the measurements sampled from the sensor, see Figure 2-3:

$$s_x = g\sin\theta - \ddot{x}\cos\theta - \ddot{y}\sin\theta \tag{2.18}$$

$$s_y = -g\cos\theta - \ddot{x}\sin\theta - \ddot{y}\cos\theta \tag{2.19}$$

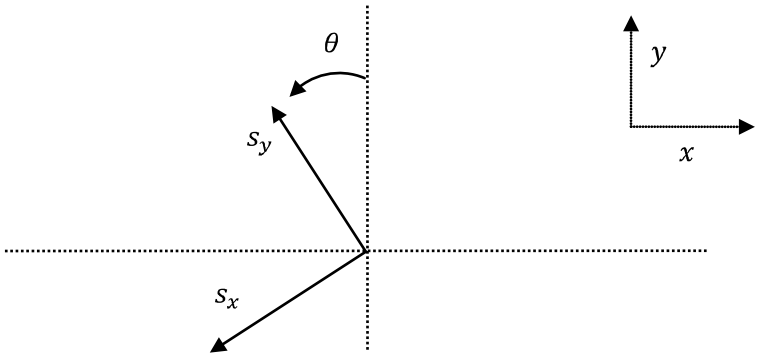


Figure 2-3 Axis placement for the accelerometer.

Where the variables s_x and s_y are representing a values measured from the sensor. Assuming rigid bodies, we have:

$$\ddot{x} = r_w\ddot{\phi} - r_s\ddot{\theta}\cos\theta \tag{2.20}$$

$$\ddot{y} = -r_s\ddot{\theta}\sin\theta \tag{2.21}$$

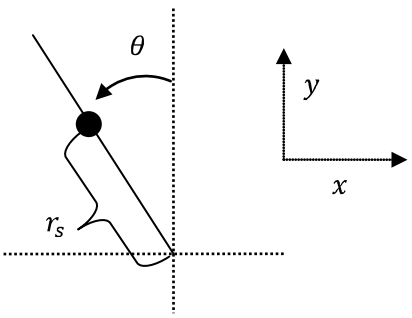


Figure 2-4 Displacement of the accelerometer from the rotational centre.

By placing the sensor close to the rotational center the variable r_s becomes small, when placed very close the term $r_s\ddot{\theta}\sin\theta$ and $r_s\ddot{\theta}\cos\theta$ can be neglected. If this simplification is used the equations (2.18-2.19) can be rewritten to a simpler form.

$$\ddot{x} = r_w\ddot{\phi} \tag{2.22}$$

$$\ddot{y} = 0 \tag{2.23}$$

$$s_x = g\sin\theta - \ddot{x}\cos\theta \tag{2.24}$$

$$s_y = -g\cos\theta - \ddot{x}\sin\theta \tag{2.25}$$

By combining the equations (2.24) and (2.25) and using the trigonometric unity we can exclude the linear acceleration $\ddot{x}$, the variables s_x and s_y are known by sampling the accelerometer.

$$f(\theta) \equiv s_x\sin\theta + s_y\cos\theta - g \tag{2.26}$$

$$f'(\theta) = s_x\cos\theta - s_y\sin\theta \tag{2.27}$$

The resulting equation (2.26) can be solved numerically to find the value of θ.

2.1.3 Angle measuring methods

To acquire the angle of the pendulum two different methods can be used. The first one is to calculate arcus sinus of the sensor value, s_x, and the second is to use a numerical method to find the roots of equation (2.26). When using the first method the linear acceleration will affect the calculated angle, and in the second one the influence of $\ddot{x}$ is removed. A description of a numerical method (Newton-Raphson) is further described in section 2.5.7.

2.2 Control theory

The inverted pendulum is an unstable and nonlinear system. To make the pendulum balancing by itself some kind of control strategy has to be implemented. The following chapter will handle the theory behind candidates for controlling the inverted pendulum.

2.2.1 PID Controller

The PID (Proportional Integral Derivative) controller is the most widely spread feedback controller [4]. The control signal *u* for a PID controller is formed from the control error *e*, which is the difference between the reference signal *r* and the feedback signal *y*. The controllers task is to keep *e* as close to zero as possible. The mathematical description of the PID in the time domain is as follows

$$u = K_p e + K_i\int_0^t e(\tau)d\tau + K_d\frac{de}{dt} \tag{2.28}$$

And in the frequency domain

$$F_{PID} = K_p + \frac{K_i}{s} + K_d s \tag{2.29}$$

There are three different parts which form the output signal, proportional, integral and derivative part. The proportional part amplify the error *e* by the controller constant K_p. In general, increased proportional constant makes the controller faster and better to compensate for process disturbances but also decreases the stability margins and increases the control signal activity. If *e* is small and constant it can sometimes happen that the proportional part of the controller cannot fully compensate for the error. The integral action of the controller integrates the error over time which makes the control signal increase and forces the error to approach zero. The constant K_i is called the integral constant and determines the gain of the integral action for the controller. The integral effect gives the controller better ability to compensate for low frequency disturbances but decrease the stability margins. To make the controller able to react better on fast process disturbances derivative action can be incorporated. Derivative in the controller normally provide better stability margins. This can sometimes compensate for the decreased stability margins the controller gets if integral action is used. However, with high derivative action the high frequency gain is increased. This makes the controller more sensitive to measurement noise. The parameter K_d sets the gain of the controller derivative action [5].

2.2.2 State feedback

A system given in state space form can be stated as

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + B_v v(t) \\ y(t) &= Cx(t) \end{aligned} \tag{2.30}$$

The system is considered to be linearized where $u(t)$ is the input signal and $v(t)$ is a disturbance signal. Figure 2-5 depicts the principles of a state feedback system.

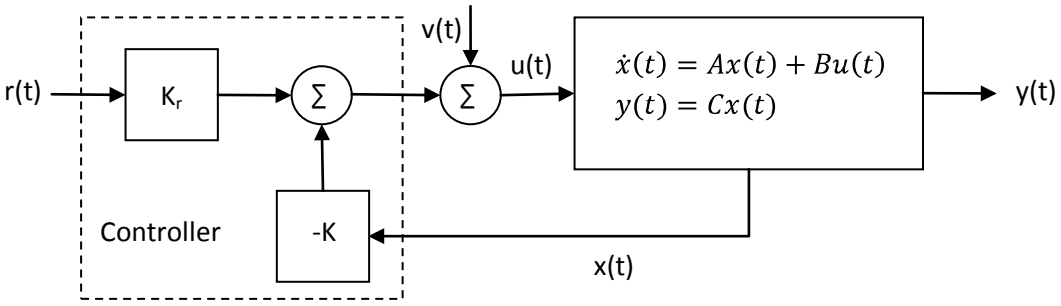


Figure 2-5 State feedback control system

Assume that all states of the system are measured the control law can be written as

$$u = -Kx(t) + K_r r(t) \tag{2.31}$$

The system dynamics can be decided by for example placing the closed loop system poles. This can be done by considering the closed loop system characteristic polynomial. When the feedback is applied the system can be described as

$$\dot{x}(t) = (A - BK)x(t) + BK_r r(t) + B_v v(t) \tag{2.32}$$

In this description it is the system matrix $(A - BK)$ with the corresponding characteristic polynomial $det(sI - A + BK) = 0$ which decides the closed loop poles. It is desirable to place the poles so that the system becomes asymptotically stable around its equilibrium point. This kind of controller design is called pole placement. The constant K_r does not affect the systems stability but it affect the steady state solution. Therefore, the gain constant K_r is chosen so that the low frequency gain from r to y for the system becomes one. K_r can then be calculated from

$$K_r = \frac{1}{C(-A + BK)^{-1}B} \tag{2.33}$$

A necessary requirement for the pole placement design to work is that all the systems states are controllable. A state is controllable when it is possible to control a state from $x(0) = 0$ to an arbitrary $x = x^*$ in finite time. The complete system is controllable when all the states are controllable [5].

Placing the poles to get the desired behavior of the system is not a trivial task [5]. Another way to choose the feedback gains is to minimize a quadratic cost function. This is called optimal control. One of the most common optimal controllers is the LQR (Linear Quadratic Regulator) controller. The quadratic criterion to be minimized can be

$$J = \int_0^\infty (x^T(t)Q_x\, x(t) + u^T(t)Q_u u(t))dt \tag{2.34}$$

The matrices $Q_x > 0$ and $Q_u > 0$, are symmetric and positive definite gain matrices. The cost function represents the tradeoff between the distance for the states from origin and the cost of control input. The gain K (matrix) is determined by first solving the algebraic Riccati equation

$$PA + A^T P - PBQ_u^{-1}B^T P + Q_x = 0 \tag{2.35}$$

where P is a positive semi definite matrix. Then K can be calculated as

$$K = Q_u^{-1}B^T P \tag{2.36}$$

This calculation can be hard to do by hand but, for example, in MATLAB the *lqr* command can be used [4].

The next step is to choose the gain matrices Q_x and Q_u. One way is to make the Q matrices diagonal, that is to say only scalars on the diagonal in the matrices, and all other entries zero. Then each state and input can be weighted to make their contribution to the cost function [4].

The LQR controller has no built-in integral action which apply that K_r has to be very well calibrated to get the correct steady state gain. K_r is calculated from the model of the system and therefore the

model has to be correct to get a well calibrated K_r. To introduce integral action in a state feedback controller the system model has to be augmented with an integral state. The integral action of the controller helps to bring the steady state error down to zero even when the model has some uncertainty. The idea of integration in the LQR controller is the same as for the PID controller described earlier, to integrate the control error so the controller can compensate for small output deviations from the reference signal. The augmented system description is

$$\begin{bmatrix}\dot{x}(t)\\ \dot{z}(t)\end{bmatrix} = \begin{bmatrix}A & 0\\ C & 0\end{bmatrix}\begin{bmatrix}x(t)\\ z(t)\end{bmatrix} + \begin{bmatrix}B\\ 0\end{bmatrix}u(t) + \begin{bmatrix}0\\ I\end{bmatrix}r(t) \tag{2.37}$$

where z is the state which integrates the controller error. Now a new control has to make use of the integration. The new law is

$$u = -Kx(t) + K_r r(t) - K_i z(t) \tag{2.38}$$

This is almost the same as before though with the modification that also the integral state is sent back via the gain K_i [4].

2.2.3 State estimation

In reality it is not realistic that all the systems states are measured by sensors which can send back information to the controller. However it is possible to estimate the system states from a few measurements and with help from the mathematical model. This technique is called to create an observer. The system which the observer should be created for must be observable, that is to say the system states can be determined by investigating the input and the output for the system. The system is not allowed to have any hidden dynamics [4].

Assume there is a system on the form in Equation (2.30). If the system output is measured with sensors there will always be some measurement noise. The real output and the output from the model will have a difference. If $y_m(t)$ is the measured output and $w(t)$ is the measurement noise, the following relation holds:

$$y_m(t) = Cx(t) - w(t) \tag{2.39}$$

The sign of the disturbance signal is not really of any importance just the absolute value. The observer can first be stated as the original state description of the system with the difference that the state vectors have been provided with a hat symbol. This symbolize that the states are estimated.

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) \tag{2.40}$$

This description of the system is of the same kind as the one we used before. The difference will be when the measured output signal with added noise is incorporated in the observer. Then the difference between the estimated output from the model, $\hat{y}(t) = C\hat{x}(t)$ and the measured output $y_m(t)$ is used to correct the estimated states. The difference is multiplied with a gain matrix $K_y$to define the correction. The magnitude of K_y determines how large the correction of the states should be. This is connected to the measurement noise. If the measurement noise is large, the difference between the measured output and the output from the model will be larger. This can be seen from equation (2.39). A large K_y means that very much trust is put into the measurements, which requires

very accurate measurements with small noise. If it is the other way around, K_y is small, more trust is put into the model. The complete observer is described by

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + K_y(y_m(t) - C\hat{x}(t)) \tag{2.41}$$

If the description of the observer is rearranged it can be expressed like

$$\dot{\hat{x}}(t) = (A - K_yC)\hat{x}(t) + Bu(t) + K_y y_m(t) \tag{2.42}$$

Now it is easy to see the observers system matrix $(A - K_yC)$. The observer dynamics is determined by the eigenvalues of the system matrix [5].

To investigate the performance of the observer one can study the estimation error which is denoted as

$$\tilde{x}(t) = x(t) - \hat{x}(t) \tag{2.43}$$

The error describes what the effect of process and measurement disturbances has on the observed states and also how fast the estimated states converge towards the true states. If the state equation in (2.30) is combined with the measurement disturbance effect (2.39) and the observer in (2.41), the result is a representation of the estimation error:

$$\dot{\tilde{x}}(t) = \big(A - K_yC\big)\tilde{x}(t) + B_v v(t) + K_y w(t) \tag{2.44}$$

When the estimated states from the observer are sent back to the controller the control law has to be modified to use the estimated states instead. The new feedback law is

$$u(t) = -K\hat{x}(t) + K_r r(t) \tag{2.45}$$

where $\hat{x}$ is the output from an observer like (4.14). It is now possible to make a model of the closed loop system. If the system to be controlled is of order n, the new closed loop system including the observer will have the dimension $2n$ because the observer is of the same order as the system. The states of the closed loop are $x(t)$ and $\tilde{x}(t)$. Combining the relation (2.30), (2.41), (2.44) and (2.45) results in the system

$$\begin{aligned} \dot{x}(t) &= (A - BK)x(t) + BK\tilde{x}(t) + BK_r r(t) + B_v v(t) \\ \dot{\tilde{x}}(t) &= \big(A - K_yC\big)\tilde{x}(t) + B_v v(t) + K_y w(t) \end{aligned} \tag{2.46}$$

The control law has to be rewritten again when using $\tilde{x}(t)$as states in the closed loop

$$u(t) = -Kx(t) + K\tilde{x}(t) + K_r r(t) \tag{2.47}$$

Then the system can be represent in matrix form as

$$\begin{bmatrix}\dot{x}(t)\\ \dot{\tilde{x}}(t)\end{bmatrix} = \begin{bmatrix}(A-BK) & BK\\ 0 & A-K_yC\end{bmatrix}\begin{bmatrix}x(t)\\ \tilde{x}(t)\end{bmatrix} + \begin{bmatrix}BK_r\\ 0\end{bmatrix} r(t) + \begin{bmatrix}B_v\\ B_v\end{bmatrix} v(t) + \begin{bmatrix}0\\ K_y\end{bmatrix} w(t)$$

$$\begin{bmatrix}y(t)\\ y_m(t)\\ u(t)\end{bmatrix} = \begin{bmatrix}C & 0\\ C & 0\\ -K & K_u\end{bmatrix}\begin{bmatrix}x(t)\\ \tilde{x}(t)\end{bmatrix} + \begin{bmatrix}0\\ 0\\ K_r\end{bmatrix} r(t) + \begin{bmatrix}0\\ -I\\ 0\end{bmatrix} w(t) \quad (2.48)$$

Figure 2-6 depicts the state feedback with an observer [4, 5].

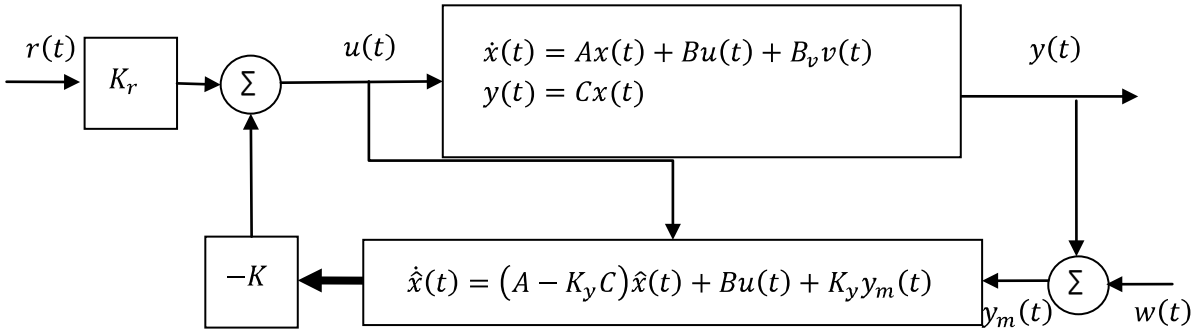


Figure 2-6 State feedback with observer.

In the same way as for the state feedback matrix K optimization can be used for the determination of the observer gain K_y. The estimation goal is to minimize the error $\tilde{x}$. When an observer gain matrix is determine that way it is called a Kalman filter. The Kalman filter and the optimization method for deciding the gain is described in more detail in Section 2.2.4. A combination of the optimal controller LQR and a Kalman filter is called LQG (Linear Quadratic Gaussian) controller. The LQG deals with Gaussian distributed additive noise on both the system and the observer states [5].

2.2.4 Kalman filter

The Kalman filter can be used for estimating the states of a system where the measurements include noise. If the noise is Gaussian distributed, an optimal estimator that minimizes the variance of the estimation error can be derived [6].

Consider the following discrete time system where the time step h is equal to one and the disturbances $v_1(k)$, $v_2(k)$ are Gaussian.

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) + v_1(k)\\ y(k) &= Cx(k) + v_2(k) \end{aligned} \quad (2.49)$$

If the matrix $CP(k|k-1)C^T + R_2$ is positive definite, the optimal filter can be calculated by [6]:

$$\hat{x}(k+1|k) = \Phi\hat{x}(k|k-1) + \Gamma u(k) + K(k)(y(k) - C\hat{x}(k|k-1)) \tag{2.50}$$

$$K(k) = (\Phi \mathrm{P}(k|k-1)C^T + R_{12})(CP(k|k-1)C^T + R_2)^{-1} \tag{2.51}$$

$$\begin{aligned} &P(k|k) \\ &= [I \quad -K(k|k-1)] \begin{bmatrix} \Phi \mathrm{P}(k|k-1)\Phi^{\mathrm{T}} + \mathrm{R}_1 & \Phi \mathrm{P}(k|k-1)\mathrm{C}^{\mathrm{T}} + \mathrm{R}_{12} \\ \mathrm{CP}(k|k-1)\Phi^{\mathrm{T}} + \mathrm{R}_{12} & \mathrm{CP}(k|k-1)\mathrm{C}^{\mathrm{T}} + \mathrm{R}_2 \end{bmatrix} \begin{bmatrix} \mathrm{I} \\ -\mathrm{K}^{\mathrm{T}}(k|k-1) \end{bmatrix} \end{aligned} \tag{2.52}$$

$$\begin{aligned} P(k|k) = \mathrm{P}(k|k-1) \\ - \mathrm{P}(k|k-1)C^T CP(k|k-1)(CP(k|k-1)C^T + R_2)^{-1} CP(k|k-1) \end{aligned} \tag{2.53}$$

$$P(0|-1) = R_0 \tag{2.54}$$

$$R = \begin{bmatrix} R_1 & R_{12} \\ R_{12} & R_2 \end{bmatrix} = Cov\{v_1, v_1\} \tag{2.55}$$

The matrix R is the covariance of the disturbance and the measured states. By combining equations (2.51) and (2.52) it is possible to derive Equation (2.53), which gives $P(k|k)$. This equation is a stationary or algebraic Riccati equation, and by solving it the Kalman gain $K(k|k)$ can be derived.

Using a Kalman filter gives the possibility to use signal fusion by combining several sensors measuring different states.

2.3 Simulations

The purpose of the simulations is to investigate the performance of, and design, controllers of different types in Matlab/Simulink. By simulation it is also possible to test different parameters like weights, heights and placement of the different components. The goal is to construct a model which is as close enough to the real system, such that the control parameters from the simulations can be more or less directly implemented in the real controller. Focus is put into development and testing two different kinds of controllers, PID and LQR controllers. Both controllers are tested and evaluated together with Kalman filtering of the measured units. The LQR controller is also simulated together with a state observer.

To make the overview of the model simpler, the regulator, the vehicle, the sensors and the state observer with the Kalman filter is modeled in separate subsystems (See Figure 2-7). Input to the vehicle model is voltages applied to the motors. The subsystem for the vehicle has also an input for simulating a process disturbance. The part which represents the sensors is provided with inputs for measurement noise. The state observer gets the sensor measurements and the control signal as input and leaves the estimation of the states as output.

The mathematical description of the system (2.17) represents the vehicle dynamics in four states. State three, $\dot{\emptyset}$, in the state space representation is not used to describe any of the other states. For simplicity and faster calculations this state can therefore be ignored in the state observer. The profit in faster calculations will of course be most obvious when the running on the real system. The output from the block with state observer and Kalman filter is then the three remaining states. The

simulations are carried out under the assumption of that the angular and the angular velocity of the pendulum is measured with sensors and the angular velocity of the wheels are estimated by the observer. Figure 2-7 shows a simplified overview of the Simulink model. The saturation after the controller restricts the control output to be within the range -12V to +12V. The complete model can be found in Appendix A.

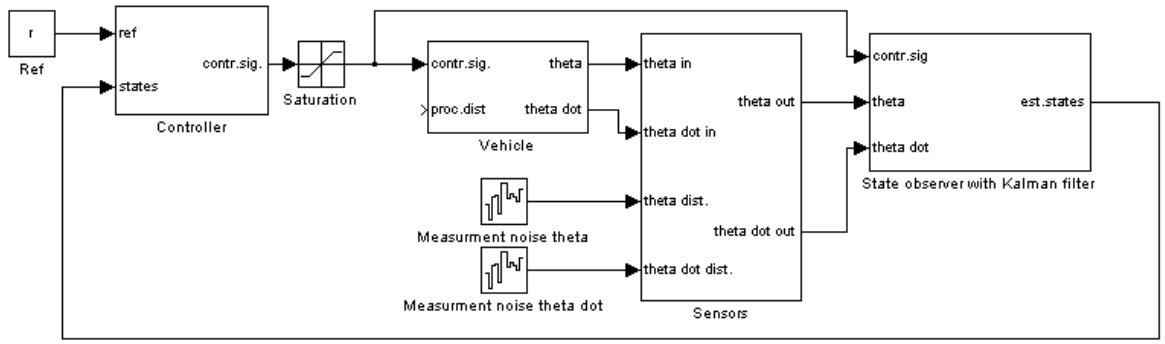


Figure 2-7 Simplified Simulink model.

A first verification of the pendulum model is done by applying a constant step voltage of one volt on the input to the open loop system. The step is applied to the system after half a second. The expected behavior would be that the pendulum falls down from upright position and stabilize around $-180°$. Figure 2-8 verifies this behavior.

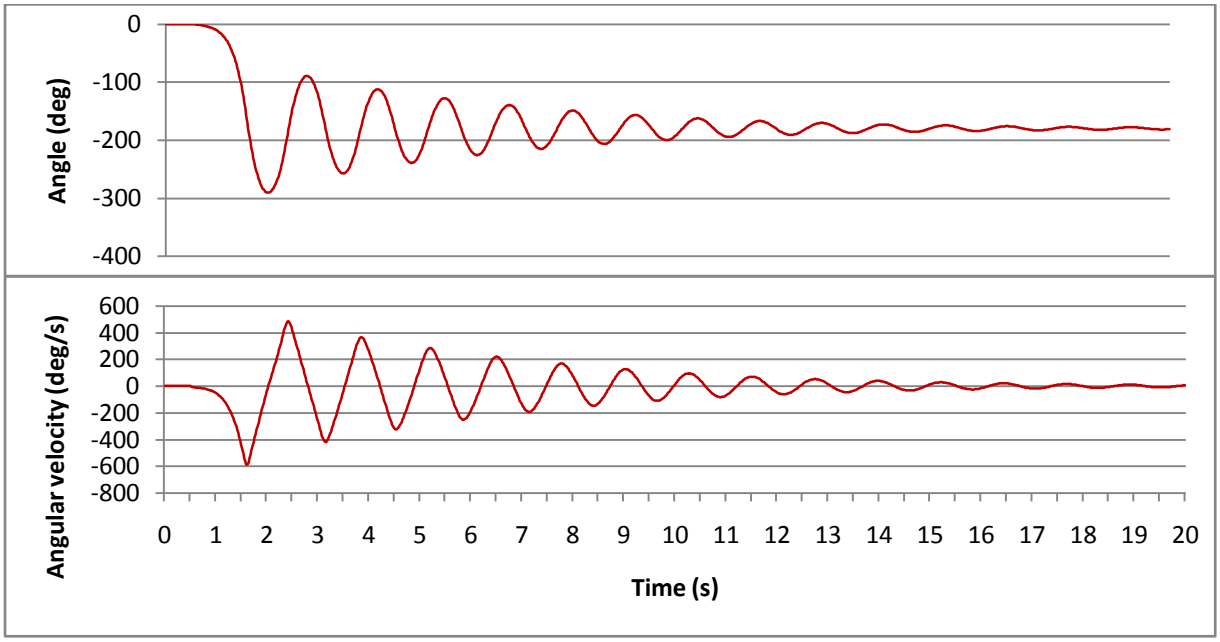


Figure 2-8 Response to unit step voltage on open loop system.

The measurements noise which is applied to the sensor subsystem is taken from the data sheets for possible candidates of angular and angular velocity sensors. The noise for the angular sensor is $0{,}00173\ rad\ rms$ and $0{,}00552\ rad/s\ rms$ for the angular velocity sensor.

The sensor signals are then put into the state observer with Kalman filter. The Kalman filter gains are calculated with the Matlab command $kalman$. Besides the system matrices the kalman command

also needs the covariance matrix for process disturbances, R_1, and the covariance matrix for measurement disturbance, R_2. The ratio between these two matrices, R_1/R_2, decides how much trust the filter should put into the system model versus the sensor measurements. If the ratio is small, that is, if R_2 is large compared to R_1 the kalman gains becomes smaller and less trust will be put into the sensor measurements and more into the model.

A filter which put much trust into the noisy sensor measurements also gives a noisy estimation, but the filter will be fast. If much trust is put on the model the filter will suppress the measurement noise more efficiently and the estimation will become smoother but a bit lagged from the true value.

Simulations in Simulink show that values on R_1 and R_2 for a good performance of the Kalman filter is when the ratio are 1/10000 for all the three states. This gives an adequate noise reduction and at the same time a quite fast filter.

$$R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_2 = [10000] \tag{2.56}$$

Figure 2-9 depicts a simulation of the Kalman filter.

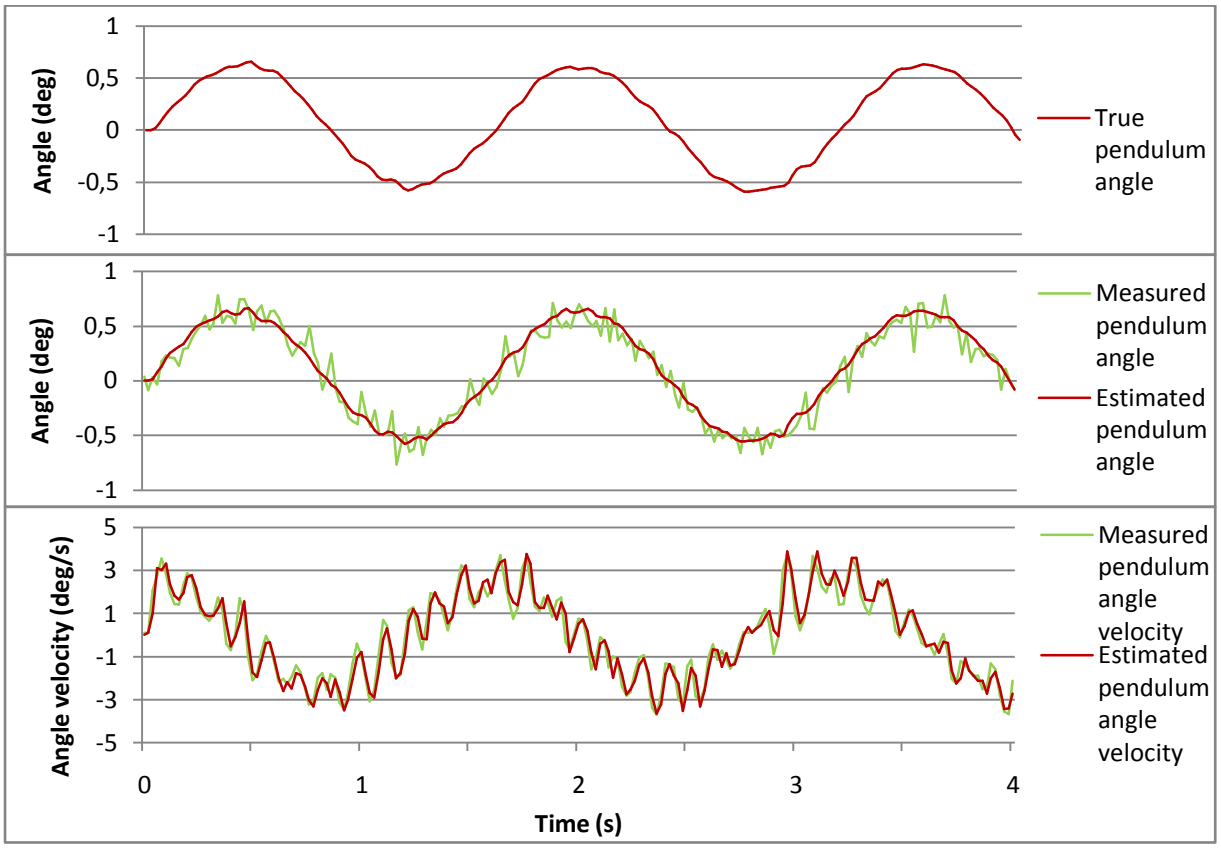


Figure 2-9 Simulation of Kalman filter with covariance matrices ratio 1/10000.

A PID controller is connected to the vehicle with feedback of the pendulum angle through the Kalman filter. The reference signal on the angle is set to zero degrees. A process disturbance on the pendulum angle, formed as a pulse, is applied to the vehicle model during the simulation. The disturbance should represent a nudge on the pendulum and is applied after one second. The control parameters K_P, K_I and K_D are tested out by trial and error until the pendulum can compensate for

the nudge without falling. Figure 2-10 depicts a simulation with a PID controller control parameters set to $K_P = 400$, $K_I = 350$ and $K_D = 1$.

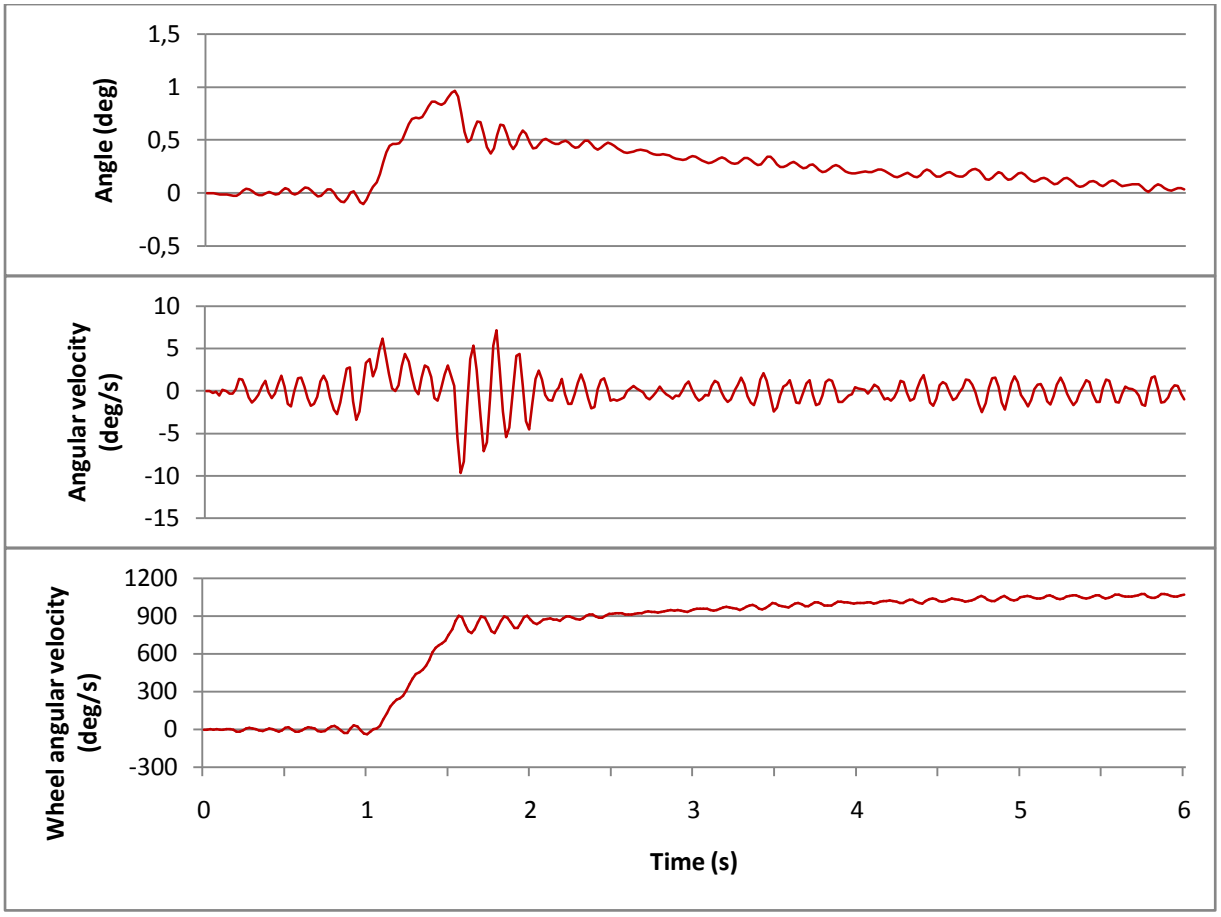


Figure 2-10 Simulation with PID controller.

After the nudge on the pendulum the controller restore the pendulum angle around zero degrees. On the angular velocity one can see that the nudge cause a small positive speed at the start and then a negative speed which is an expected behavior. The PID controller in this performance only cares about the angle of the pendulum. This can be visualized when looking at the angular velocity of the pendulum wheels in Figure 2-10. When the pendulum angle is back to zero the wheel speed is still positive and hence the pendulum has a continued linear speed. This has to do with the integral action in the controller. The integral has accumulated the positive angular error during the process disturbance and still causing an output signal even if the angle error is very small. On the real system this behavior will probably not be so obvious because the model used in Simulink is ignore losses in form of rotational friction and wind resistance. These losses will likely have the effect that the pendulums linear speed will decrease.

When the LQR controller should be evaluated and dimensioned in Matlab the function $lqrd$ is used for calculating the gains. The $lqrd$ command minimize the quadratic criterion (2.34) using the linearized system matrices and the weight matrices Q_x and Q_u. The weight matrix Q_x is chosen to be diagonal with the same size as the number of states in the feedback loop:

$$Q_x = \begin{bmatrix} q_{x1} & 0 & 0 \\ 0 & q_{x2} & 0 \\ 0 & 0 & q_{x3} \end{bmatrix} \tag{2.57}$$

The entries in the Q_x matrix represent the weighting of the system states. A higher number in the matrix makes the specific state contribute more in the minimizing function. Hence, a higher value should be put in the states which are important to remain small. The Q_u matrix is just a scalar because the controller only has one input signal. A higher value of Q_u attenuates the output from the controller. Figure 2-11 shows the same simulation as with the PID controller but now with the LQR controller instead.

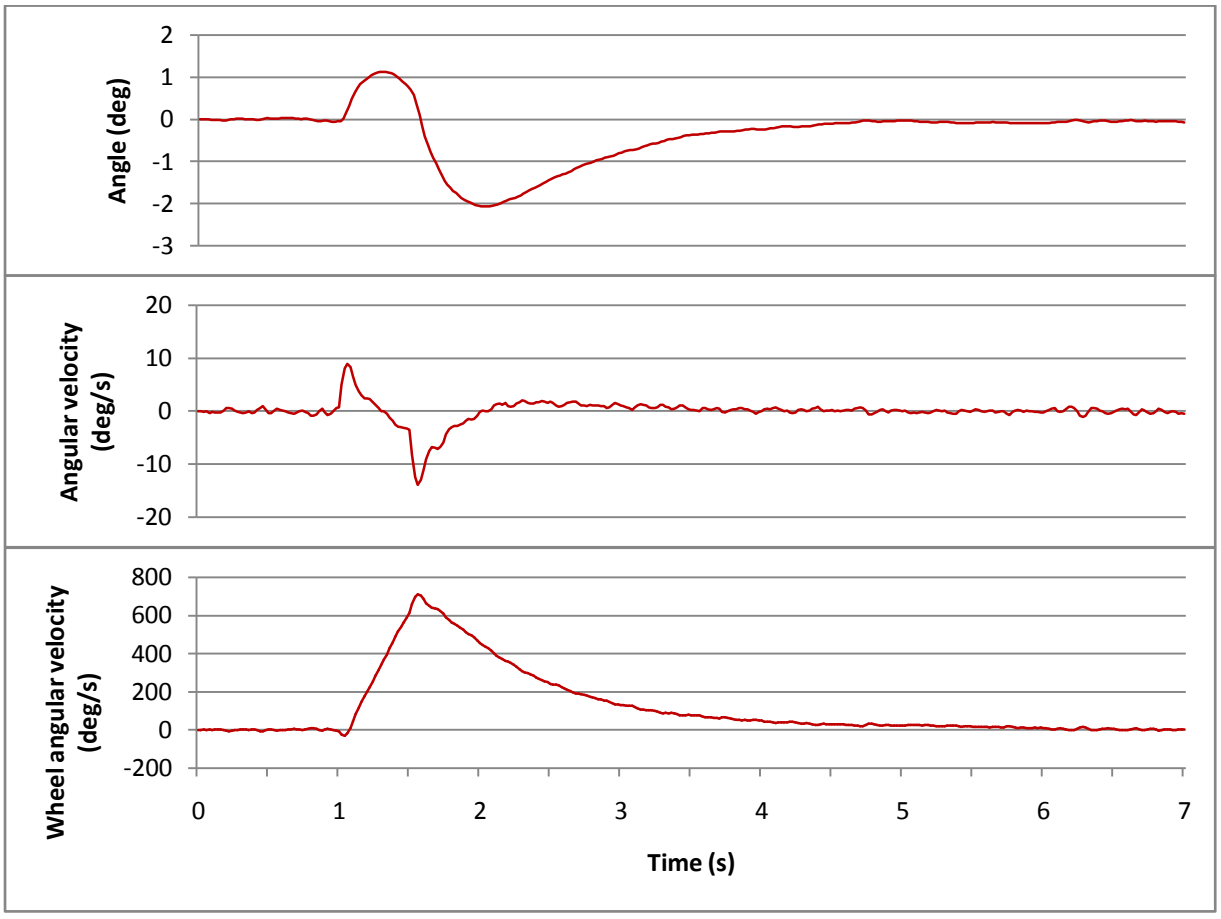


Figure 2-11 Simulation of pendulum system with LQ controller.

The weighting matrixes are by trial and error chosen to

$$Q_x = \begin{bmatrix} 200000 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Q_u = [10] \tag{2.58}$$

And this gives the feedback gains

$$K = [-142{,}1 \quad -17{,}1 \quad -1{,}16] \tag{2.59}$$

The most important state to follow is the pendulum angular. Therefore a high value is put in the first entry of the Q_x matrix.

The LQR controller is trying to force all the systems states to zero. There is no integral action in this LQR controller, so the problem with accumulated integral as in the simulation with the PID controller is not a problem here.

2.3.1 Simulation of angle measuring

To measure the angle of the pendulum two methods have been used, also mentioned in Section 2.1.3. To evaluate these two methods simulations are performed using Matlab. The simulation uses constructed values that represent sampled values from the accelerometer on the robot. These values are constructed according to equations (2.24) and (2.25), this data are shown in Figure 2-12.

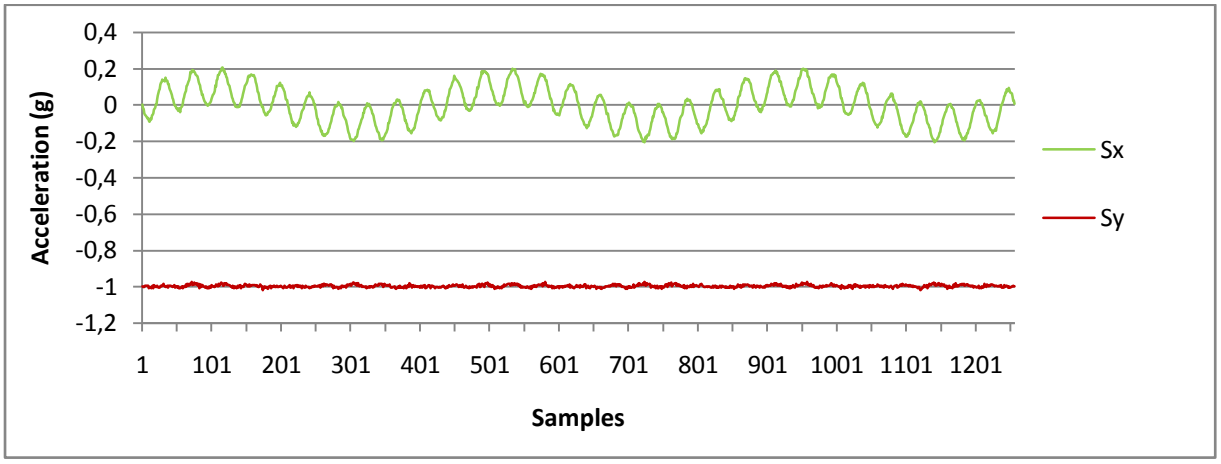


Figure 2-12 Simulated accelerometer values.

It can be seen that the values consist of two sinusoids, one of low frequency, representing changing θ, and one high frequency, representing changes in $\ddot{x}$. There is also noise added to the values to give more realistic values.

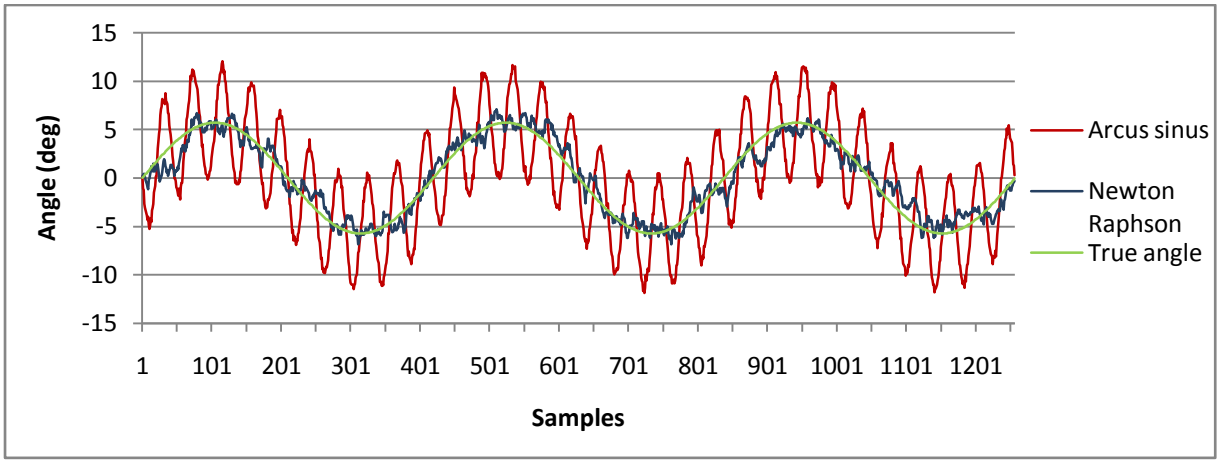


Figure 2-13 Simulated Arcus sinus and Newton Raphson performance.

Figure 2-13 illustrates the angle given by the two methods and the correct angle. It can be seen that the method which only use arcus sinus is heavily affected by the linear acceleration $\ddot{x}$ and that the method using Newton Raphson has the ability to remove much of the linear acceleration and gives a more accurate value of the angle θ.

2.4 Movement control

When controlling the linear movement of the pendulum, different behaviors of the two controllers were found.

When using the PID controller the reference signal is θ, i.e. the pendulums angle. To make the pendulum move forward and backwards the reference signal is set to make the pendulum lean a little bit forward or a little bit backward. This way of controlling the movement makes the velocity of the wheels constantly increasing as long as the reference signal is not equal to zero. The consequence of this is that if the reference signal is held apart from zero and with the same sign too long, the velocity of the motors will reach there maximum speed and the pendulum will fall over. To avoid this the reference had to be pulsed to keep a constant speed of the vehicle.

When the LQR controller is used there is more freedom in what state to use as reference signal because both the measured states, θ and $\dot{\theta}$, and also the estimated state $\dot{\emptyset}$, are sent back from the Kalman filter. The reference signal used in this controller is the estimated angular velocity of the wheels, $\dot{\emptyset}$. By using the speed of the wheels as reference instead of the pendulum angle there is no need for pulsing the reference to avoid the pendulum from falling.

The turning movement of the pendulum is not done by the PID or the LQR controller. To make the pendulum turn an offset is added to the controller output. When turning right a positive offset is put on the left motor output and an equally large but negative offset to the right motor. On left turns the sign on the offsets is switched. By putting offsets on both the motor outputs instead of just one, the pendulum gets the ability to make very sharp turns.

2.5 Software

2.5.1 Real-time operating systems

In a real-time operating system there are a number of desirable features that may not be included in a regular general purpose operating system. The main features that designate real-time operating systems are [7]:

- *Determinism:* The ability to perform an operation in a strictly defined time period.
- *Responsiveness: T*he ability to quickly respond and handle events when they occur.
- *User control:* Provides the user the possibility to control the priorities of processes executed on the system, and that a program is "*memory resistant*" meaning that there is no possibility to move the program out of the memory to a storage device.
- *Privileged capability: A* process is given the ability to switch to a privileged mode, where interrupts can be enabled/disabled and its own priority can be changed.
- *Reliability:* The system can run continuously for a long time without failing.

2.5.2 Real-time in Linux

There are two main ways to get real-time support in Linux, the co-kernel approach and the fully preemptive approach [8].

In the co-kernel concept a co-kernel runs side-by-side with Linux on the same system. The co-kernel is handling real-time tasks and other tasks are executed in the Linux kernel as usual. To be able to guarantee real-time all interrupts must go through the co-kernel first and then to the Linux kernel [9].

The second approach is based on modifying the standard Linux kernel to guarantee that higher priority kernel processes can preempt low priority processes and get control over the needed resources. To achieve this, the standard device-driver interrupt handler has to be changed [8].

2.5.3 uClinux

uClinux is a small Linux distribution meant for microcontrollers without MMUs (Memory Management Units). This small operating system has a range of applications, libraries and tool chains [10].

2.5.4 Adeos (Adaptive Domain Environment for Operating Systems)

Adeos is used to provide a more flexible environment for sharing hardware resources, which can then be shared with multiple OSes (Operating Systems) running. This is done by letting Adeos take control over some hardware commands coming from the OSes. Adeos handles all interrupts, CPU traps and exceptions by replacing the handlers setup in the Linux kernel with by Adeos own handlers. Adeos introduces an interrupt pipe which is a pipeline where interrupts travel through different OS domains (See Figure 2-14). The domains can be placed in a prioritized order in the pipeline giving the first domain in the pipeline access to the interrupt first. The domain receiving an interrupt can take control over it and chose to accept, ignore, discard or terminate it.

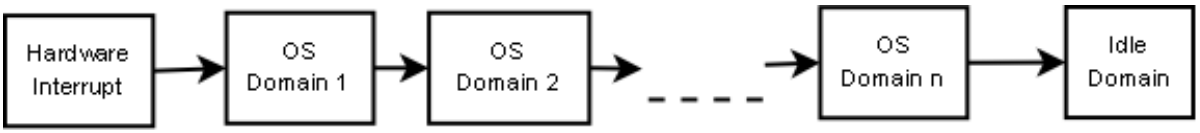


Figure 2-14 Adeos interrupt pipe.

Adoes uses the architecture shown in Figure 2-15 to solve the communication between different domains. Each OS has full control over its domain but OSes that recognize Adeos can, if allowed, interact with resources within other domains.

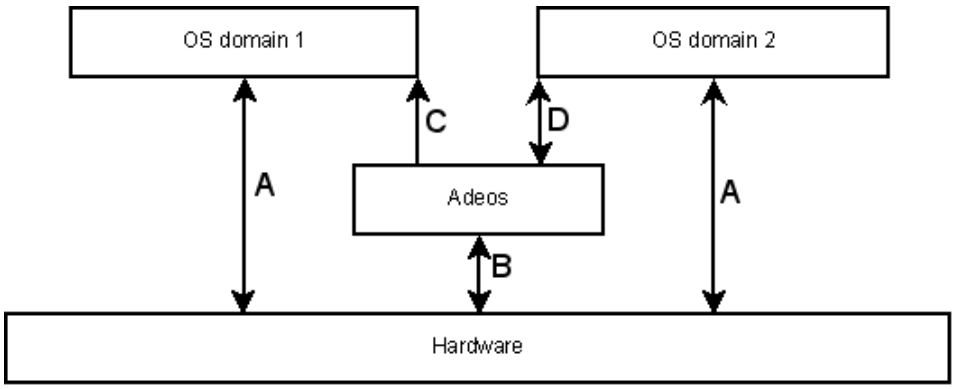


Figure 2-15 Adeos architecture

In Figure 2-15 there are four different ways how the communication between Adeos, hardware and domain OS can be achieved. The first one (A) is working like Adeos does not exist, i.e. the OS domain communicates to the hardware in a regular fashion. In the second mode (B) Adeos controls the hardware and takes care of software and hardware interrupts. (C) is the third mode of communication and is used when a trap uses the interrupt handler to collect information about an interrupt. In the last mode (D) there is a two-way communication between the OS domain and Adeos, which requires the OS domain to be aware of Adeos. In this mode it is possible for an OS domain to ask for full or partial access to hardware that is already allocated by some other domain. It is also possible for a domain to ask for higher priority when it comes to interrupt handling. This mode is where Adeos capabilities are used to its full extent [11, 12].

2.5.5 Xenomai

Xenomai is a small co-kernel which uses the first approach discussed in 2.5.2 to achieve real-time support in a Linux operating system. Xenomai runs side-by-side with the original Linux kernel and takes control over the hardware interrupt management and makes it possible to run real-time tasks aside from the host GNU/Linux system. Multi-threading and basic synchronization are supported and gives several important features when it comes to using threads. The Xenomai API (Application Programming Interface) provides easy to use creation and scheduling of real-time threads. The co-kernel controls the scheduled real-time threads to achieve very low latencies [13, 14].

In order to keep predictable latencies and reliability only use of services within Xenomai is allowed. If any regular Linux command is used like read() or write() one cannot be sure that the scheduled threads gets low latencies. For developing real-time drivers for devices to communicate outside the co-kernel there are a common framework named RTDM (Real-Time Driver Model) [9].

2.5.5.1 Mutual exclusion

When several threads in a program share a common resource problems can arise. For example if two processes decides to write to the same resource at the same time and potentially cause unintended consequence. To solve this problem mutual exclusion has to be provided, forcing threads to access resources one at a time. A *mutex* provides mutual exclusion in Xenomai, the *mutex* also applies priority inheritance which is described in Section 2.5.5.2 [7].

2.5.5.2 Priority Inheritance

When processes of different priority use a common recourse protected by mutual exclusion a problem called priority inversion can occur. For example, if a low priority process has acquired a common resource and a high priority processes tries to acquire the same recourse, in this situation the high priority process has to wait for the low priority one. This is called priority inversion. To avoid priority inversion priority inheritance can be used. Priority inheritance makes a process holding a common recourse inherit priority if a process with higher priority tries to acquire the recourse. Basically the low priority process inherits the priority from the high priority one. Priority inheritance guaranties that a process do not have to wait for a process with lower priority [7].

2.5.6 Fixed-point arithmetic

To reduce the cost of the embedded system many systems do not have a FPU (Floating-Point Unit) to assist when performing floating-point operations. To perform calculations with floating-points there are ways to emulate it in software, but it demands more computational power and may not be suitable if the algorithm is executed at high rate.

In cases where the execution time is of importance it is possible to use a fixed-point representation. This representation is created by dividing a float-point number into one integer part and one fractional part. For example if a 32-bit *int* is used it is possible to use the 16 least significant bits to represent fractions, the next 15 as integer bits and the most significant as sign bit. The range which can be described with 15-bit integer part and one sign bit is $(-2^{15}) \leq x \leq (2^{15} - 1)$ and 16-bit fraction part gives the granularity of $\approx 0.00001526\ (2^{-16})$. This is of course just an example and the number of bits used in the integer and fractional part can be adjusted to suitable resolution and range [15].

Performing operations like addition and subtraction can be done in orderly fashion, but when it comes to multiplication and division it is necessary to extend the word length by a factor two to preserve the correct representation of the fixed-point numbers. For example, if a multiplication is performed with two *ints* (32 bits) having 16-bits fractional parts, a normal multiplication will result in bit shifting the answer up by 16 bits, which can result in shifting out the most significant bits. By first typecasting one of the numbers to an *int* with 64-bits precision and then perform the multiplication, shift down the answer 16-bits and typecasting it back to an *int* with 32 bits a correct representation of the numbers can be preserved. Divisions are carried out in a similar way [15].

2.5.7 Newton Raphson

When computers are used for finding the roots of an algebraic equation some kind of numerical method used often. A widely used method for this is the Newton Raphson method. From an initial starting estimate of the root the Newton Raphson method iteratively tries to find a better estimate. Let r be the true root of an equation $f = 0$. If x_0 is a good estimate of r it is possible to express the

true root like $r = x_0 + h$, where h is a small number and a measure of how far from the true root the estimate is. Because h is small we use a linear approximation and say

$$0 = f(r) = f(x_0 + h) \approx f(x_0) + hf'(x_0) \tag{2.60}$$

If $f'(x_0)$ is not zero then h can be expressed like

$$h = -\frac{f(x_0)}{f'(x_0)} \tag{2.61}$$

A better estimate of the true root r would then be

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{2.62}$$

This in turn, gives the more general expression for the Newton Raphson method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.63}$$

x_n is the current estimate of the root and x_{n+1} is the next, and hopefully a more accurate estimate. Figure 2-16 shows a graphic interpretation of one iteration.

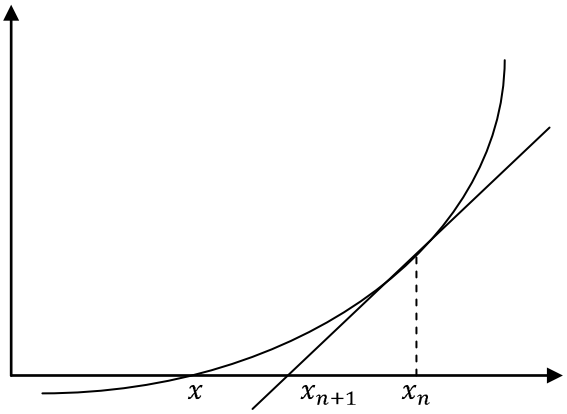


Figure 2-16 Graphic interpretation of one iteration with the Newton Raphson method.

For continues differentiable functions f the Newton Raphson mostly works well, but there are some drawbacks. For example, if the initial estimate of the root is too far away from the true root it may happen that the method converge to the wrong root or diverges towards infinity. However in most cases when the method works well a good estimate of the true root can be found in quite few iterations. The number of correct decimals almost doubles in each iteration. To check if the estimate is close enough to the root, $f(x) = 0$, an easy verification can be done. Take your estimation x_n and decide a desired accuracy, fault limit say 10^{-5}, then check the signs of $f(x_n + 10^{-5})$ and $f(x_n - 10^{-5})$. If the signs are different there must be a root between $x_n + 10^{-5}$ and $x_n - 10^{-5}$ with an absolute value error of at most 10^{-5} [16].

3 Construction

This chapter describes the complete physical system. The system is divided into two major parts, one mechanical and one electrical, but there is also a brief description of the software components included in the system.

3.1 General concept

Figure 3-1 depicts a sketch the pendulum.

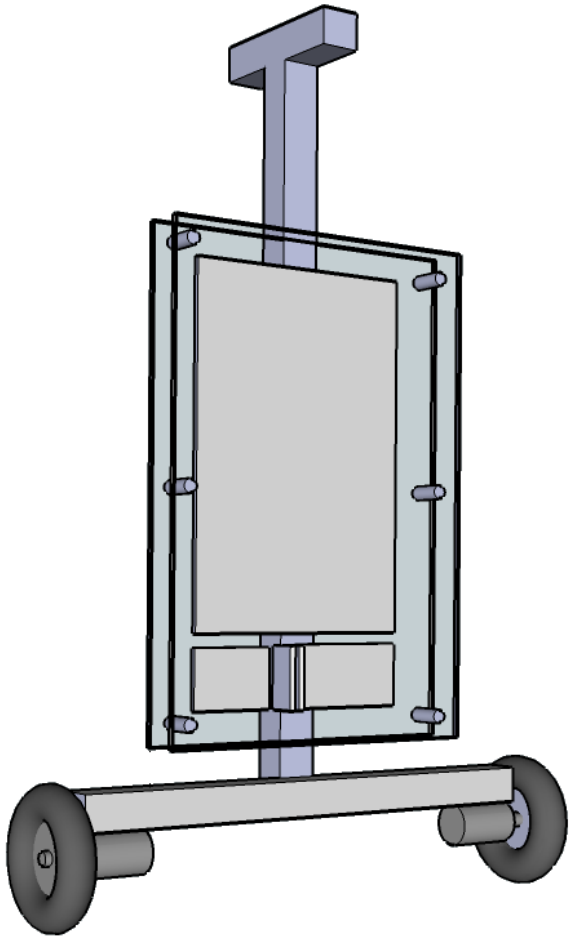


Figure 3-1 General model of the inverted pendulum.

3.2 Mechanical system

This section describes the mechanical parts that are used in the pendulum construction and show how they are put together. Dimensions and other details on the components can be found in Appendix C.

3.2.1 Frame

The frame gives the rough shape of the pendulum and also provide flexible fasten points for other components in the pendulum structure. The weight of the frame is important, because it will affect the dynamics of the pendulum.

The frame material chosen is an aluminum profile, which are light and are easy to mount other components on to.

3.2.2 Wheels and Motors

The motors driving the pendulum are mounted on the frame, and the wheels are placed directly on the outgoing shaft of the geared motor.

3.2.3 Electronic box

To be able to control the robot a number of electronic components are required. These components are placed in a box that is mounted on the frame. The box can be moved around on the frame, giving opportunities to change the dynamics of the robot.

The box is constructed by two plexiglass plates, spacers, washers and screws.

3.3 Electrical system

This section describes the electrical parts used, and how they are placed and connected.

3.3.1 Electronic system

As mentioned in 3.2.3 all electrical components are placed inside a box except the DC-motors. This concept gives a simple solution and makes the electronic parts less vulnerable to strikes of any kind. In Figure 3-2 the placing of components inside the box is shown. The battery is placed on the back of the box and, hence, is not visible in the figure.

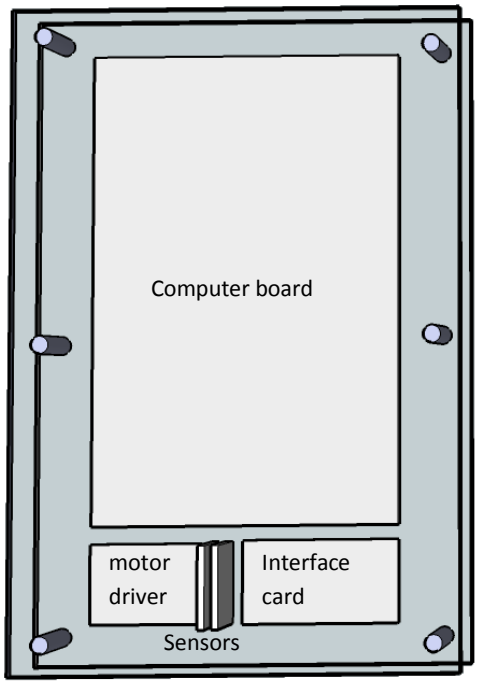


Figure 3-2 Electronic box.

3.3.2 Computer board

The computer board chosen for the control of the system is BF537 STAMP, which is a development platform based on the Blackfin processor.

The BF537 STAMP board has a Blackfin processor with 500MHz core clock, 64Mbytes of SDRAM, 10/100 Mbps Ethernet Interface and several more features. For the board to communicate with external devices several I/O connectors such as RS-232 UART interface, SPI (Serial Peripheral Interface) Bus, SERIAL PORT (SPORT, synchronous serial ports) are available on the board. The board can be configured to have up to 48 GPIOs (General-Purpose Input/Output) which can be used for general I/O usage [17].

The board is setup to run uClinux, together with Adeos and Xenomai. This setup gives the board an open source real-time operating system.

3.3.3 Sensors

To get information about angles and angular rate, different sensors are used. This section describes the sensors chosen to obtain these magnitudes.

3.3.3.1 Accelerometer

An Accelerometer is measuring acceleration, giving a signal or data value that is proportional to the acceleration.

The sensor ADIS16003 from Analog Devices is selected for this project. This sensor is a dual-axis accelerometer, including a temperature sensor, and communicates through SPI. The SPI interface is used to read values from both sensors acceleration axes [18].

The accelerometer is used to measure the angle of the pendulum. When the axis angle changes relative the direction of gravity the accelerometer is affected more or less by the force of gravity. This makes it possible to measure the angles (See Section 2.1.2).

3.3.3.2 Gyroscope

A gyroscope is used for measuring angular rate.

To measure the angular rate the sensor ADIS16080, also from Analog Devices, is chosen. The gyroscope can measure the angular rate around one axis, and has an SPI interface used for writing and reading data to the sensor. The sensor also comes with two additional analog to digital converters (ADC), and one temperature sensor [19].

3.3.4 Interface card

An interface card has been constructed to solve a number of problems. The most fundamental one is that to be able to connect the sensors to the computer board some sort of level translation is needed. The sensors desire 5V on both supply and signal connectors, but the computer board only delivers 3.3V from its signal connector, hence the need for a level translators.

There are also other functionalities built into this card. The main power supply is connected through this card and is distributed to the microcontroller and the motor driver, and there is a main power switch, room to place a Bluetooth module on it and two extra light-emitting diodes that can be controlled from the computer board.

3.3.5 Bluetooth module

The pendulum is equipped with a Bluetooth module for wireless communication. The module which is used is of the type known as “serial cable replacement”. This type of module has built-in Bluetooth stack which makes the connection and use with the computer board easy. As the name implies, the module works as a replacement of a serial cable and, hence, the communication from the program on the computer board is carried out in the same manner. When the device is opened, sending and receiving is done in the same way as write and read to a file or a COM port.

3.3.6 Motors

To be able to balance and move the pendulum two DC motors are used. In order to make a suitable choice of motors the systems dynamics were modeled. The model gave guidance motors could be suitable for drive the pendulum.

The motors selected are regular brushed gear head DC-motors. This means that a gearbox is mounted directly on the DC-motor shaft. The gearbox has a gear ratio of 30:1 which reduces the outgoing shaft to a maximum speed of 200 rpm. The DC-motors are 12V motors and with gearbox they have a stall torque of 0.49 Nm.

3.3.7 Motor driver

To be able to control the speed and direction of the motors from the computer board amplifiers has to be used. For this purpose a special motor driver board is used. The motor driver is a small circuit board with its own microprocessor. It can separately and simultaneous control two motors in different speeds and direction. The board is fed with 11.1V from the interface card for power supply to the motors and the internal circuits. The communication between the driver and the computer board is serial and the motors are controlled by sending an 8 bit character to the driver. Each motor can be controlled by 7 bits resolution. By sending a value between 1 and 127 the first motor is controlled. 1 is full reverse, 64 is stop and 127 correspond to full speed forward. The second motor is controlled in the same way, but by sending a number between 128 and 255, where 128 is full reverse, 192 stop and 255 is full speed forward. One of the serial ports on the computer board is used for this communication.

3.3.8 Power supply

To power the electronic system a Li-Po (Lithium Polymer) battery is used. The reason for choosing this type of battery is the high ratio between energy and weight. The Li-Po used as power supply has three cells (11.1V) which delivers 1500mAh and weigh 118 grams.

3.3.9 Communication

Figure 3-3 shows an overview of how the electronic parts on the pendulum are connected to each other and which communication interfaces respectively component uses.

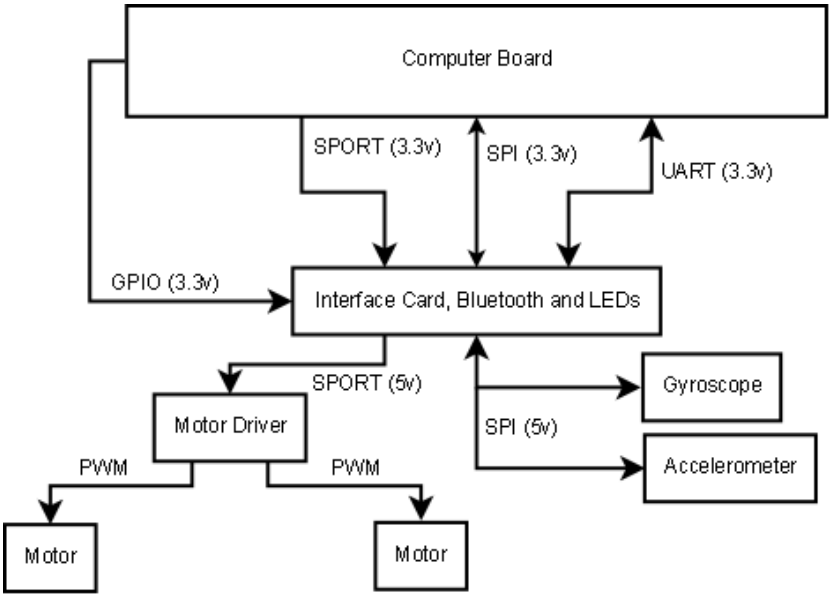


Figure 3-3 Overview of the communication between the electrical components.

3.4 Software

The software controlling the pendulum is developed in regular C, and because the computer board runs Linux the software is basically developed like any other Linux software. One feature added in this case is the real-time support given by kernel patches from Adeos and Xenomai. Xenomai provides a real-time framework which is used for scheduling high priority threads and mutual exclusion.

To give the software a good structure it is divided into a number of separate files, each with a specific purpose. These files and their purpose will be shortly described in the next section.

3.4.1 The control software structure

In Figure 3-4 the structure of the program is illustrated, the arrows indicating the direction of the communication between the files. Each rectangular box in Figure 3-4 consists of one source file and one header file, but for simplicity they are illustrated as one file. The parallelograms are character devices.

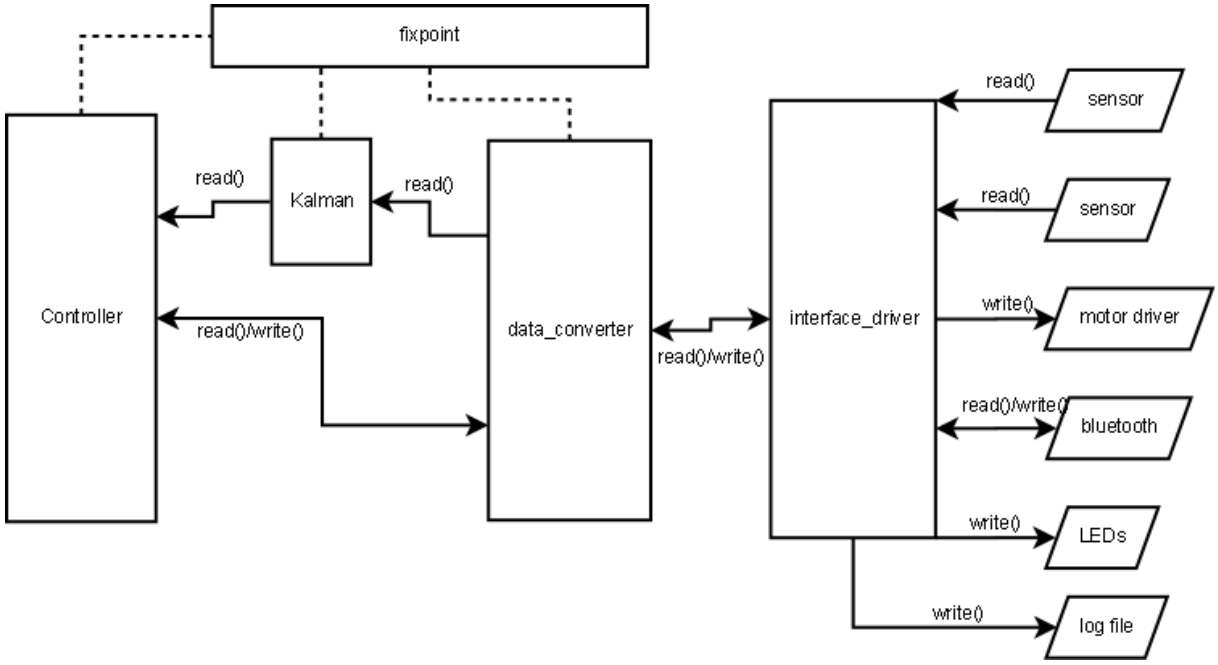


Figure 3-4 Program structure

The different files are described below.

- *interface_driver:* The main purpose of this file is to handle all forms of communication to or from devices, such as the SPI devices, the motor driver device or the Bluetooth device. This file is only included in the data_converter file which means that all communication comes direct or indirect from this file. Interface_driver uses regular Linux commands for read from and write to the character devices.
- *data_converter:* This file is used to communicate with the interface_driver and in most cases the type of the data sent through data_converter is converted. For instance, the raw data read from the gyroscope has to be converted before it is used in calculations in the controller.

- *fixpoint:* The fixpoint file defines a type which is used to implement fixed-point arithmetic. It also includes functions used for division and multiplication of two variables of this type. The type is called “FIXED16_16” and is used in the majority of the calculations in the program.
- *kalman:* In the kalman file the kalman filter and the state observer is implemented, it basically consists only of one function that calculates and returns a state estimation given the output to the motor and the latest sensor values.
- *controller:* The controller handles both the control loop and the Bluetooth communication but in two separate threads. The controller file uses the three files fixpoint, kalman and data_converter.

3.4.2 Threads

As mentioned earlier, the program consists of a number of threads. These threads are prioritized threads spawned with help from Xenomai. The different priorities are given to the threads. For instance, the control loop thread which calculates new output data is given high priority and the communication thread is given a low priority.

These two loops share common recourses, one example of a common recourse is the reference signal. The control loop reads this signal every iteration and the communication loop has the ability to change this signal. This implies that the reference signal has to be protected by mutual exclusion making it impossible for both threads to access it at the same time. Before any reading or writing is done to the reference signal the *mutex* protecting it has to be acquired. Now, because the *mutex* supports priority inheritance the communication loop can inherit the higher priority from the control loop.

3.4.3 Control loop

The control loop is the most essential part of the program. It reads sensor data and calculates new output signals to the motors to keep the pendulum in an upright position. In Figure 3-5 a basic flow chart of the control loop can be seen.

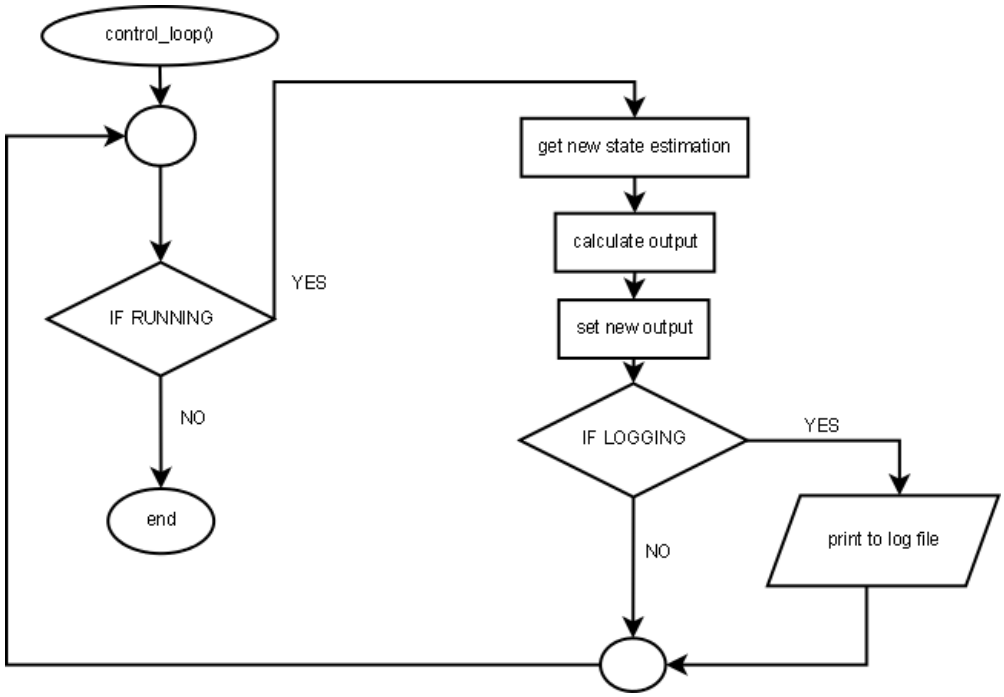


Figure 3-5 Control loop flow chart.

The loop consists of four major operations described below.

- *get new state estimation:* This operation includes reading a new sampled value and process it through the kalman filter and the state observer.
- *calculate output:* When a new state estimate is received a new output signal from the controller is calculated in this operation.
- *Set new output:* Once the new output is calculated it is sent out to the motor driver through the data_converter.
- *Print to log file:* If logging is enabled the control loop writes information about the estimated states, output values and reference value to the log file.

3.4.4 Remote control

To control the mobile inverted pendulum a client is developed to function as a remote control to the pendulum. This client is of simple character providing functionalities like start/stop, exit, drive forward/backward and turn. The platform chosen for the client is a regular mobile phone with a touch screen, the client is developed in Java Micro edition and uses Java™ APIs for Bluetooth (JSR 82), for wireless communication. Figure 3-6 shows a screenshot from the mobile client.

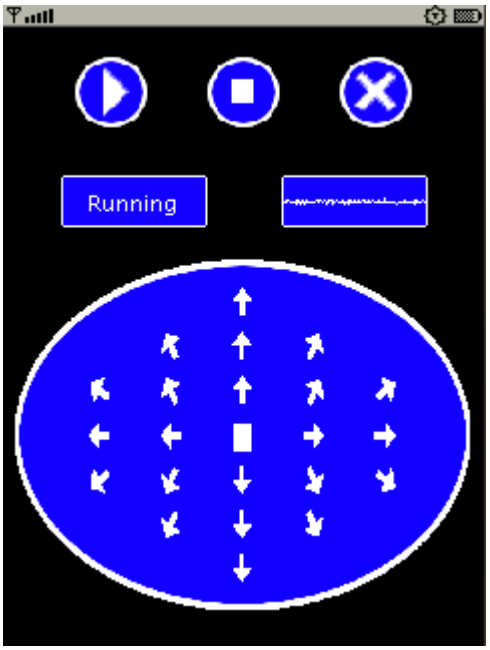


Figure 3-6 Screenshot from mobile client.

4 Result and discussion

4.1 Mechanical and electrical construction

The actual hardware components of the pendulum were presented in Chapter 3. The result of the construction work is shown in Figure 4-1.

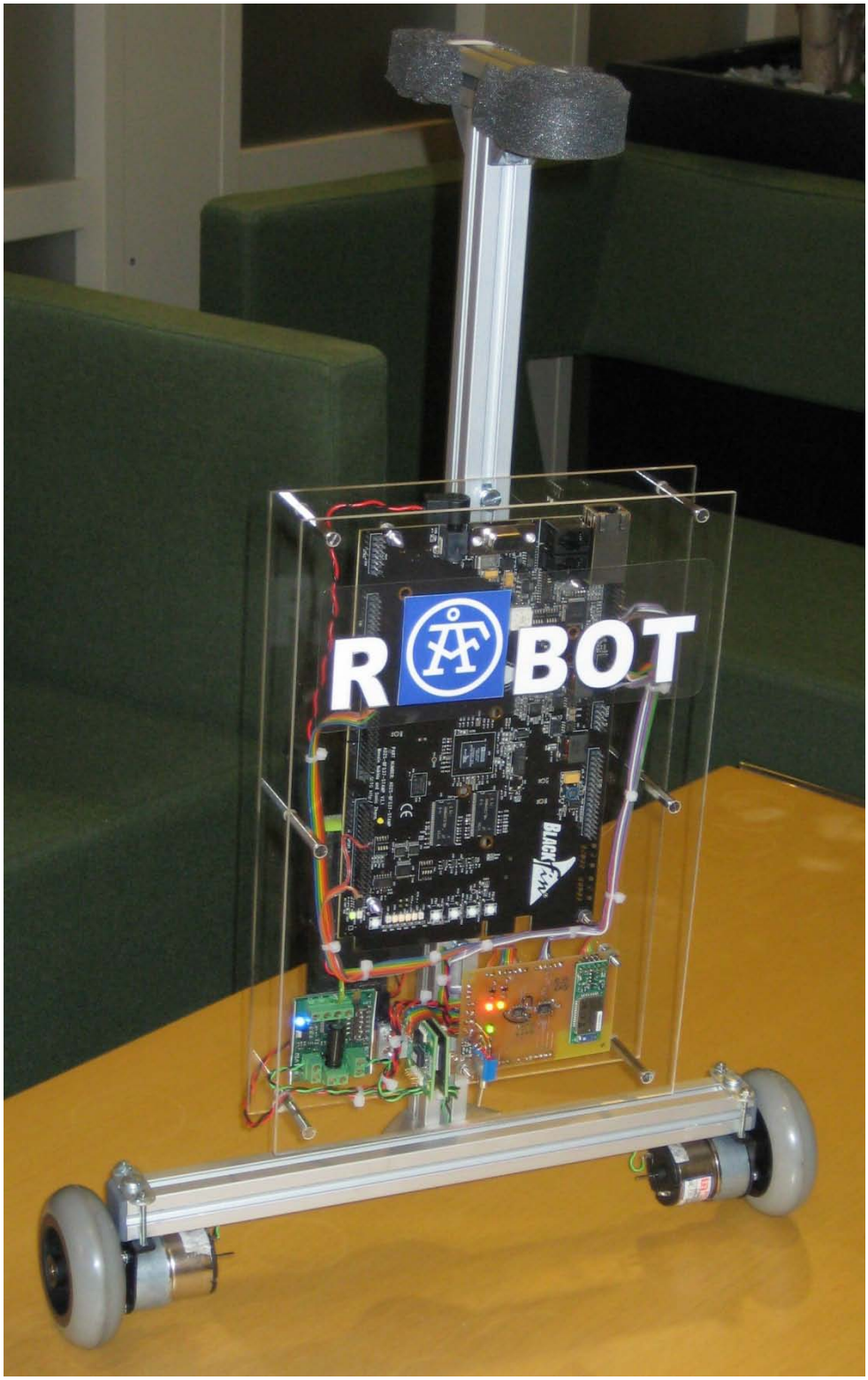


Figure 4-1 Picture of the pendulum

4.2 Control system

This section discusses the results and evaluates the parts which have to do with the control system of the pendulum.

4.2.1 Control evaluation

Two different types of controllers have been used to control the pendulum, PID and LQR. When it comes to balancing the pendulum both types of controllers work and give satisfying performance after some tuning. The controller constants for both the PID and the LQR controller used on the real system had to be a little bit larger compared to the simulations to be able to balance the pendulum with the same performance.

Keeping the pendulum in an upright position is one desired property, another is that the pendulum should not move around in the room too much just to be in a upright position. The PID controller only controls the pendulum given the angle θ and does not care about the angular velocity of the wheels, ($\dot{\phi}$). The LQR controller uses three states (θ, $\dot{\theta}$ and $\dot{\phi}$) making it possible to control the angular velocity of the wheels. One would therefore suspect that the LQR controller should be the best one to fulfill the desire to keep the wheel speed close to zero given its ability to also control the state $\dot{\phi}$. As it turns out, both controllers had problems with the angular velocity of the wheels. The problem occurred due to calibration errors of the measured pendulum angle. For instance the controller and the state observer assumes that the angle θ is equal to zero when the pendulum is in an upright position, and this makes it crucial for the measured pendulum angle to be zero when the pendulum is in an upright position. This means that if the measured angle gives a value of one degree when the pendulum is in the upright position and θ should be equal to zero, the controllers tries to compensate to make the measured angle equal to zero. This will result in an angular velocity on the wheels. In the case of the PID controller the velocity increases until it reaches the maximum and the pendulum falls, but in the case of the LQR controller the introduced velocity gets constant. To solve this problem the measured angle is calibrated to zero when the pendulum is in upright position. This approach was tried and removed the unwanted angular velocity of the wheels. However, this solution is not a particularly robust. If a small change was made to the pendulum, changing the upright positions zero angle, the measured angle had to be recalibrated. To avoid the recalibrations a self calibrating algorithm was introduced. The algorithm uses the output to the motors to adjust the measured angle. The idée behind the algorithm is that if the pendulum is in an upright position the output to the motors should be equal to zero. If it is not, the algorithm adjusts the measured angle. This approach solved the problem with unwanted angular velocity on the wheels and also gave some extra robustness to the system, making it possible to add small weights to the pendulum without reconfiguring system parameters.

Once the unwanted angular velocity on the wheels was removed another problem arose. The pendulum started to oscillate back and forth at a constant frequency. These oscillations were studied and found to be caused by a dead-zone in the DC-motors. By performing experiments on the DC-motors a dead zone of approximately ± 0.8V was found, which the state observer did not take into account. Outputs to the DC-motors smaller then ± 0.8V therefore affected the state observer but not the real system. The oscillations could also be seen in the Simulink model by adding the dead-zone on the input to the system but not to the state-observer. To solve the problem the state observer was changed to set output values equal to zero if they were smaller then ± 0.8V, and remove 0.8V if

larger than and add 0.8V if smaller than -0.8V. When these changes to the state observer were implemented the oscillations disappeared.

When it comes to movement control, the concepts described in Section 2.4 were implemented and the turning movement worked very well for both controllers. The different concepts for backwards and forwards movement worked for both PID and LQR control, but in the LQR case the movement control works far better. In the PID case the speed is hard to control. One could say that it is only possible to control the direction of the movement in the PID case.

4.2.2 Sensor modeling

To measure the angle of the pendulum two alternative methods where evaluated: simple arcus sinus method, and the more complicated Newton-Raphson method. To illustrate the performance of the methods three different evaluations were preformed.

In the first evaluation case, the accelerometer is tilted back and forth without giving the sensor any linear acceleration. Both methods calculate an angle for each sampled value from the sensor to more easily compare them. In Figure 4-2 it is seen that both methods work by returns reasonable angle values, but the Newton-Raphson method returns a more distorted signal.

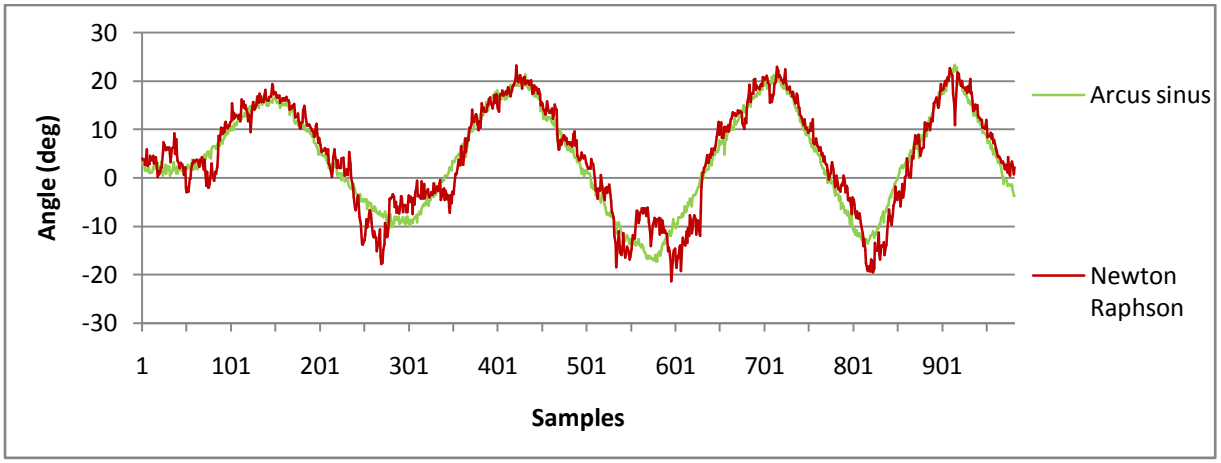


Figure 4-2 First case, Arcus sinus versus Newton Raphson

In the next evaluation case the accelerometer is moved back and forth without tilting it, this will only introduce linear acceleration. In this case the idea is to show how the Newton-Raphson method will give an angle close to zero and that the arcus sinus method will be highly affected by the linear acceleration.

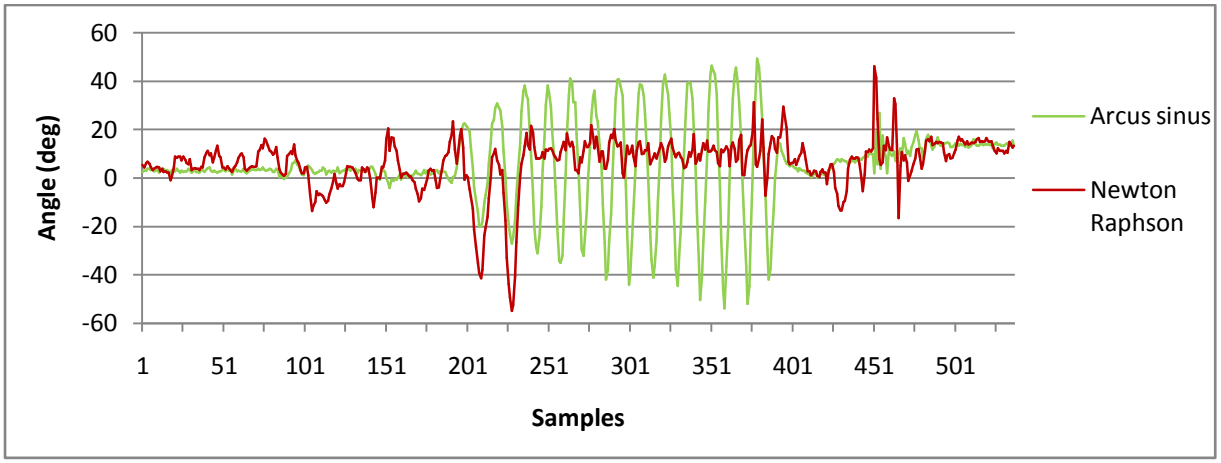


Figure 4-3 Second case, Arcus sinus versus Newton Raphson

According to Figure 4-3 the Newton Raphson method is able to compensate for the linear acceleration to large extent but not completely. Looking at values around sample 200 to 250 it is possible to see that sometimes the Newton Raphson method also returns a value affected by the linear acceleration.

The last evaluation case is a comparison of the two methods when driving the pendulum back and forth, making the pendulum angle to change like a sinus.

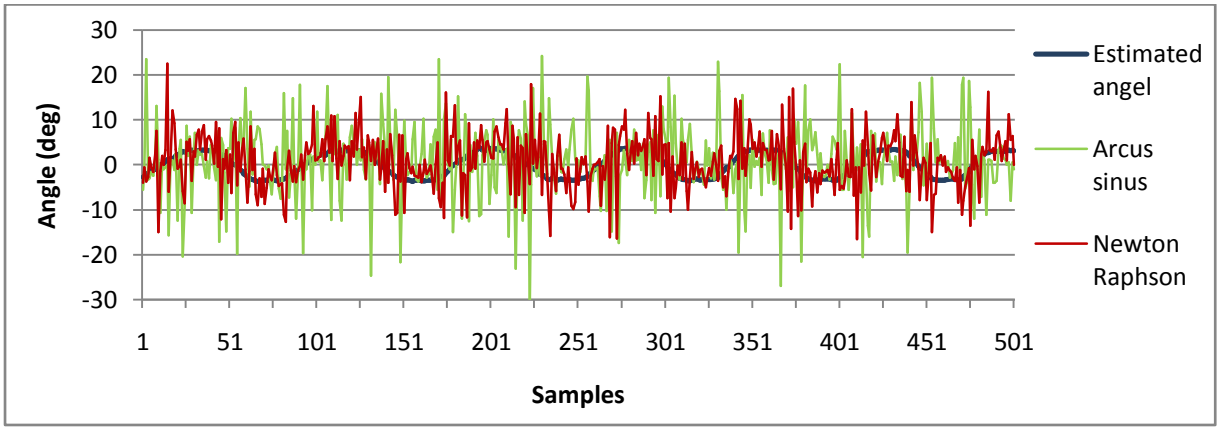


Figure 4-4 Driving performance

Figure 4-4 shows that the Newton Raphson method returns a slightly less distorted values than the arcus sinus method. The correlation between the signals gives a rather mixed result, the correlation of the estimated angle and Newton Raphson is 0.4338 and between the estimated angle and arcus sinus the correlations is -0.159. The negative correlation in the case of arcus sinus is somewhat strange, but if taking a look at Figure 4-2 the correlation is clearly positive in the case without linear acceleration.

4.2.3 Kalman filter performance

When the Kalman filter was implemented on the real system the gains calculated during the simulation had to be modified a bit to get a good performance. The filter designed during the simulation gave too noisy angle estimates. The largest problem was that the angle measurement from the accelerometer was noisier on the real system than in the Matlab simulation. This is because the influence of the pendulums linear movement on the accelerometer is not considered in the

Simulink model. The numerical calculation by the Newton Rhapson method described earlier does not completely compensate for this effect. Therefore some additional noise is introduced to the angular measurement. The Kalman filter had to be modified to trust the model more. Covariance matrices which give a good balance between reducing the noise and do not lag the estimation too much on the real system are

$$R_1 = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_2 = [100] \tag{4.1}$$

These matrices give a larger ratio, R_1/R_2, compared to the matrices used during the simulation, which in turn means that we get smaller Kalman gains and hence put more trust on the model in the estimation.

Figure 4-5 shows the Kalman filter performance on the real system.

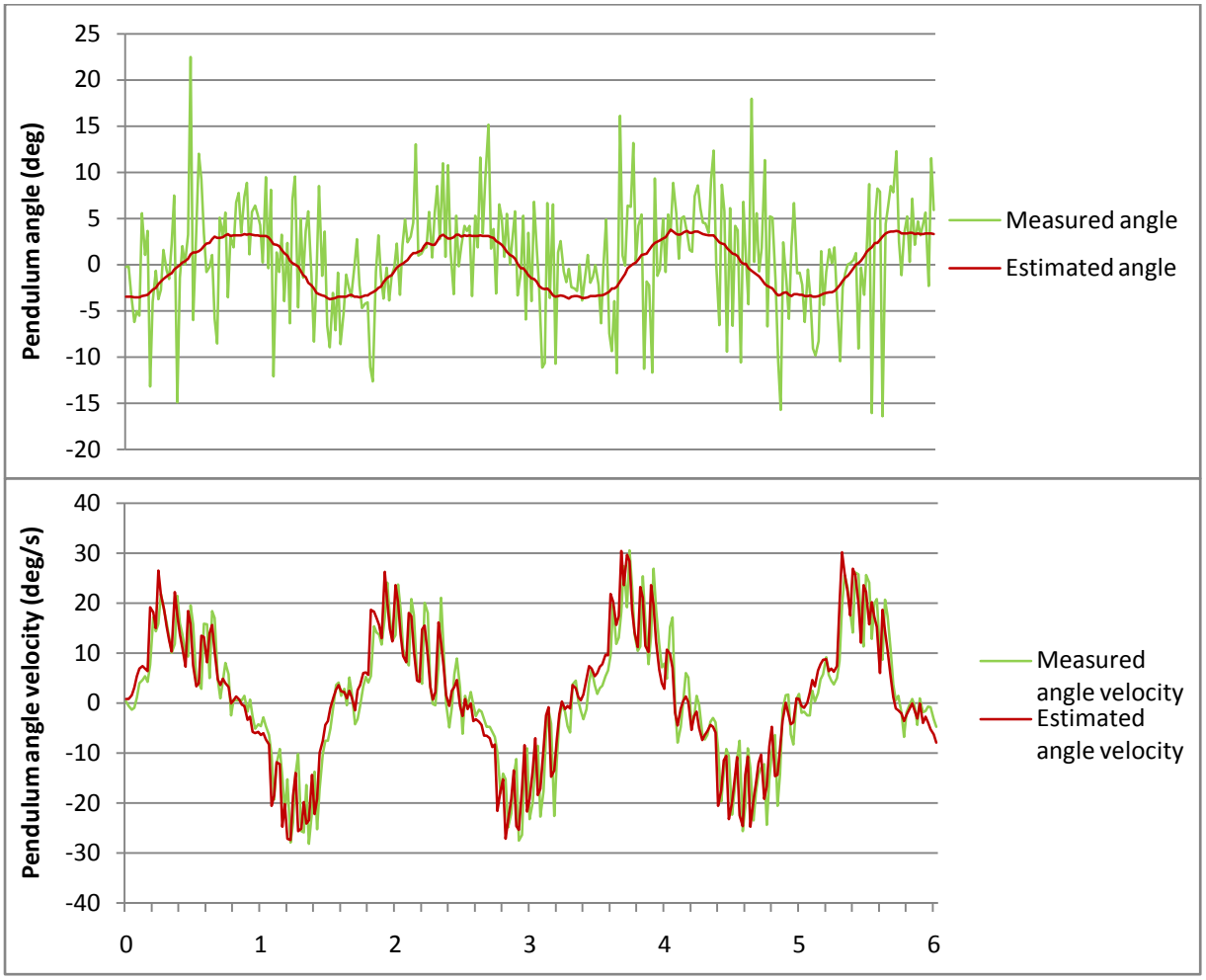


Figure 4-5 Kalman performance on the real system

One can see that the suppression of the angle measurement is very good and at the same time the estimation is not lagged. The measurement from the gyroscope is not as much damped but the estimation is good enough.

During the drive where the data to Figure 4-5 is taken the reference signal for the wheels angular velocity was manually set forward and backward to get the swinging behavior of the pendulum. The velocity of the wheels is the estimated state in the state observer. Figure 4-6 depicts the reference signal and the estimated state on the wheels rotational velocity.

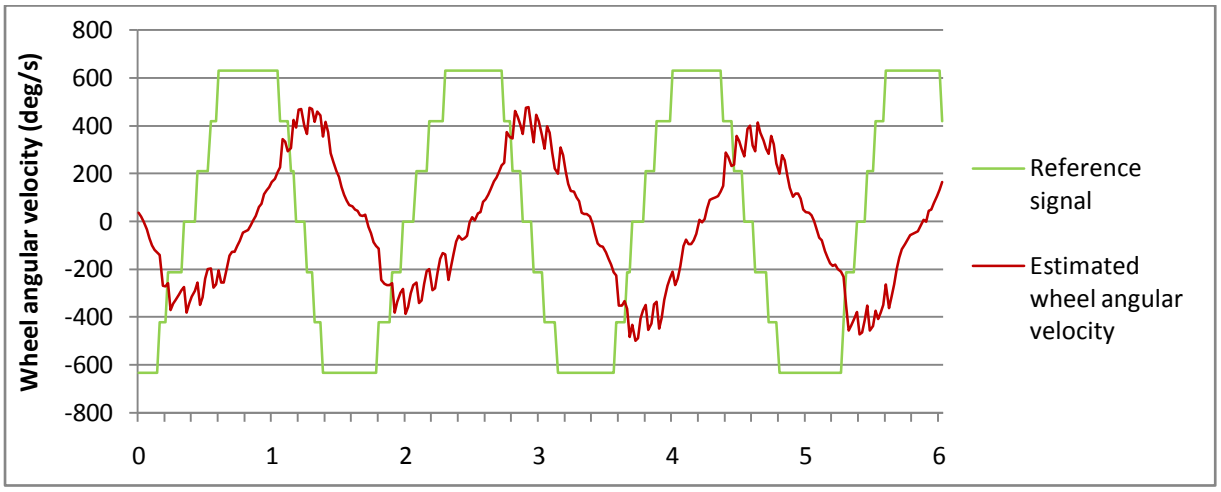


Figure 4-6 Reference signal following on the real system

Because of the relatively fast reference signal changes the estimated state have some problem to reach the peaks in the reference. However, this reference signal following is sufficient for this case, where the pendulum angular state is the most important.

4.2.4 Comparison model versus real system

Since the model of the system is used to design the Kalman filter, the state observer and the control parameters it is important that the model behavior agree with the real system. By comparing data from simulations of the model and the real system it is possible to see if they react in similar ways to reference changes. The data collected for this evaluation is gathered by logging the real system when the reference value is changed, and then the exact same reference changes is used in the simulink model. Figure 2-7 and 2-8 illustrate the behavior of the model states and the estimated states from the real system. The same control parameters and Kalman gain were used.

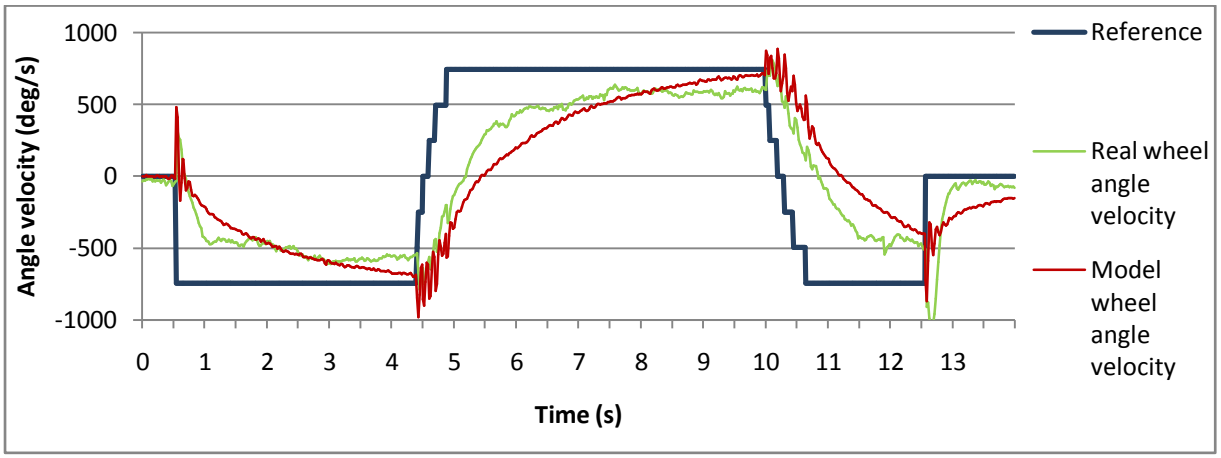


Figure 4-7 Wheel angular velocity: A comparison between simulated and measured closed loop response.

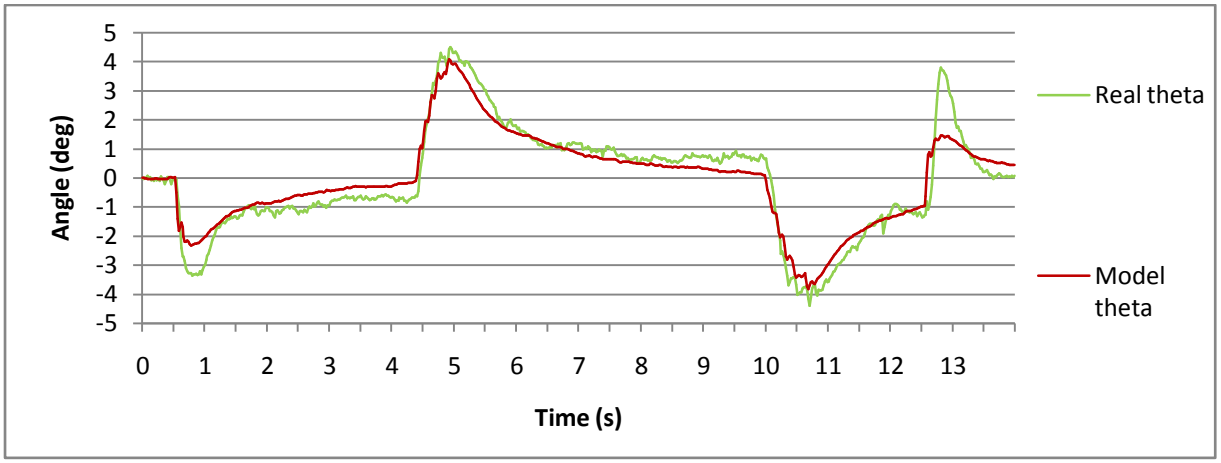


Figure 4-8 Pendulum angle.

In Figure 4-7 the angular velocity of the wheel is shown and in Figure 4-8 the angle of the pendulum is shown. The responses of the model and the real system to the reference changes are similar, this indicating that the model and the real system agree.

4.3 Linux real-time performance

To measure the performance of the real-time operating system consisting of Adeos, Xenomai and uClinux the execution time of the periodic control loop is measured. The control loop is attended to execute every 20 milliseconds, and within every execution a number of calculations are performed. The control loop is running as a high priority real-time thread, and the calculations performed in the loop should take the same time if the system is under a high workload or not. The calculation time and the time the loop sleeps between executions are logged during high and low workload.

In Figure 4-9 it is easy to notice when high workload is applied. The calculation time is highly affected by the increase of workload. When the calculation time increases the sleep time naturally decreases. Hence, the loop is to execute in a steady phase of 50 Hz (20 milliseconds). The dips in the calculation time that occurs can be explained by the behavior of the Newton Raphson algorithm. They occur when the Newton Raphson method find a solution within few iterations. In Figure 4-10 the same type of workload is applied but the control loop does not handle the task of writing the output value to the motors.

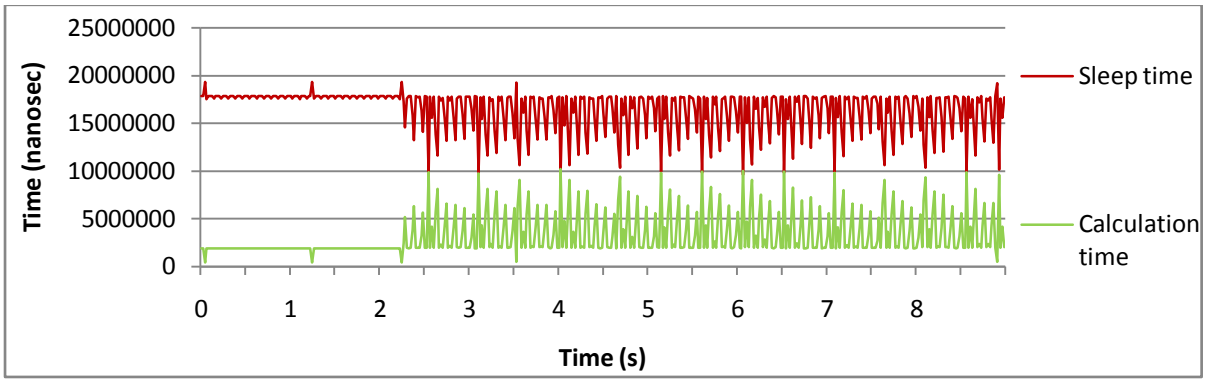


Figure 4-9 Sleep time, calculation time.

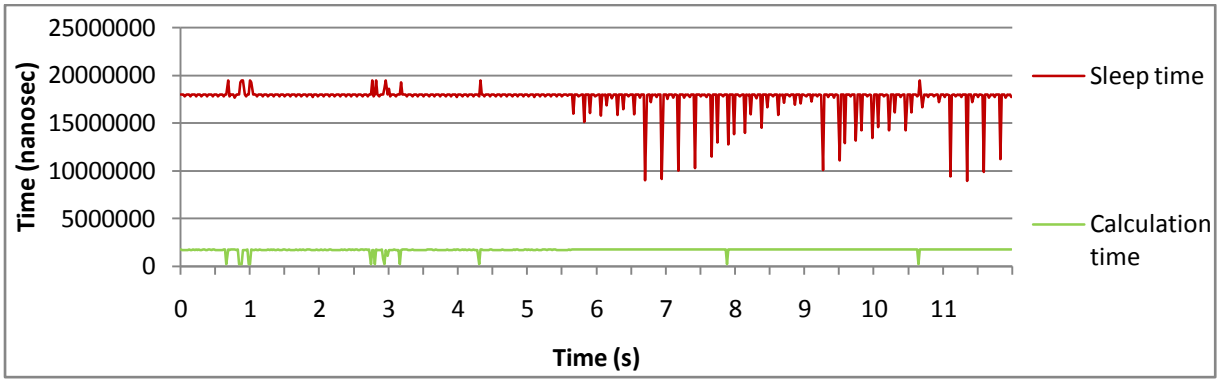


Figure 4-10 Sleep time, calculation time.

The calculation time is more or less unaffected by the workload, but it is possible to see that the sleep time is affected. The sleep time is not only dependent on the calculation time. It also depends on the time it takes to write information to the log file, which explains the decreases in sleep time.

Figure 4-11 shows the same calculation time as in Figure 4-10, but with another scale on the y-axis. When the workload is applied a small increase in calculation time can be seen.

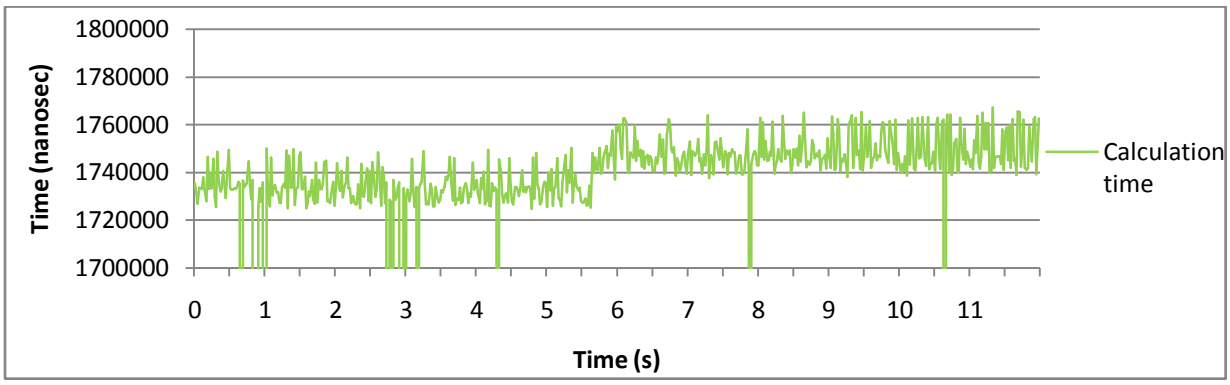


Figure 4-11 Calculation time

The results of the performance test is somewhat mixed. Figure 4-9 shows that the control loop is highly affected by low priority workload, which is bad. This implies that the control algorithms can be delayed by high workload and change the overall performance of the control system. On the other hand the results shown in Figure 4-10 and Figure 4-11 implies that it is the writing of the output to the motors which affects the workload and not the time it takes to perform the calculations. This shows that the real-time operating system manages to prioritize the high priority when high workload is applied, and it also reveals that using standard Linux drivers can jeopardize the real-time performance.

5 Conclusion

5.1 Goal fulfillment

The absolutely main goal to get the pendulum balancing by itself has been very well fulfilled. The pendulum is able to balancing and can compensate for quite hard nudges without falling. Movement control was also successfully implemented. The pendulum can move forward and backward and make sharp turns with no risk for instability or losing control. The last goal, i.e. the wireless connection and the ability to control the pendulum from a mobile phone or a computer was also fulfilled. In the start this goal was a little bit unsure because the time required to get the basic functionality to work was unknown. The time was enough and a well working mobile controller has been developed. The ability to control the pendulum over a wireless connection really increases the product quality and usefulness.

5.2 Control and filtering

Two control strategies were tested and evaluated on the real system, PID control and LQR control. The LQR controller where chosen. Though not because the balancing properties were better the two controllers worked with about the same performance in that aspect but because the LQR controller is a more simple solution when it comes to movement control. When using the LQR controller the reference signal was put on the wheels angular velocity. This appears to work better than to put the reference on the pendulum angle, which was the case when using the PID control.

Without the Kalman filter there it would probably have been very hard to get a good balancing performance of the pendulum because of noisy angular measurements. The Kalman filter also performed well for state observation of the non-measured state. The function of the state observation indicates that the mathematical model of the system is sufficiently accurate.

The linear acceleration of the pendulum has been shown to have large influence on the accelerometer measurements. The sensor model which takes advantages of both the accelerometers axes did not work as good as expected, but is still the one used in the final solution. It performs slightly better than the method which uses only measurements from one axis of the sensor. However, the difference between the two was expected to be more significant. Maybe the sensor model can be improved and if more time is put into tuning and optimization of the implementation of Newton Raphson method which is used to solve the equation in the model.

5.3 Hardware

As stated in the introduction not much effort has been put into make the mechanical construction optimal. The overall component choice has been based on functionality and finding parts which can be put together in an quite easy way.

The choice of aluminum profiles for the pendulum frame was a good choice which gave a low weight, easy and robust construction.

When it comes to the electrical components the functionality and the communication between the different components has worked as intended. Some things can however be pointed out. The evaluation board with the Blackfin processor is quite large. However, the Blackfin board has been beneficial in many ways not at least with its diligent used internet forum where answers to a lot of questions can be found. Another thing which should have decreased the complexity of the

construction would have been to find sensors and motor driver with the same voltage level on the signals as the computer boards. The extra interface card could then have been constructed without level translators.

The two brushed DC motors have no problem balancing the pendulum but in retrospect the choice of motors should have been a pair with higher top speed. That would allowed driving the pendulum at higher speed.

5.4 Open source real-time OS

The open source OS uClinux has been working better than expected. It was easy to get into and get it started. There were a lot of documentations and information on the internet about the uClinux distribution, which came with the computer board. This has of course been very useful. Also the real-time support trough Xenomai has been working very well when avoiding the usage of non real-time drivers. From the beginning, much trouble was expected with uClinux. Naturally, there have been some troubles on the way and maybe an open source OS contain some bugs which one would probably not find in a commercial OS.

5.5 Future work

Even though the control system works well, it would be desirable to develop real-time drivers. By developing real-time drivers for performing input/output communication, problems that occur when high workload is applied can be avoided.

Some more time can be put into optimizing the use of the Newton Rhapson method for finding the root of the equation in the sensor model.

The project can be further extended in many ways. One idea would be to equip the pendulum with WLAN(Wireless Local Area Network) and a camera. The picture from the camera could then been streamed to a computer. The range of control could be increased and it would be possible to control the pendulum from another spot. One can imagine implementing some image processing. Maybe let the pendulum follow a specific color or a specific object shape.

6 Bibliography

1. **Björn Wittenmark, Karl Johan Åström, Karl-Erik Årzén.** *Computer Control: An Overview.* Lund, Sweden : Department of Automatic Control, Lund Institute of Technology.

2. **Segway Inc.** http://www.segway.com. [Online] [Citat: den 29 04 2009.] http://www.segway.com.

3. **Ragnar Grahn, Per-Åke Jansson.** *Mekanik, Statik och dynamik.* Lund, Sweden : Studentlitteratur, 2002.

4. **Karl Åström, Richard M. Murray.** *Feedback Systems.* Lund : Department of Automatic Control, Lund Univercity, Control and Dynamical Systems, California Institute of Technology, 2007.

5. **Lennartson, Bengt.** *Reglerteknikensgrunder.* Lund : Studentlitteratur, 2002.

6. **Karl J. Åström, Björn Wittenmark.** *Computer Controlled Systems.* United States of America : Prentice Hall, Inc, 1997. 0-13-314899-8.

7. **Karl-Erik Årzen, Jad El-Khoury, Jianlin Shi, Martin Törngren.** *Real-Time Control Systems.* Göteborg : Department of Automatic Control Lund Institute of Technology, Department of Machine Design The Royal Institute of Technology, 2008.

8. **Aeolean Inc.** http://www.aeolean.com. [Online] den 11 12 2002. [Citat: den 29 4 2009.] http://www.aeolean.com/html/RealTimeLinux/RealTimeLinuxReport-2.0.0.pdf.

9. **Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, Philippe Gerum.** *Building Embedded Linux Systems.* United States of America : O'Reilly Media, Inc, 2008. 978-0-596-52968-0.

10. **uClinux.** uClinux. *uClinux Embedded Linux/Microcontroller Project.* [Online] den 1 1 2009. [Citat: den 9 2 2009.] http://www.uclinux.org/.

11. **Yaghmour, Karim.** Adaptive Domain Environment for Operating Systems. *Opersys inc.* [Online] den 1 4 2004. [Citat: den 9 2 2009.] http://opersys.com/ftp/pub/Adeos/adeos.pdf.

12. **Gerum, Philippe.** Porting Adeos (beneath the Linux kernel). [Online] den 19 8 2003. [Citat: den 9 2 2009.] http://download.gna.org/adeos/doc/porting.txt.

13. **Xenomai.** Xenomai:About. *Xenomai.* [Online] Xenomai, den 9 10 2006. [Citat: den 9 2 2009.] http://www.xenomai.org/index.php/Xenomai:About.

14. **Gerum, Philippe.** Xenomai - Implementing a RTOS emulation framework on GNU/Linux. *Xenomai.* [Online] den 1 4 2004. [Citat: den 9 2 2009.] http://www.xenomai.org/documentation/branches/v2.4.x/pdf/xenomai.pdf.

15. **Oberstar, Erick L.** Fixed-Point Representation & Fractional Math. *superkits.net/.* [Online] den 30 9 2007. [Citat: den 17 3 2009.] http://www.superkits.net/whitepapers/Fixed%20Point%20Representation%20&%20Fractional%20Math.pdf.

16. **Anstee, Richard.** University of british columbia, Mathematics department. [Online] September 2006. [Citat: den 18 March 2009.] http://www.math.ubc.ca/~anstee/math104/104newtonmethod.pdf.

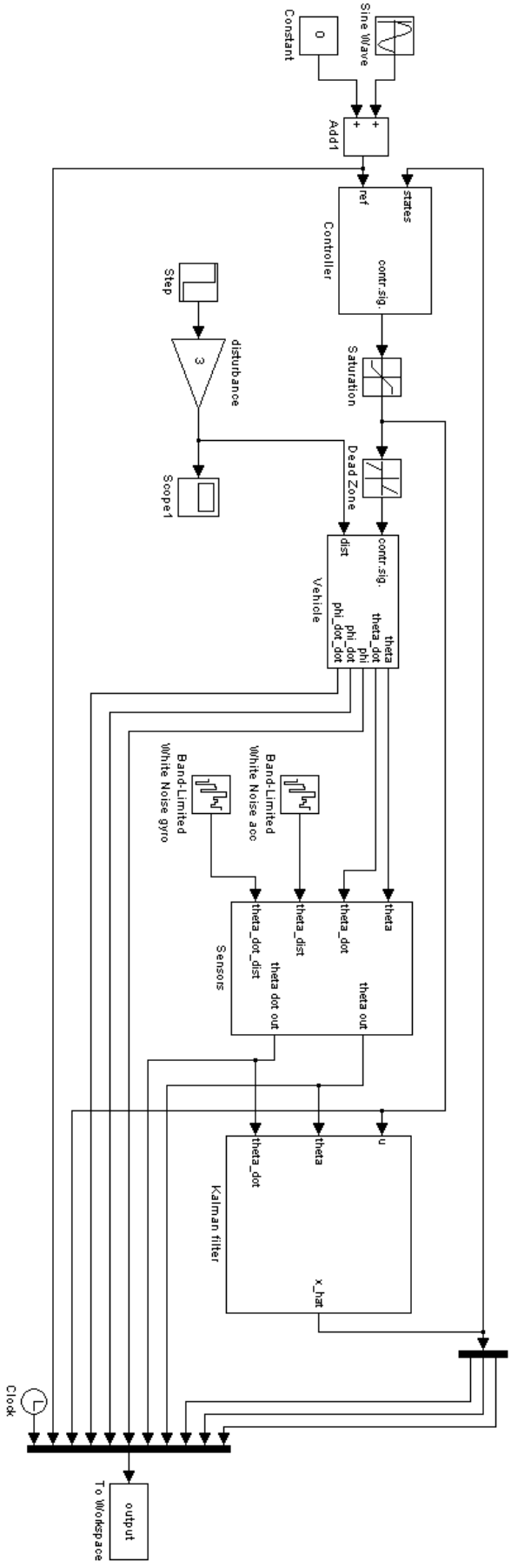
17. **vapier.** Blackfin Linux Docs. *BF537 STAMP Board.* [Online] den 18 01 2009. [Citat: den 8 2 2009.] http://docs.blackfin.uclinux.org/doku.php?id=hw:boards:bf537-stamp.

18. **Analog Devices.** Dual-Axis ±1.7 g Accelerometer with SPI Interface. *Analog Devices.* [Online] den 1 10 2007. [Citat: den 9 2 2009.] http://www.analog.com/static/imported-files/data_sheets/ADIS16003.pdf.

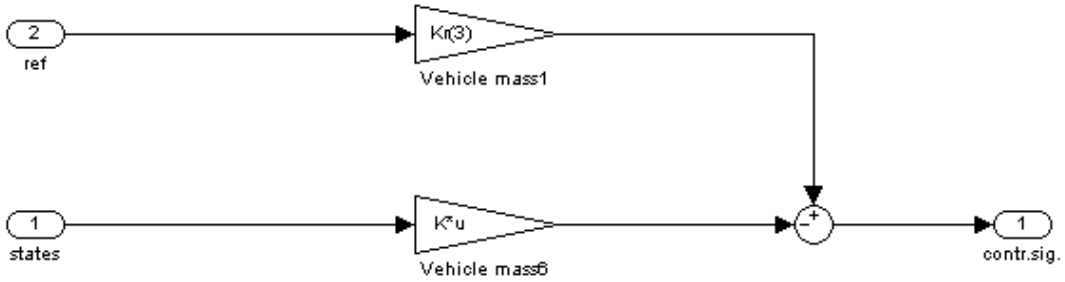
19. **Analog devices.** ±80°/sec Yaw Rate Gyroscope with SPI. *Analog devices.* [Online] den 1 4 2007. [Citat: den 9 2 2009.] http://www.analog.com/static/imported-files/data_sheets/ADIS16080.pdf.

Appendix A

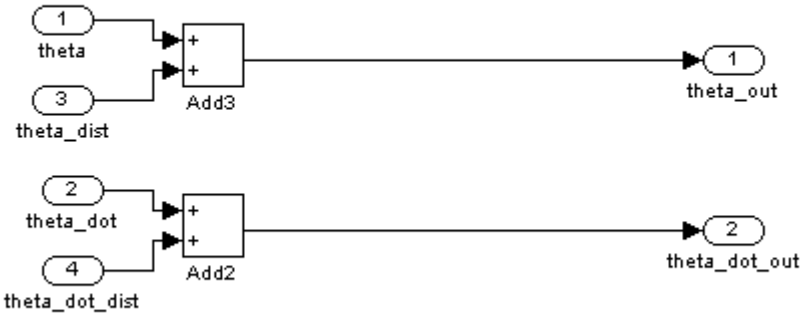
Main model



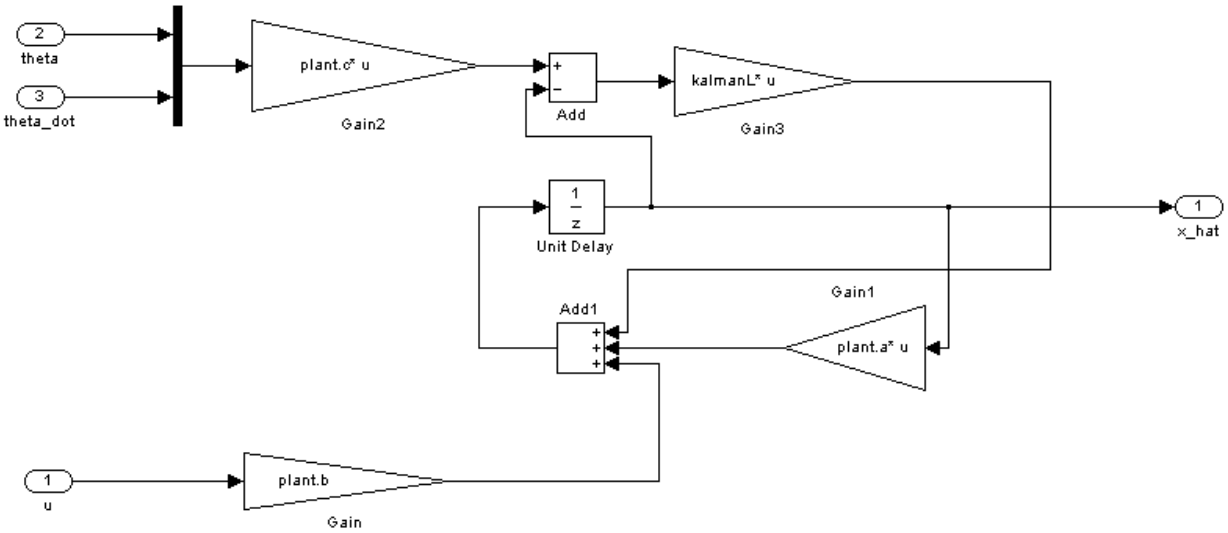
Controller



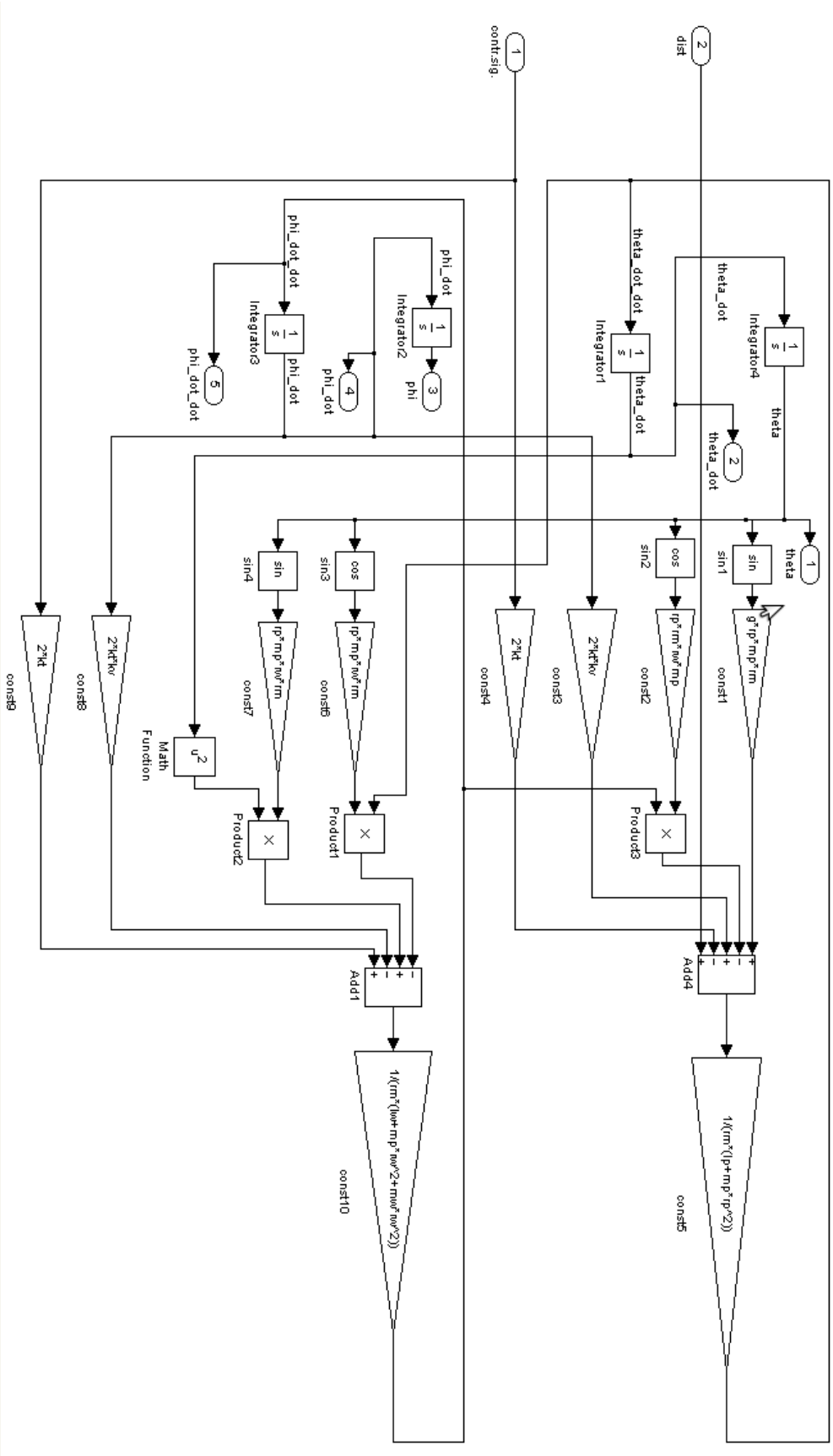
Sensors



Kalman filter



Vehicle


contr.sig.
dist
theta_dot_dot
theta_dot
theta
Integrator1
Integrator4
phi_dot_dot
phi_dot
phi
Integrator3
Integrator2
sin
cos
sin1
sin2
sin3
sin4
const1
const2
const3
const4
const5
const6
const7
const8
const9
const10
Product1
Product2
Product3
Add1
Add4
Math Function
u^2

Mathlab simulation file

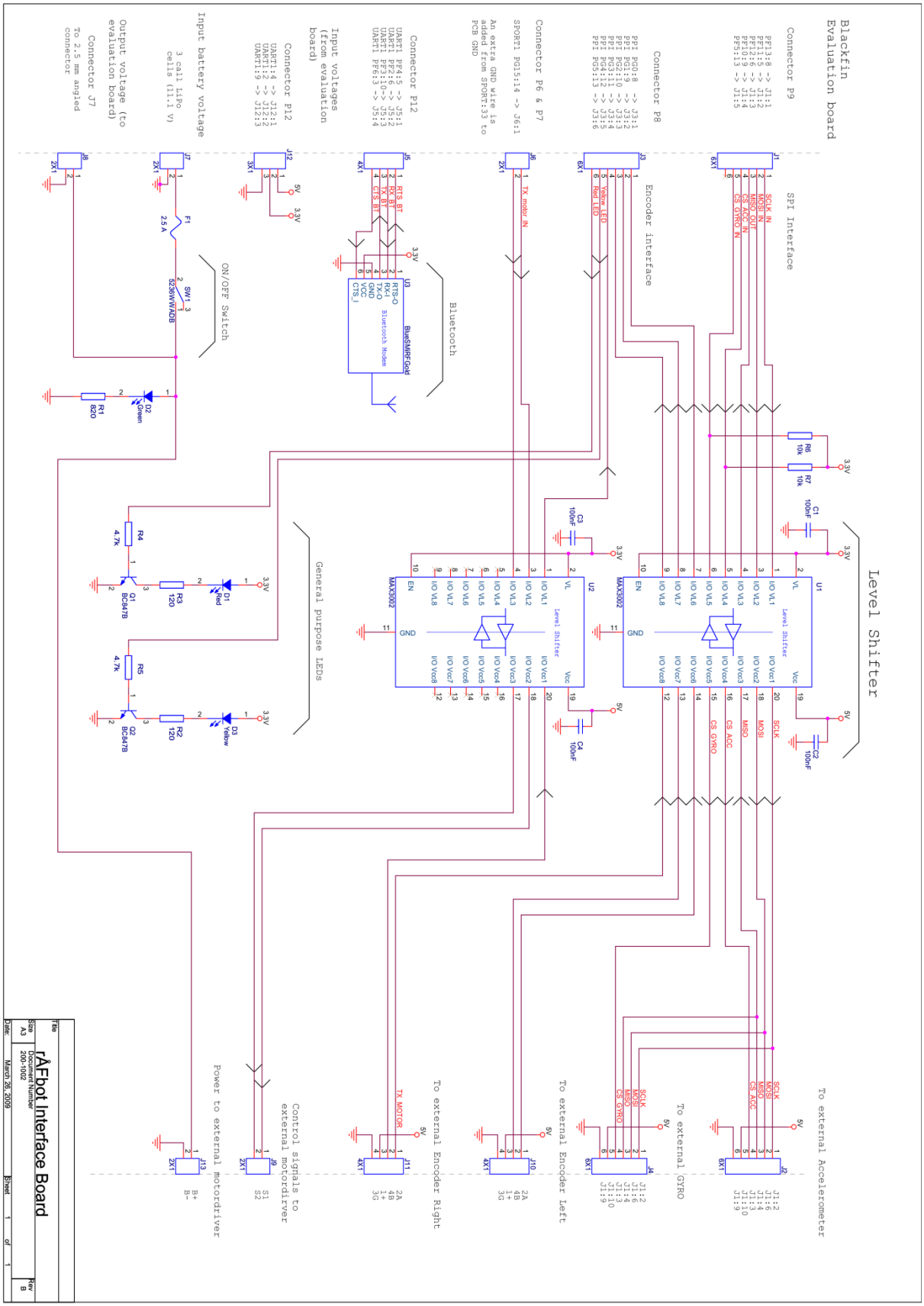
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculate the inertias of the vehicle %%
mtot = 0;                                   % Total mass of the pendulum
ttot = 0;

%% Wheels
%wheels are not included when calculating center of mass
mwheel = 0.03*2;                            % Mass of the wheel
mtot = mtot + mwheel;
rwheel = (0.075/2);                         % Radius of the wheel
Jwheels = ((1/2)*mwheel*rwheel^2)           % Ineria of one wheel times two.

%% Motors
mmotor = 0.05528;                           % mass of the motor
mtot = mtot + mmotor;
rmotor = 0.0135;                            % Radius of the motor
Jmotors = ((1/2)*mmotor*rmotor^2)*2         % Ineria of one motor times two.
% Center of mass calculations
ttot = ttot + mmotor*0;

%% Frame vertical
lframev = 0.50;                             % Lenght of the frame
dframev = 0.0375;
mframev = 0.6*lframev;                      % Mass of the frame
mtot = mtot + mframev;
Jframe1 = (1/12)*mframev*lframev^2 + ...
 ... mframev*(lframev/2 + dframev)^2;       % Ineria of the vertical frame.
 % Center of mass calculations
ttot = ttot + mframev*(lframev/2);

%% Top part of the frame
lframet = 0.10;                             % Lenght of the frame
dframet = 0.5485;
mframet = 0.6*lframet;                      % Mass of the frame
bframet = 0.022;                            % Height of the frame
cframet = 0.1;                              % Width of the frame
mtot = mtot + mframet;
Jframe2 = (1/12)*mframet*(bframet^2 + cframet^2) + ...
    mframet*dframet^2;                      % Ineria of the top frame.
%center of mass calculations
ttot = ttot + mframet*dframet;

%% Frame horisontal (steiners law)
lframeh = 0.3;                              % Length of the frame
mframeh = 0.6*lframeh;                      % Mass of the frame
mtot = mtot + mframeh;
bframeh = 0.022;                            % Height of the frame
cframeh = 0.022;                            % Width of the frame
dframeh = 0.03;                             % Distance from the rotational axis
Jframe3 = (1/12)*mframeh*(bframeh^2 + cframeh^2) + ...
    mframeh*dframeh^2;                      % Ineria of the horisontal frame.
%center of mass calculations
ttot = ttot + mframeh*dframeh;

%% Box containing all the electronics(steiners law vertical placing)
mbox = 1.5;                                 % mass of the microcontroller
mtot = mtot + mbox;
bbox = 0.039;                               % height of the microcontroller
```

```
cbox = 0.2;                                % lenght of the microcontroller
dbox = 0.15;                               % distance from the rotational axis
Jbox = (1/12)*mbox*(bbox^2 + cbox^2) + ...
    mbox*dbox^2;                           % Ineria of the microcontroller.
%center of mass calculations
ttot = ttot + mbox*dbox;

%% Total inertia of the vehicle
mtot
masscenter = ttot/mtot
%Jwheels in not included in Jvehicle
Jvehicle = Jmotors + Jframe1 + Jframe2 +Jframe3 + Jbox

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Constants for simulink model for rÅFbot %%
%% General constants
g = 9.82;                                  %Constant of gravity
Ts = 0.02;                                 %Sampling time

%% DC-motor constants
N = 30;                                    %Gear ratio
rm = 7.4;                                  %Resistance in DC-motor(ohm)
max_omega = 200*(2*pi/(60));
max_volt = 12;
stall_torque = 0.49;
kt = (stall_torque*rm)/max_volt;           %Torque constant in DC-motor(Nm/A)
kv = (stall_torque*rm)/(kt*max_omega);     %Voltage constant DC-motor(Vs/rad)

%% Vehicle constants
Ip = Jvehicle;                             %Inertia for the pendulum
Iw = Jwheels;                              %Inertia of the wheels
rp = masscenter;                           %Length form the wheel rotational
center to the centre of gravity
mp = mtot;                                 %Mass of the pendulum
rw = rwheel;                               %Radius of the wheels
mw = mwheel;                               %Mass of the wheels

%% State feedback controller
alpha = rm*(mp*rp^2+Ip);
beta = rm*(Iw+mp*rw^2+mw*rp^2);
gamma = mp*rp*rm;
A = [0 1 0;
    (beta*gamma*g-rw^2*gamma^2)/(alpha*beta-gamma^2*rw^2) 0
2*kt*kv*(gamma*rw+beta)/(alpha*beta-gamma^2*rw^2);
    (gamma*rw*alpha-gamma^2*rw*g)/(alpha*beta-gamma^2*rw^2) 0 2*kt*kv*(-
gamma*rw-alpha)/(alpha*beta-gamma^2*rw^2)];
B = [0 2*kt*(-gamma*rw-beta)/(alpha*beta-gamma^2*rw^2)
2*kt*(gamma*rw+alpha)/(alpha*beta-gamma^2*rw^2)]';
C = [1 0 0; 0 1 0; 0 0 1];
D = [0 0 0]';

%% LQ calculations
Q = [200000 0 0; 0 1 0; 0 0 1];
R = 10; %plot simulation
[L,S,p] =lqrd(A,B,Q,R,Ts);
Kr = 1/((C*inv(-A+B*L))*B)

%% Kalman calculations
plant = ss(A,B,[1 0 0; 0 1 0; 0 0 0], 0);
plant_cont = plant;
```

```
plant = c2d(plant,Ts);
kalmanQ = 100;
kalmanR = [3 0 0; 0 2 0; 0 0 1];
[kalmf,kalmanL, kalmanP, kalmanM, kalmanZ] = kalman(plant, kalmanQ,
kalmanR);
```

Appendix B

Interface card schematic



Component placement

Bill of material for interface card

Document no: 201-1002
Rev: A
Product name: rÅFbot

Item Number	Quantity	Part Reference	Value	Voltage	Size/Package	Supplier	Supplier_Part_Number
1	4	C1 C2 C3 C4	100nF	50V	0603		
2	1	D1	Red	2.0 V	1206	ELFA	75-307-51
3	1	D2	Green	2.0 V	1206	ELFA	75-311-30
4	1	D3	Yellow	2.0 V	1206	ELFA	75-307-36
5	1	F1	2.5 A	15 V	7.98 X 5.44 X1.25 mm	Farnell	4161087
10	2	Q1 Q2	BC847B	45 V	SOT23	ELFA	73-301-31
11	1	R1	820	50V	0603		
12	2	R2 R3	120	50V	0603		
13	2	R4 R5	4.7k	50V	0603		
14	2	R6 R7	10k	50V	0603		
15	1	SW1	5236WWADB	30 V & 4A	13.2 X 7 X 11 mm	ELFA	35-230-16
16	2	U1 U2	MAX3002	5.5 V	TSSOP20	Farnell	1379762
17	1	U3	BlueSMiRFGold	3.3V-6V	51.5 x 15.8 x 5.6mm	Electrokit	41003012

Appendix C

Component list with cost for the pendulum

Description	Supplier	Quantity	Price	Total price
Header	elfa	1	19.2	19.2
Header	elfa	2	65.6	131.2
Header	elfa	1	9.99	9.99
Header	elfa	4	71.2	284.8
ADIS16003 Accelerometer	elfa	1	512	512
ADIS16080 Gyroscope	elfa	1	788	788
Flat cable	elfa	1	68.3	68.3
ADZS-BF537-STAMP Computer board	farnell	1	2218.69	2218.69
Bluetooth modem BlueSMiRF Gold	electrokit	1	479.2	479.2
GHM-16 motor with gear - 12vdc 30:1 200rpm 6mm	electrokit	2	198.4	396.8
Sabertooth 2X5 motor driver	electrokit	1	543.2	543.2
MMT-02 motor assembly GHM-1-4, 12-13 (par)	electrokit	1	68	68
Aluminum profile, frame 22x22 XDBM L22	alucon	300	0.11	33
Aluminum profile, frame 22x22 XDBM L22	alucon	500	0.11	55
Aluminum profile, frame 22x22 XDBM L22	alucon	100	0.11	11
Profile ending 22x22	alucon	4	10	40
Track nut	alucon	2	42	84
Aluminum angle to the frame 20x20, XDFA 22 A	alucon	4	20	80
Washer nylon M3	farnell	1	11.1	11.1
Distance M3 35mm Ø5mm	efla	6	4.11	24.66
MAXIM - MAX3002EUP Level translator	farnell	2	44.68	89.36
POLYFUSE, SMD2920 2.5A	farnell	2	1.82	3.64
Screw M3	clasohlson	1	9	9
Screw M3	clasohlson	1	10	10
Nut M3	clasohlson	2	8	16
Screw M5	clasohlson	1	24	24
Plexiglass	clasohlson	2	49	98
Washer M5	clasohlson	1	9	9
PCB	Cogra	1	575	575
Battery charger 1-4CELLER LI-PO E-SKY	hobbex	1	395	395
Battery LI-Po 11,1V 1500mAh	hobbex	1	475	475
Battery contacts	hobbex	2	40	80
				7562.14

Appendix D

Source code for the inverted pendulum controller

```
/*
 * controller.h
 *
 *  Created on: Feb 26, 2009
 *      Author: blackfin
 */

#ifndef CONTROLLER_H_
#define CONTROLLER_H_

#include "interface_driver.h"
#include "kalman.h"
#include "fixpoint.h"
#include "data_converter.h"

#include <stdio.h>
#include <stdlib.h>

#include <signal.h>

#include <native/task.h>
#include <native/mutex.h>
#include <native/timer.h>
#include <sys/mman.h>


void stop_controller();


#endif /* CONTROLLER_H_ */
```

```
/*
 * controller.c
 *
 *  Created on: Feb 26, 2009
 *      Author: blackfin
 *
 *      controller.c implements the control loop and the communication loop,
 *      it also contains a PID controller, a LQ controller, drive functions,
 *      turn functions and the main functions.
 */

#include "controller.h"

// static variables
static const float START_REF_FRAC = -0.0305;
static const int DEBUG = 0;
static const int CONTROL_LOOP_PRIO = 99;
static const int MOVEMENT_PRIO = 50;
static const int COM_GET_INDATA_PRIO = 2;
static const int COM_HANDLE_INDATA_PRIO = 2;
static const int LOG_HANDLE_INDATA_PRIO = 2;
static const int MAX_CONT_OUTPUT = 11;

static const RTIME CONTROL_LOOP_DELAY = 20000000u; //200 Hz
static const RTIME COM_GET_INDATA_DELAY = 10000000u;
static const RTIME LOG_DATA_DELAY = 10000000u;
static const RTIME END_DELAY = 2000000000u;
static const RTIME TURN_DELAY = 150000000u;
static const RTIME DRIVE_DELAY = 900000000u;
static const RTIME DRIVE_DELAY_2 = 10000000u;
static const RTIME DRIVE_DELAY_3 = 100000000u;

//loop variables
int end_control_tasks = 0;
int end_com_get_tasks = 0;
int logging_data = 0;

RT_TASK control_loop_task;
RT_TASK com_handle_indata_task;
RT_TASK com_get_indata_task;
RT_TASK log_data_task;

RT_MUTEX control_loop_mutex;
RT_MUTEX com_data_mutex;
RT_MUTEX state_vector_mutex;
RT_MUTEX reference_signal_mutex;
RT_MUTEX output_signal_mutex;
RT_MUTEX control_parameter_mutex;
RT_MUTEX turn_mutex;
RT_MUTEX drive_mutex;

//controller varibles
FIXED22_10 X_hat[8];
FIXED22_10 r;
FIXED22_10 u_turn[2];
FIXED22_10 turn_speed1;
FIXED22_10 turn_speed2;
FIXED22_10 r_drive_maxspeed;
FIXED22_10 r_drive_speed_frac;
FIXED22_10 r_drive_maxspeed_div;
FIXED22_10 r_drive_speed1;
FIXED22_10 r_drive_speed2;
FIXED22_10 r_drive_speed3;
FIXED22_10 DYNAMIC_OFSET_STEP;
FIXED22_10 r_step;
FIXED22_10 P_step;
FIXED22_10 Ki_step;
FIXED22_10 Kd_step;
FIXED22_10 u;
FIXED22_10 u_tmp_turn[2];
FIXED22_10 u_div;
FIXED22_10 MAX_U;
FIXED22_10 ZERO_P_M1;
FIXED22_10 ZERO_P_M2;
FIXED22_10 MAX_ANGEL;

//PID controller variables
```

```
FIXED22_10 theta_c_in;
FIXED22_10 error;
FIXED22_10 error_old;
FIXED22_10 i_old;
FIXED22_10 der_pid;
FIXED22_10 P;
FIXED22_10 Ki;
FIXED22_10 Kd;
FIXED22_10 sampletime;

//LQ controller variables
FIXED22_10 L[5];
FIXED22_10 Kr;

//data buffer variables
unsigned char in_pay_load[256];
unsigned char out_pay_load[256];
unsigned char out_packet[259];
unsigned char start_data[2];
unsigned char u_out[2];

//send communication data
void com_send_outdata(char *data)        {
              out_packet[0] = 0x01;
              out_packet[1] = strlen(data) + 1;
              unsigned int i = 0;
              for(; i<out_packet[1]; i++){
                            out_packet[i + 2] = data[i];
              }
              send_out_data(out_packet);
}

//set turn speed
void turn_right(FIXED22_10 turn_speed)   {

              if(!rt_mutex_acquire(&turn_mutex, TM_INFINITE))       {
                            rt_mutex_acquire(&output_signal_mutex, TM_INFINITE);
                            u_turn[0].full = turn_speed.full;
                            u_turn[1].full = -turn_speed.full;
                            rt_mutex_release(&output_signal_mutex);
                            rt_mutex_release(&turn_mutex);
              }

}

//set turn speed
void turn_left(FIXED22_10 turn_speed)    {

              if(!rt_mutex_acquire(&turn_mutex, TM_INFINITE))       {
                            rt_mutex_acquire(&output_signal_mutex, TM_INFINITE);
                            u_turn[0].full = -turn_speed.full;
                            u_turn[1].full = turn_speed.full;
                            rt_mutex_release(&output_signal_mutex);
                            rt_mutex_release(&turn_mutex);
              }

}

//set drive speed
void drive_forward(FIXED22_10 speed)     {

              if(!rt_mutex_acquire(&drive_mutex, TM_INFINITE))      {
                            rt_mutex_acquire(&reference_signal_mutex, TM_INFINITE);
                            r.full = -speed.full;
                            rt_mutex_release(&reference_signal_mutex);
                            rt_mutex_release(&drive_mutex);
              }

}

//set drive speed
void drive_backwards(FIXED22_10 speed)   {

              if(!rt_mutex_acquire(&drive_mutex, TM_INFINITE))      {
                            rt_mutex_acquire(&reference_signal_mutex, TM_INFINITE);
                            r.full = speed.full;
                            rt_mutex_release(&reference_signal_mutex);
```

```
                        rt_mutex_release(&drive_mutex);
                }

}

//set drive to halt
void stop_drive()           {

                rt_mutex_acquire(&drive_mutex, TM_INFINITE);
                rt_mutex_acquire(&reference_signal_mutex, TM_INFINITE);
                r.full = 0;
                rt_mutex_release(&reference_signal_mutex);
                rt_mutex_release(&drive_mutex);

}

//set turn to halt
void stop_turn()            {

                rt_mutex_acquire(&turn_mutex, TM_INFINITE);
                rt_mutex_acquire(&output_signal_mutex, TM_INFINITE);
                u_turn[0].full = 0;
                u_turn[1].full = 0;
                rt_mutex_release(&output_signal_mutex);
                rt_mutex_release(&turn_mutex);

}

//handle received data
void com_handle_indata(char data)         {
                //printf("com_handeler %c\n", data);
                if(data == 0x02)            {
                        //request status
                        char c[100];
                        rt_mutex_acquire(&reference_signal_mutex, TM_INFINITE);
                        rt_mutex_acquire(&output_signal_mutex, TM_INFINITE);
                        rt_mutex_acquire(&state_vector_mutex, TM_INFINITE);
                        FIXED22_10 g;
                        get_theta_dot(&g);
                        sprintf (c, "\x10g: %d r: %d x1: %d x2: %d x3: %d x4: %d\nu_tmp1:
%d u_tmp2: %d offset: %d", g.full, r.full, X_hat[0].full, X_hat[4].full, X_hat[2].full,
X_hat[3].full, u_tmp_turn[0].full, u_tmp_turn[1].full, start_offset.full);
                        rt_mutex_release(&reference_signal_mutex);
                        rt_mutex_release(&output_signal_mutex);
                        rt_mutex_release(&state_vector_mutex);
                        com_send_outdata(c);
                }
                else if(data == 0x03)       {
                        //request status
                        char c[100];
                        rt_mutex_acquire(&reference_signal_mutex, TM_INFINITE);
                        rt_mutex_acquire(&output_signal_mutex, TM_INFINITE);
                        rt_mutex_acquire(&state_vector_mutex, TM_INFINITE);

                        if(!end_control_tasks)      {
                                sprintf (c, "\x14%d\xfeRunning\xff", X_hat[0].full);
                        }
                        else          {
                                sprintf (c, "\x14%d\xfeStopped\xff", X_hat[0].full);
                        }

                        rt_mutex_release(&reference_signal_mutex);
                        rt_mutex_release(&output_signal_mutex);
                        rt_mutex_release(&state_vector_mutex);
                        com_send_outdata(c);
                }
                else if(data == 0x1E)       {
                        drive_forward(r_drive_speed1);
                        //printf("com_handeler %c\n", data);
                }
                else if(data == 0x1F)       {
                        drive_forward(r_drive_speed2);
                        //printf("com_handeler %c\n", data);
                }
                else if(data == 0x20)       {
                        drive_forward(r_drive_speed3);
                        //printf("com_handeler %c\n", data);
```

```
		}
		else if(data == 0x21)	{
				drive_backwards(r_drive_speed1);
				//printf("com_handeler %c\n", data);
		}
		else if(data == 0x22)	{
				drive_backwards(r_drive_speed2);
				//printf("com_handeler %c\n", data);
		}
		else if(data == 0x23)	{
				drive_backwards(r_drive_speed3);
				//printf("com_handeler %c\n", data);
		}
		else if(data == 0x24)	{
				stop_drive();
				//printf("com_handeler %c\n", data);
		}
		else if(data == 0x25)	{
				turn_right(turn_speed1);
				//printf("com_handeler %c\n", data);
		}
		else if(data == 0x26)	{
				turn_right(turn_speed2);
				//printf("com_handeler %c\n", data);
		}
		else if(data == 0x27)	{
				turn_left(turn_speed1);
				//printf("com_handeler %c\n", data);
		}
		else if(data == 0x28)	{
				turn_left(turn_speed2);
				//printf("com_handeler %c\n", data);
		}
		else if(data == 0x29)	{
				stop_turn();
				//printf("com_handeler %c\n", data);
		}
		else if(data == 'R')	{
				rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
				start_offset.full = start_offset.full + r_step.full;
				rt_mutex_release(&control_parameter_mutex);
				//printf("com_handeler %c %d : %d\n", data, start_offset.full,
r.full);
		}
		else if(data == 'r')	{
				rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
				start_offset.full = start_offset.full - r_step.full;
				rt_mutex_release(&control_parameter_mutex);
				//printf("com_handeler %c %d : %d\n", data, start_offset.full,
r.full);
		}
		else if(data == 'L')	{
				//start log
				//printf("com_handeler %c\n", data);
				init_write_log_data();
				rt_mutex_acquire(&reference_signal_mutex, TM_INFINITE);
				rt_mutex_acquire(&output_signal_mutex, TM_INFINITE);
				rt_mutex_acquire(&state_vector_mutex, TM_INFINITE);
				rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
				logging_data = 1;
				rt_mutex_release(&reference_signal_mutex);
				rt_mutex_release(&output_signal_mutex);
				rt_mutex_release(&state_vector_mutex);
				rt_mutex_release(&control_parameter_mutex);
		}
		else if(data == 'l')	{
				//stop log
				rt_mutex_acquire(&reference_signal_mutex, TM_INFINITE);
				rt_mutex_acquire(&output_signal_mutex, TM_INFINITE);
				rt_mutex_acquire(&state_vector_mutex, TM_INFINITE);
				rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
				logging_data = 0;
				rt_mutex_release(&reference_signal_mutex);
				rt_mutex_release(&output_signal_mutex);
				rt_mutex_release(&state_vector_mutex);
				rt_mutex_release(&control_parameter_mutex);
				close_log_data();
```

```
                        //printf("com_handeler %c\n", data);
                }
                else if(data == 'G')        {
                        //get log
                        //printf("com_handeler %c\n", data);
                        if(!logging_data)           {
                                init_read_log_data();
                                char log[256];
                                if(!read_log_data(log))     {
                                        com_send_outdata("\x11");
                                        char pay[256];
                                        do              {
                                                strcpy(pay,"\x12");
                                                strcat(pay,log);
                                                com_send_outdata(pay);
                                        }while(!read_log_data(log));
                                        com_send_outdata("\x13");
                                        }
                                close_log_data();
                        }
                        else            {
                                char *c = "\x10You can not get log file when logging,
press \'l\' first.";
                                com_send_outdata(c);
                        }
                }
                else if(data == 'S')        {
                        reinit_controller();
                        //printf("com_handeler %c\n", data);

                }
                else if(data == 's')        {
                        //printf("com_handeler %c\n", data);
                        stop_controller();
                }
                else if(data == 'A')        {
                        //printf("com_handeler %c\n", data);
                        led_on_red();
                }
                else if(data == 'a')        {
                        //printf("com_handeler %c\n", data);
                        led_off_red();
                }
                else if(data == 'B')        {
                        //printf("com_handeler %c\n", data);
                        led_on_yellow();
                }
                else if(data == 'b')        {
                        //printf("com_handeler %c\n", data);
                        led_off_yellow();
                }
                else if(data == 'm')        {
                        //printf("com_handeler %c\n", data);
                        unsigned char m[2];
                        m[0] = 0;
                        m[1] = 0;
                        set_motor_driver(m);
                }
                else if(data == 'M')        {
                        //printf("com_handeler %c\n", data);
                        unsigned char m[2];
                        m[0] = 30;
                        m[1] = 157;
                        set_motor_driver(m);
                }
                else if(data == 'P')        {

                        rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
                        P.full = P.full + P_step.full;
                        //printf("P inc %d\n", P.full);
                        rt_mutex_release(&control_parameter_mutex);
                }
                else if(data == 'p')        {
                        //printf("P dec\n");
                        rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
                        P.full = P.full - P_step.full;
                        //printf("P inc %d\n", P.full);
```

```
                        rt_mutex_release(&control_parameter_mutex);
                }
                else if(data == 'I')        {
                        //printf("I inc\n");
                        rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
                        Ki.full = Ki.full + Ki_step.full;
                        L[4].full = L[4].full + Ki_step.full;
                        //printf("Ki inc %d %d\n", Ki.full, L[4].full);
                        rt_mutex_release(&control_parameter_mutex);
                }
                else if(data == 'i')        {
                        //printf("com_handeler %c\n", data);
                        rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
                        Ki.full = Ki.full - Ki_step.full;
                        L[4].full = L[4].full - Ki_step.full;
                        //printf("Ki dec %d %d\n", Ki.full, L[4].full);
                        rt_mutex_release(&control_parameter_mutex);
                }
                else if(data == 'D')        {
                        //printf("com_handeler %c\n", data);
                        rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
                        Kd.full = Kd.full + Kd_step.full;
                        //printf("Kd inc %d\n", Kd.full);
                        rt_mutex_release(&control_parameter_mutex);
                }
                else if(data == 'd')        {
                        //printf("com_handeler %c\n", data);
                        rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);
                        Kd.full = Kd.full - Kd_step.full;
                        //printf("Kd dec %d\n", Kd.full);
                        rt_mutex_release(&control_parameter_mutex);
                }
}

//get in data
void com_get_indata_loop() {
                RT_TASK_INFO info;
                RTIME start, done;
                rt_task_inquire(rt_task_self(), &info);
                rt_task_set_periodic(NULL, TM_NOW, COM_GET_INDATA_DELAY);


                while(!end_com_get_tasks)  {
                        start = rt_timer_read();
                        if(DEBUG)     {
                                printf("message from %s\n", info.name);
                        }
                        get_com_data(&start_data[0]);
                        //printf("indata_message: %d\n", start_data[0]);
                        if(start_data[0] == 0x01)  {
                                get_in_data(&start_data[1]);
                                unsigned int i = 0;
                                for(; i<start_data[1]; i++){
                                        get_com_data(&in_pay_load[i]);
                                }
                                //no messages with payload over 1 byte are sent to
the server.
                                com_handle_indata(in_pay_load[0]);
                        }
                        done = rt_timer_read();
                        rt_task_wait_period(NULL);

                }
}

//the LQ controller
void lq_controller()        {

                get_new_state_vector_state(X_hat,u);

                theta_c_in.full = X_hat[0].full;
                //printf("%d:%d:%d:%d:%d:%d:%d\n", X_hat[0].full, X_hat[1].full, X_hat[2].full,
X_hat[3].full, X_hat[4].full, X_hat[5].full, u.full);
                if ((theta_c_in.full > MAX_ANGEL.full) || (theta_c_in.full < -MAX_ANGEL.full))
                {
                        //printf("end controller\n");
                        end_control_tasks = 1;
```

```
		}
		else		{

				u.full = MULT22_10(r,Kr) - (MULT22_10(X_hat[0],L[0]) +
MULT22_10(X_hat[1],L[1]) + MULT22_10(X_hat[2],L[2]));

				//Correct output when turning
				u_tmp_turn[0].full = (u.full + u_turn[0].full);
				u_tmp_turn[1].full = (u.full + u_turn[1].full);

				if(u_tmp_turn[0].full >= MAX_U.full)	{
						u_tmp_turn[0].full = MAX_U.full;
				}
				else if (u_tmp_turn[0].full <= -MAX_U.full)		{
						u_tmp_turn[0].full = -MAX_U.full;
				}

				if(u_tmp_turn[1].full >= MAX_U.full)	{
						u_tmp_turn[1].full = MAX_U.full;
				}
				else if (u_tmp_turn[1].full <= -MAX_U.full)		{
						u_tmp_turn[1].full = -MAX_U.full;
				}

				if(!rt_mutex_acquire(&turn_mutex, TM_NONBLOCK))		{
						if(!rt_mutex_acquire(&drive_mutex, TM_NONBLOCK))
		{
								if(r.full == 0){
										start_offset.full +=
MULT22_10(DYNAMIC_OFSET_STEP,u);
								}
								rt_mutex_release(&drive_mutex);
						}
						rt_mutex_release(&turn_mutex);
				}

				set_u_value(u_tmp_turn);

		}
}

//The PID controller
void pid_controller()	{

		get_new_state_vector_state(X_hat,u);

		theta_c_in.full = -X_hat[0].full;
		//printf("%d:%d:%d:%d:%d:%d:%d\n", X_hat[0].full, X_hat[1].full, X_hat[2].full,
X_hat[3].full, X_hat[4].full, X_hat[5].full, u.full);

		if ((theta_c_in.full > MAX_ANGEL.full) || (theta_c_in.full < -MAX_ANGEL.full))
		{
				//printf("end controller\n");
				end_control_tasks = 1;
		}
		else		{
				error.full = r.full - theta_c_in.full;	//left

				der_pid.full = error.full - error_old.full;		//left
derivative
				der_pid.full = DIV22_10(der_pid, sampletime);
				der_pid.full = MULT22_10(Kd, der_pid);

//				//Calculate output signal one
				u.full = i_old.full + der_pid.full + MULT22_10(P,error);

				u_tmp_turn[0].full = (u.full + u_turn[0].full);
				u_tmp_turn[1].full = (u.full + u_turn[1].full);

				if(u_tmp_turn[0].full >= MAX_U.full)	{
						u_tmp_turn[0].full = MAX_U.full;
				}
				else if (u_tmp_turn[0].full <= -MAX_U.full)		{
						u_tmp_turn[0].full = -MAX_U.full;
				}

				if(u_tmp_turn[1].full >= MAX_U.full)	{
```

```
                                u_tmp_turn[1].full = MAX_U.full;
                        }
                        else if (u_tmp_turn[1].full <= -MAX_U.full)           {
                                u_tmp_turn[1].full = -MAX_U.full;
                        }

                        if(u.full >= MAX_U.full)    {
                                u.full = MAX_U.full;
                        }
                        else if (u.full <= -MAX_U.full)          {
                                u.full = -MAX_U.full;
                        }
                        else          {
                                i_old.full = i_old.full + MULT22_10(Ki,error);
                        }

                        set_u_value(u_tmp_turn);

                        if(!rt_mutex_acquire(&turn_mutex, TM_NONBLOCK))       {
                                if(!rt_mutex_acquire(&drive_mutex, TM_NONBLOCK))
                {
                                                start_offset.full +=
MULT22_10(DYNAMIC_OFSET_STEP,u);
                                                rt_mutex_release(&drive_mutex);
                                }
                                rt_mutex_release(&turn_mutex);
                        }

                        error_old.full = error.full;

                }
}

//The control loop
void control_loop()        {
                RT_TASK_INFO info;
                RTIME start, done, done_calc, old_calc_time, done_log;
                done = 0;
                old_calc_time = 0;
                rt_task_inquire(rt_task_self(), &info);
                rt_task_set_periodic(NULL, TM_NOW, CONTROL_LOOP_DELAY);

                while(!end_control_tasks)  {

                        rt_task_set_mode(0,T_PRIMARY,NULL);
                        start = rt_timer_read();
                        if(DEBUG)     {
                                printf("message from %s\n", info.name);
                        }

                        rt_mutex_acquire(&reference_signal_mutex, TM_INFINITE);
                        rt_mutex_acquire(&output_signal_mutex, TM_INFINITE);
                        rt_mutex_acquire(&state_vector_mutex, TM_INFINITE);
                        rt_mutex_acquire(&control_parameter_mutex, TM_INFINITE);

                        //pid_controller();
                        lq_controller();

                        done_calc = rt_timer_read();
                        if(logging_data)           {
                                char log_data[256];
                                char numbers[256];
                                strcpy(log_data,"");
                                sprintf(numbers,
"%d:%d:%d:%d:%d:%d:%d:%d:%lld:%lld:%d:%d:%d",r.full, X_hat[0].full, X_hat[1].full,
X_hat[2].full, X_hat[3].full, X_hat[4].full, X_hat[5].full, u_tmp_turn[0].full,
u_tmp_turn[1].full, (done_calc - start), (start - done), X_hat[6].full, X_hat[7].full,
i_old.full);
                                strcat(log_data,numbers);
                                write_log_data(log_data);
                        }
                        rt_mutex_release(&reference_signal_mutex);
                        rt_mutex_release(&output_signal_mutex);
                        rt_mutex_release(&state_vector_mutex);
                        rt_mutex_release(&control_parameter_mutex);
                        done_log = rt_timer_read();
                        old_calc_time = (done_log - start);
```

```
                        done = rt_timer_read();
                        rt_task_wait_period(NULL);
                }
                stop_motor_driver();
                led_low_yellow();
}

//reinitiation of the controller
int reinit_controller()    {
                end_control_tasks = 1;
                sleep(2);
                end_control_tasks = 0;
                init_kalman();
                reinit_converter();

                //initiate the X_hat vector.
                X_hat[0].full = FIXED22_10CONST(0,0);
                X_hat[1].full = FIXED22_10CONST(0,0);
                X_hat[2].full = FIXED22_10CONST(0,0);
                X_hat[3].full = FIXED22_10CONST(0,0);


                //initiate the start values on u, u_combined and u_div.
                u.full = FIXED22_10CONST(0,0);
                u_div.full = FIXED22_10CONST(2,0);
                u_tmp_turn[0].full = FIXED22_10CONST(0,0);
                u_tmp_turn[1].full = FIXED22_10CONST(0,0);
                r.full = 0;

                //Initiate PID controller constants
                i_old.full = FIXED22_10CONST(0,0);
                error_old.full = FIXED22_10CONST(0,0);

                led_high_yellow();
                //start control loops
                rt_task_spawn(&control_loop_task, "control_loop_task", 0, CONTROL_LOOP_PRIO, 0,
&control_loop, NULL);

                return 0;

}

//initiate the contoller
int init_controller()      {

                start_offset.full = 1300;
                start_offset2.full = 1007;

                init_kalman();
                init_converter();

                //initiate the X_hat vector.
                X_hat[0].full = FIXED22_10CONST(0,0);
                X_hat[1].full = FIXED22_10CONST(0,0);
                X_hat[2].full = FIXED22_10CONST(0,0);
                X_hat[3].full = FIXED22_10CONST(0,0);
                X_hat[4].full = FIXED22_10CONST(0,0);
                X_hat[5].full = FIXED22_10CONST(0,0);

                //initiate the reference values r and r_step.
                r.full = 0;
                r_step.full = FIXED22_10CONST(0,0.0003);

                MAX_ANGEL.full = 45000;

                //initiate the start values on u, u_combined and u_div.
                u.full = FIXED22_10CONST(0,0);

                u_tmp_turn[0].full = FIXED22_10CONST(0,0);
                u_tmp_turn[1].full = FIXED22_10CONST(0,0);
                u_div.full = FIXED22_10CONST(2,0);
                MAX_U.full = FIXED22_10CONST(MAX_CONT_OUTPUT,0);

                //Initiate PID controller constants
                float samp = (float)CONTROL_LOOP_DELAY/1000000000;
                sampletime.full = FIXED22_10CONST(0,samp);
                P.full = 22598520;
```

```
		Ki.full = 5418070;
		Kd.full = 86135;
		i_old.full = FIXED22_10CONST(0,0);
		error_old.full = FIXED22_10CONST(0,0);
		P_step.full = FIXED22_10CONST(0,0.5);
		Ki_step.full = FIXED22_10CONST(0,0.1);
		Kd_step.full = FIXED22_10CONST(0,0.01);

		//Initiate LQ controller constants

		L[0].full = FIXED22_10CONST(-263,-0.8836);
		L[1].full = FIXED22_10CONST(-28,-0.0161);
		L[2].full = FIXED22_10CONST(-1,-0.7480);

		Kr.full = FIXED22_10CONST(-1,-0.1750);

		//Initiate movement paramenters
		u_turn[0].full = FIXED22_10CONST(0,0);
		u_turn[1].full = FIXED22_10CONST(0,0);
		turn_speed1.full = FIXED22_10CONST(1,0.5);
		turn_speed2.full = FIXED22_10CONST(3,0);
		r_drive_maxspeed.full = FIXED22_10CONST(11,0);
		r_drive_maxspeed_div.full = FIXED22_10CONST(3,0);
		r_drive_speed_frac.full = DIV22_10(r_drive_maxspeed, r_drive_maxspeed_div);
		r_drive_speed1.full = r_drive_speed_frac.full;
		r_drive_speed2.full = r_drive_speed_frac.full + r_drive_speed_frac.full;
		r_drive_speed3.full = r_drive_speed_frac.full + r_drive_speed_frac.full +
r_drive_speed_frac.full;
		DYNAMIC_OFSET_STEP.full = FIXED22_10CONST(0,0.0001);

		//create mutex for data get/set loops
		int mux_err = rt_mutex_create(&control_loop_mutex, "CONTROL_LOOP_MUTEX");
		if (mux_err < 0) {
			perror("mutex CONTROL_LOOP_MUTEX");
			return 1;
		}
		//create mutex for data get/set loops
		mux_err = rt_mutex_create(&com_data_mutex, "COM_DATA_MUTEX");
		if (mux_err < 0) {
			perror("mutex COM_DATA_MUTEX");
			return 1;
		}
		//create mutex for state vector
		mux_err = rt_mutex_create(&state_vector_mutex, "STATE_VECTOR_SIGNAL_MUTEX");
		if (mux_err < 0) {
			perror("mutex STATE_VECTOR_SIGNAL_MUTEX");
			return 1;
		}
		//create mutex for reference signal
		mux_err = rt_mutex_create(&reference_signal_mutex, "REFERENCE_SIGNAL_MUTEX");
		if (mux_err < 0) {
			perror("mutex REFERENCE_SIGNAL_MUTEX");
			return 1;
		}
		//create mutex for output signal
		mux_err = rt_mutex_create(&output_signal_mutex, "OUTPUT_SIGNAL_MUTEX");
		if (mux_err < 0) {
			perror("mutex OUTPUT_SIGNAL_MUTEX");
			return 1;
		}
		//create mutex for control parameters P, Ki, Kd
		mux_err = rt_mutex_create(&control_parameter_mutex, "CONTROL_PARAMETER_MUTEX");
		if (mux_err < 0) {
			perror("mutex CONTROL_PARAMETER_MUTEX");
			return 1;
		}
		//create mutex for turning right and left
		mux_err = rt_mutex_create(&turn_mutex, "TURN_MUTEX");
		if (mux_err < 0) {
			perror("mutex TURN_MUTEX");
			return 1;
		}
		//create mutex for driving forward and backwards
		mux_err = rt_mutex_create(&drive_mutex, "DRIVE_MUTEX");
		if (mux_err < 0) {
			perror("mutex DRIVE_MUTEX");
			return 1;
```

```
                }

                led_high_red();

                end_control_tasks = 0;
                sleep(1);
                //start control loops
                int err;
                err = rt_task_spawn(&control_loop_task, "control_loop_task", 0,
CONTROL_LOOP_PRIO, 0, &control_loop, NULL);
                led_high_yellow();
                err = rt_task_spawn(&com_get_indata_task, "com_get_indata_task", 0,
COM_HANDLE_INDATA_PRIO, 0, &com_get_indata_loop, NULL);
                if (err)
                {
                        //printf("com_get_indata_task: %d\n", err);
                        return 0;
                }

                return 0;

}

//stop the controller
void stop_controller()      {

                end_control_tasks = 1;
                //sleep 1 sec
                sleep(1);
                //end tasks
                rt_task_delete(&control_loop_task);

}

//end the controller
void end_controller()       {

                led_low_yellow();
                end_control_tasks = 1;
                end_com_get_tasks = 1;
                //sleep 1 sec
                sleep(1);
                //end tasks
                rt_task_delete(&control_loop_task);
                rt_task_delete(&com_get_indata_task);

                //delete mutex
                rt_mutex_delete(&control_loop_mutex);
                rt_mutex_delete(&com_data_mutex);
                rt_mutex_delete(&state_vector_mutex);
                rt_mutex_delete(&reference_signal_mutex);
                rt_mutex_delete(&output_signal_mutex);

}

// signal-handler, to ensure clean exit on Ctrl-C
void clean_exit(int dummy) {
                if(!end_control_tasks)      {
                        //printf("cleanup\n");

                        end_controller();
                        led_low_red();
                        //sleep 1 sec
                        sleep(1);

                        end_interface_driver();
                        //printf("end\n");
                }


}

//the main function
int main(int argc, char **argv){
                // install signal handler
                signal(SIGTERM, clean_exit);
                signal(SIGINT, clean_exit);
```

```
                mlockall(MCL_CURRENT | MCL_FUTURE);

                init_controller();

                pause();
                fflush(NULL);
                return 0;
}
```

```c
/*
 * data_converter.h
 *
 *  Created on: Feb 26, 2009
 *      Author: blackfin
 */

#ifndef DATA_CONVERTER_H_
#define DATA_CONVERTER_H_

#include <stdio.h>
#include "fixpoint.h"
#include "interface_driver.h"
#include <stdlib.h>

static const int MAX_SENSOR_ACC_X = 0;
static const int MAX_SENSOR_ACC_Y = 0;
static const int MIN_SENSOR_ACC_X = 0;
static const int MIN_SENSOR_ACC_Y = 0;

static const int MAX_SENSOR_GYRO = 0;
static const int MIN_SENSOR_GYRO = 0;

static const int MAX_MOTOR_DRIVER = 0;
static const int MIN_MOTOR_DRIVER = 0;

int old_theta;
int old_theta_dot;

int init_converter();
int reinit_converter();

void get_theta(FIXED22_10 t[], FIXED22_10 *kalmest);
void get_theta_dot();

void set_u_value(FIXED22_10 u[]);
void stop_motor_driver();

void led_high_red();
void led_low_red();
void led_high_yellow();
void led_low_yellow();

void send_out_data(unsigned char data[]);
void get_in_data(unsigned char *data);

void conv_xy_raw_to_g(FIXED22_10 *raw_data);

#endif /* DATA_CONVERTER_H_ */
```

```
/*
 * data_converter.c
 *
 *  Created on: Feb 26, 2009
 *      Author: blackfin
 *
 *      data_converter.c is used to convert raw data into fixpoint representation,
 *      and from fixpoint to raw data.
 */

#include "data_converter.h"

//Allocate varables

//sensor varables
__u16 raw_acc_data[2];
__u16 raw_gyro_data[1];
FIXED22_10 sx;
FIXED22_10 sy;
FIXED22_10 g;
FIXED22_10 asin_const;

//Constant variables for conv_xy_raw_to_g
FIXED22_10 lsb_g;
FIXED22_10 zeropoint;

//Constants for newton rapson
FIXED22_10 theta_new;
FIXED22_10 theta_old;
FIXED22_10 fx_old;
FIXED22_10 fx_old_old;
FIXED22_10 theta_der_old;
FIXED22_10 sin_theta;
FIXED22_10 cos_theta;
FIXED22_10 temp;
FIXED22_10 k;
FIXED22_10 k2;
FIXED22_10 MIN_DTHETA;
FIXED22_10 ZERO_POINT_M1;
FIXED22_10 ZERO_POINT_M2;
FIXED22_10 SCALE_U_TO_MOTOR;
FIXED22_10 DEG_TO_RAD;
FIXED22_10 PREV_ANGLE;
FIXED22_10 MAX_ANGLE;



FIXED22_10 THETA_OFFSET;
FIXED22_10 MIN_FX;
FIXED22_10 MIN_DERR;
FIXED22_10 FX_INT;
FIXED22_10 NEWTON_K;



//Init function for data converter
int init_converter()       {
              //Create fixed point variables
              g.full = FIXED22_10CONST(9,0.82);        //g
              asin_const.full = FIXED22_10CONST(3,0);  //constans used in the series expansion
of asin
              lsb_g.full = FIXED22_10CONST(820,0);     //LSB/g for accelerometer
              zeropoint.full = FIXED22_10CONST(2048,0);             //input from accelerometer
when angel = 0rad
              ZERO_POINT_M1.full = FIXED22_10CONST(64,0);
              ZERO_POINT_M2.full = FIXED22_10CONST(191,0);
              SCALE_U_TO_MOTOR.full = FIXED22_10CONST(5,0); // 63/11
              DEG_TO_RAD.full = FIXED22_10CONST(-0,-0.001705); //deg to rad and lsb/rad/s

              //Init newton rapson
              theta_old.full = FIXED22_10CONST(0,0);
              PREV_ANGLE.full = FIXED22_10CONST(0,0);
              MAX_ANGLE.full = FIXED22_10CONST(2,0);

              THETA_OFFSET.full = FIXED22_10CONST(0,0.02);

              MIN_FX.full = FIXED22_10CONST(0,0.0005);
```

```
		MIN_DERR.full = FIXED22_10CONST(5,0);
		FX_INT.full = FIXED22_10CONST(1,0);
		NEWTON_K.full = FIXED22_10CONST(2,0);

		init_interface_driver();

		return 0;
}

//Init function for data converter
int reinit_converter()	{
		//Create fixed point variables
		g.full = FIXED22_10CONST(9,0.82);	//g
		asin_const.full = FIXED22_10CONST(3,0);  //constans used in the series expansion
of asin
		lsb_g.full = FIXED22_10CONST(820,0);	//LSB/g for accelerometer
		zeropoint.full = FIXED22_10CONST(2048,0);		//input from accelerometer
when angel = 0rad
		ZERO_POINT_M1.full = FIXED22_10CONST(64,0);
		ZERO_POINT_M2.full = FIXED22_10CONST(191,0);
		SCALE_U_TO_MOTOR.full = FIXED22_10CONST(5,0); // 63/11
		DEG_TO_RAD.full = FIXED22_10CONST(-0,-0.001705); //deg to rad and lsb/rad/s

		//Init newton rapson
		theta_old.full = FIXED22_10CONST(0,0);

		return 0;
}


void get_theta_dot(FIXED22_10 *t)	{

		FIXED22_10 theta_dot;

		read_gyro(&raw_gyro_data);

		theta_dot.full = (int)raw_gyro_data[0];
		theta_dot.full = theta_dot.full - 1986; //1986 mean value of gyro steady state
		theta_dot.full = theta_dot.full<<DEC_BITS;
		theta_dot.full = MULT22_10(theta_dot,DEG_TO_RAD);
		//printf("gyro:%d\n", theta_dot.full);

		t->full = -theta_dot.full;

}

//stop the motors
void stop_motor_driver()	{
		unsigned char a[2];
		a[0] = 0;
		a[1] = 0;
		set_motor_driver(a);
}

//set motor output
void set_u_value(FIXED22_10 u[])	{

		FIXED22_10 u_temp[2];
		unsigned char u_out[2];

		u_temp[0].full = ZERO_POINT_M1.full + MULT22_10(SCALE_U_TO_MOTOR,u[0]);
		u_temp[1].full = ZERO_POINT_M2.full + MULT22_10(SCALE_U_TO_MOTOR,u[1]);
		u_out[0] = u_temp[0].full>>DEC_BITS;
		u_out[1] = u_temp[1].full>>DEC_BITS;

		//printf("PID control signal u[0]=%d u[1]=%d\n", u_out[0], u_out[1]);

		set_motor_driver(u_out);
}

//send communication data
void send_out_data(unsigned char data[]) {
		send_com_data(data);
}

//get communication data
void get_in_data(unsigned char *data)	{
```

```
            get_com_data(data);
}

//set led high
void led_high_red()        {
            led_on_red();
}

//set led low
void led_low_red()         {
            led_off_red();
}

//set led high
void led_high_yellow()     {
            led_on_yellow();
}

//set led low
void led_low_yellow()      {
            led_off_yellow();
}


//Function for calculate fraction of numeric estimation for sine and cosine; one fraction like
x^mult/div
//calculate the (in^mult)/div.
FIXED22_10 mult_div(FIXED22_10 *in, int mult, int div){

            FIXED22_10 temp;
    FIXED22_10 ret;
    int i;

    ret.full = in->full;
    temp.full = in->full;
    for(i = 1; i < mult; i++){
        ret.full = MULT22_10(ret,temp);
    }
    temp.full = FIXED22_10CONST(div,0);
    ret.full = DIV22_10(ret,temp);
    return ret;
}

//Series expansion of cosine.
FIXED22_10 cos_num(FIXED22_10 *x)        {

            FIXED22_10 temp;
    FIXED22_10 ret;

    ret.full = 1<<16;
    temp = mult_div(x,2,2);
    ret.full = ret.full - temp.full;
    temp = mult_div(x,4,24);
    ret.full = ret.full + temp.full;
    temp = mult_div(x,6,720);
    ret.full = ret.full - temp.full;
    return ret;
}

//Series expansion of sine.
FIXED22_10 sin_num(FIXED22_10 *x){

            FIXED22_10 temp;
    FIXED22_10 ret;

    ret.full = x->full;;
    temp = mult_div(x,3,6);
    ret.full = ret.full - temp.full;
    temp = mult_div(x,5,120);
    ret.full = ret.full + temp.full;
    temp = mult_div(x,7,5040);
    ret.full = ret.full - temp.full;
    return ret;

}

//Series expansion of arcsine.
```

```
FIXED22_10 asin_num(FIXED22_10 *x){

            FIXED22_10 temp;
    FIXED22_10 ret;

    ret.full = x->full;
    temp = mult_div(x,3,6);
    ret.full = ret.full + temp.full;
    temp = mult_div(x,5,40);
    temp.full = MULT22_10(asin_const,temp);
    ret.full = ret.full + temp.full;
    return ret;

}

//converts raw accelerometer values to fixpoint
void conv_xy_raw_to_g(FIXED22_10 *raw_data)           {

            FIXED22_10 temp;

            temp.full = raw_data->full - zeropoint.full;
            temp.full = DIV22_10(temp,lsb_g);
            raw_data->full = temp.full;

}

//Numerical method to calculate new theta values
FIXED22_10 newton_raps(const FIXED22_10 sx,const FIXED22_10 sy, FIXED22_10 *kalmest){

                        FIXED22_10 fx;
                        FIXED22_10 tmp;
                        FIXED22_10 theta;
                        FIXED22_10 theta_1;
                        FIXED22_10 theta_2;

                        FIXED22_10 derr;

                        theta.full = kalmest->full;

                        int i = 0;

                        for(i = 0;i < 20; i++)     {

                        theta_1.full = theta.full + THETA_OFFSET.full;
                        theta_2.full = theta.full - THETA_OFFSET.full;

                        sin_theta = sin_num(&theta_1);
                                    cos_theta = cos_num(&theta_2);


                        //Calculate fx for the old theta

                        tmp.full = MULT22_10(sx,sin_theta);
                        fx.full = FX_INT.full - tmp.full;
                        tmp.full = MULT22_10(sy,cos_theta);
                        fx.full = fx.full + tmp.full;


                        //printf("sx, cos_theta: %d : %d\n", sx.full, cos_theta.full);
                        derr.full = MULT22_10(sx,cos_theta);
                        //printf("tmp1: %d\n", derr.full);

            //          printf("sy, sin_theta: %d : %d\n", sy.full, sin_theta.full);
                        tmp.full = MULT22_10(sy,sin_theta);
                        //printf("tmp2: %d\n", tmp.full);
                        derr.full = -tmp.full - derr.full;
                        //printf("derr: %d\n", derr.full);
                        derr.full = MULT22_10(derr,NEWTON_K);
                        //printf("derr*k: %d\n", derr.full);

                        if ((derr.full) < MIN_DERR.full && (derr.full) > -MIN_DERR.full)
            {
                                    if (derr.full <= 0)        {
                                                derr.full = -MIN_DERR.full;
                                    //          printf("derr - : %d\n", derr.full);
                                    }
                                    else          {
```

```
                                        derr.full = MIN_DERR.full;
                                //            printf("derr + : %d\n", derr.full);
                                }
                                }

                        if ((fx.full) < MIN_FX.full && (fx.full) > -MIN_FX.full)
                {
                                        break;
                                }
                //            printf("derr: %d\n", derr.full);
                //            printf("fx: %d\n", fx.full);
                //            printf("theta: %d\n", theta.full);

                        theta.full = theta.full -  DIV22_10(fx,derr);
                //            printf("theta: %d\n", theta.full);
                    }

                    if((theta.full >= MAX_ANGLE.full) || (theta.full <= -MAX_ANGLE.full))
                {
                        //printf("error in newton rapson: %d\n", theta_new.full);
                        theta.full = kalmest->full;
                    }

                    //printf("fx %d : %d\n", fx.full, i);

                    return theta;
}

//get new theta value
void get_theta(FIXED22_10 t[], FIXED22_10 *kalmest)   {

                FIXED22_10 theta;

                read_acc_xy(&raw_acc_data);

                sx.full = (int)raw_acc_data[0];
                sy.full = (int)raw_acc_data[1];

                //printf("%d gx\n", sx.full);
                //printf("%d gy\n", sy.full);

                sx.full = sx.full<<16;
                sy.full = sy.full<<16;

                conv_xy_raw_to_g(&sx);
                conv_xy_raw_to_g(&sy);

                if((sx.full > 100000) || (sx.full < -100000))         {
                        sx.full = 0;
                }

                theta = asin_num(&sx);
                //printf("%d theta\n", theta.full);
                t[0].full = -theta.full;
                //theta = newton_raps(&sx,&sy,kalmest);
                theta = newton_raps(sx,sy,kalmest);
                t[1].full = -theta.full;
                t[2].full = sx.full;
                t[3].full = sy.full;
}
```

```
/*
 * fixpoint.h
 *
 *  Created on: Mar 2, 2009
 *      Author: blackfin
 */


#ifndef FIXPOINT_H_
#define FIXPOINT_H_

#include <linux/types.h>

//typedef union FIXED22_10tag {
//    int full;
//    struct part22_10tag {
//    int fraction: 10;
//    int integer: 22;
//    } part;
//} FIXED22_10;

typedef union FIXED22_10tag {
             int full;
    struct part22_10tag {
    int fraction: 16;
    int integer: 16;
    } part;
} FIXED22_10;

//#define FIXED22_10CONST(A,B) (int)((A<<10) + ((B + 0.001953125)*1024))
#define FIXED22_10CONST(A,B) (int)((A<<16) + ((B + 0.00003051757813)*65536))

static const int DEC_BITS = 16;

int MULT22_10(FIXED22_10 a, FIXED22_10 b);
int DIV22_10(FIXED22_10 nom, FIXED22_10 denom);
void print_text_fix(char text[], FIXED22_10 *temp);
void print_fix(FIXED22_10 *temp);

#endif /* FIXPOINT_H_ */
```

```
/ * fixpoint.c
 *
 *  Created on: Mar 5, 2009
 *      Author: blackfin
 *
 *      fixpoint.c is used for fixed point arithmetics.
 *      Multiplications and division functions
 */

#include "fixpoint.h"

//multipy two fixpoint numbers
int MULT22_10(FIXED22_10 a, FIXED22_10 b)             {

              int temp,result,dec;
    int sign;
    __u64 temp1;
    __u64 temp2;

    sign = 1;
    if (a.full < 0)
    {
        sign *= -1;
        a.full = -a.full;
    }
    if (b.full < 0)
    {
        sign *= -1;
        b.full = -b.full;
    }

    temp1 = (__u64)a.full;
    temp1 = temp1 * b.full;
    temp1 = temp1>>16;

    result = (int)temp1;
    result = result * sign;
    return (result);
}

//divide two fixpoint numbers
int DIV22_10(FIXED22_10 nom, FIXED22_10 denom)        {
              int dec;
              int temp;
              int answer;
              int sign;
              __u64 temp1;
              __u64 temp2;

              sign = 1;
              if (nom.full<0)            {
                            nom.full = -nom.full;
                            sign = -sign;
              }
              if (denom.full<0)          {
                            denom.full = -denom.full;
                            sign = -sign;
              }

              temp1 = (__u64)(nom.full & 0xFFFF0000);
              temp1 = temp1<<16;
              temp1 = temp1/denom.full;

              temp2 = (__u64)(nom.full & 0x0000FFFF);
              temp2 = temp2<<16;
              temp2 = temp2/denom.full;

              temp1 = temp1 + temp2;
              answer = (int)temp1;
              answer *= sign;

              return answer;
}

//print fixpoint
void print_text_fix(char text[], FIXED22_10 *temp)    {
               printf("%s %d. %2.2d\n",  text, temp->part.integer,
```

```
                    (temp->part.fraction*100+32768)/65536);
}

//print fixpoint
void print_fix(FIXED22_10 *temp)         {
               printf("%d.%2.2d\n", temp->part.integer,
                      (temp->part.fraction*100+512)/1024);
}
```

```c
/*
 * interface_driver.h
 *
 *  Created on: Feb 26, 2009
 *      Author: blackfin
 */

#ifndef INTERFACE_DRIVER_H_
#define INTERFACE_DRIVER_H_

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

#include <sys/ioctl.h>
#include <native/task.h>
#include <native/mutex.h>
#include <native/timer.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>


int init_interface_driver();
int end_interface_driver();

void read_acc_xy(__u16 acc_data[]);
void read_gyro(__u16 gyro_data[]);

void set_motor_driver(unsigned char data[]);
void led_on_red();
void led_on_yellow();
void led_off_red();
void led_off_yellow();

void send_com_data(unsigned char data[]);
void get_com_data(unsigned char *data);

void init_write_log_data();
void init_read_log_data();
void close_log_data();
void write_log_data(char *data);
int read_log_data(char *data);


#endif /* INTERFACE_DRIVER_H_ */
```

```
/*
 * interface_driver.c
 *
 *  Created on: Feb 26, 2009
 *      Author: blackfin
 *
 *      interface_driver.c opens all devices and handles all communication to them.
 */

#include "interface_driver.h"

//static variables
static const int DEBUG = 0;

static const __u16 ACC_CONF_BITS[2] = {0x8c00, 0x8400};
static const __u16 GYRO_CONF_BITS[1] = {0x8310};

static const RTIME END_DELAY = 2000000000u;
static const RTIME UPDATE_SENSOR_DELAY = 1000000u; //nanosec

static const int LED_IO_PRIO = 2;
static const int SENSOR_PRIO = 30;
static const int MOTOR_PRIO = 30;

//device variables
int acc_dev;                                            //accelerometer
int gyro_dev;                                           //gyroscope
FILE *motor_dev;                                                    //motor driver
int led_red_dev;                                        //led red
int led_yellow_dev;                                     //led yellow
FILE *com_dev;                                                      //communication dev, (com
port/bluetooth)
FILE *data_log;                                                     //data log file

//data buffers
__u16 acc_sample[2];
__u16 gyro_sample;
unsigned char motor_out[2];

//mutex for reading/sending data
RT_MUTEX acc_sample_mutex;
RT_MUTEX gyro_sample_mutex;
RT_MUTEX motor_out_mutex;

//loop variables
int end_interface_tasks = 0;
int do_logging_data = 0;
int do_read_logging_data = 0;

//real-time tasks
RT_TASK update_acc_task;
RT_TASK update_gyro_task;
RT_TASK update_motor_task;

RT_TASK rt_led_on_red_task;
RT_TASK rt_led_off_red_task;
RT_TASK rt_led_on_yellow_task;
RT_TASK rt_led_off_yellow_task;

//get the latest accelerometer value
void read_acc_xy(__u16 acc_data[])      {
             rt_mutex_acquire(&acc_sample_mutex, TM_INFINITE);
             acc_data[0] = acc_sample[0];
             acc_data[1] = acc_sample[1];
             rt_mutex_release(&acc_sample_mutex);
}

//get the latest accelerometer value
void read_gyro(__u16 gyro_data[])       {
             rt_mutex_acquire(&gyro_sample_mutex, TM_INFINITE);
             gyro_data[0] = gyro_sample;
             rt_mutex_release(&gyro_sample_mutex);
}

//turn on led
void led_on_red_task()     {
             RT_TASK_INFO info;
```

```
                rt_task_inquire(rt_task_self(), &info);

                //printf("message from %s\n", info.name);
                write(led_red_dev, "O1",2);
}

//turn on led
void led_on_yellow_task()  {
                RT_TASK_INFO info;
                rt_task_inquire(rt_task_self(), &info);

                //printf("message from %s\n", info.name);
                write(led_yellow_dev, "O1",2);
}

//turn off led
void led_off_red_task()    {
                RT_TASK_INFO info;
                rt_task_inquire(rt_task_self(), &info);

                //printf("message from %s\n", info.name);
                write(led_red_dev, "O0",2);
}

//turn off led
void led_off_yellow_task() {
                RT_TASK_INFO info;
                rt_task_inquire(rt_task_self(), &info);

                //printf("message from %s\n", info.name);
                write(led_yellow_dev, "O0", 2);
}

//spawn led task
void led_on_red()          {
                rt_task_spawn(&rt_led_on_red_task, "led_on_red_task", 0, LED_IO_PRIO, 0,
&led_on_red_task, NULL);
}

//spawn led task
void led_on_yellow()       {
                rt_task_spawn(&rt_led_on_yellow_task, "led_on_yellow_task", 0, LED_IO_PRIO, 0,
&led_on_yellow_task, NULL);
}

//spawn led task
void led_off_red()         {
                rt_task_spawn(&rt_led_off_red_task, "led_off_red_task", 0, LED_IO_PRIO, 0,
&led_off_red_task, NULL);
}

//spawn led task
void led_off_yellow()      {
                rt_task_spawn(&rt_led_off_yellow_task, "led_off_yellow_task", 0, LED_IO_PRIO, 0,
&led_off_yellow_task, NULL);
}

//write data to communication device
void send_com_data(unsigned char data[]) {
                unsigned int len = data[1] + 2;
                fwrite(data,len,1,com_dev);
}

//read data from communication device
void get_com_data(unsigned char *data)   {
                fread(data, 1, 1, com_dev);
}

//init the log write
void init_write_log_data() {
                if(!do_logging_data || !do_read_logging_data )         {
                                data_log = fopen("/tmp/data_log", "w");
                                if (data_log < 0) {
                                                perror("open /tmp/data_log");
                                }
                                else          {
                                                do_logging_data  = 1;
```

```
                        }
                }
                else            {
                        close_log_data();
                        data_log = fopen("/tmp/data_log", "w");
                        if (data_log < 0) {
                                perror("open /tmp/data_log");
                        }
                        else            {
                                do_logging_data  = 1;
                        }
                }
}

//close the log writer
void close_log_data()       {
                if(do_logging_data || do_read_logging_data)           {
                        fclose(data_log);
                }
                do_logging_data  = 0;
                do_read_logging_data = 0;
}

// write to the log file
void write_log_data(char data[])        {
                if(do_logging_data)        {
                        fprintf(data_log,"%s\n", data);
                }
}

//init read from the log file
void init_read_log_data()  {
                if(!do_logging_data || !do_read_logging_data )        {
                        data_log = fopen("/tmp/data_log", "r");
                        if (data_log < 0) {
                                perror("open /tmp/data_log");
                        }
                        else            {
                                do_read_logging_data  = 1;
                        }
                }
                else            {
                        close_log_data();
                        data_log = fopen("/tmp/data_log", "r");
                        if (data_log < 0) {
                                perror("open /tmp/data_log");
                        }
                        else            {
                                do_read_logging_data = 1;
                        }
                }
}

//read from the log file
int read_log_data(char data[])          {
                if(do_read_logging_data)   {
                        if(!feof(data_log))        {
                                fgets(data, 256, data_log);
                                return feof(data_log);
                        }
                        else            {
                                return feof(data_log);
                        }
                }
                else            {
                        return 1;
                }
}

//sample the accelerometer
void update_acc_sensor_values()         {

                RT_TASK_INFO info;
                rt_task_inquire(rt_task_self(), &info);

                struct spi_ioc_transfer    xfer[1];
                __u16 buf[2];
```

```
		int status;

		memset(xfer, 0, sizeof xfer);
		memset(buf, 0, sizeof buf);

		xfer[0].tx_buf = (__u64) ACC_CONF_BITS;
		xfer[0].len = 4;
		xfer[0].rx_buf = (__u64) buf;

		RTIME start, done;

		rt_task_set_periodic(NULL, TM_NOW, UPDATE_SENSOR_DELAY);


		while(!end_interface_tasks){
				rt_task_set_mode(0,T_PRIMARY,NULL);
				start = rt_timer_read();

				if(DEBUG)	{
						printf("message from %s\n", info.name);
				}


				status = ioctl(acc_dev, SPI_IOC_MESSAGE(1), xfer);
				if (status < 0) {
						perror("SPI_IOC_MESSAGE");
						return;
				}

				rt_mutex_acquire(&acc_sample_mutex, TM_INFINITE);
				acc_sample[0] = buf[0];
				acc_sample[1] = buf[1];

				if(DEBUG)	{
						printf("update_acc_sensor_values: ");
						printf(" %04x", acc_sample[0]);
						printf(" %04x", acc_sample[1]);
						printf("\n");
				}

				rt_mutex_release(&acc_sample_mutex);

				done = rt_timer_read();
				//rt_task_sleep(UPDATE_SENSOR_DELAY - (done - start));
				rt_task_wait_period(NULL);
		}

}

//sample the gyroscope
void update_gyro_sensor_values()	{

		RT_TASK_INFO info;
		rt_task_inquire(rt_task_self(), &info);


		struct spi_ioc_transfer	xfer[1];
		__u16 buf[2];
		int status;

		memset(xfer, 0, sizeof xfer);
		memset(buf, 0, sizeof buf);

		xfer[0].tx_buf = (__u64) GYRO_CONF_BITS;
		xfer[0].len = 2;
		xfer[0].rx_buf = (__u64) buf;


		RTIME start, done;

		rt_task_set_periodic(NULL, TM_NOW, UPDATE_SENSOR_DELAY);

		while(!end_interface_tasks){
				rt_task_set_mode(0,T_PRIMARY,NULL);
				start = rt_timer_read();

				if(DEBUG)	{
```

```
                                printf("message from %s\n", info.name);
                        }

                        status = ioctl(gyro_dev, SPI_IOC_MESSAGE(1), xfer);

                        if (status < 0) {
                                perror("SPI_IOC_MESSAGE");
                                return;
                        }

                        rt_mutex_acquire(&motor_out_mutex, TM_INFINITE);
                        gyro_sample = buf[0];

                        if(DEBUG)     {
                                printf("update_gyro_sensor_values: ");
                                printf(" %04x", gyro_sample);
                                printf("\n");
                        }

                        rt_mutex_release(&motor_out_mutex);

                        done = rt_timer_read();
                        //rt_task_sleep(UPDATE_SENSOR_DELAY - (done - start));
                        rt_task_wait_period(NULL);
                }
}

//send motor data task
void send_motor_data_task(){
                RT_TASK_INFO info;
                rt_task_inquire(rt_task_self(), &info);
                if(DEBUG)     {
                        printf("message from %s\n", info.name);
                }
                unsigned char motor_out_tmp[2];

                while(!end_interface_tasks){
                        rt_task_suspend(&update_motor_task);
                        rt_mutex_acquire(&motor_out_mutex, TM_INFINITE);
                        motor_out_tmp[0] = motor_out[0];
                        motor_out_tmp[1] = motor_out[1];
                        rt_mutex_release(&motor_out_mutex);

                        fputc(motor_out_tmp[0],motor_dev);
                        fputc(motor_out_tmp[1],motor_dev);
                        fflush(motor_dev);

                        if(DEBUG)     {
                                printf("send_motor_data_task: ");
                                printf(" %d", motor_out_tmp[0]);
                                printf(" %d", motor_out_tmp[1]);
                                printf("\n");
                        }
                }

}

//set motor output
void set_motor_driver(unsigned char data[])           {
                rt_mutex_acquire(&motor_out_mutex, TM_INFINITE);
                motor_out[0] = data[0];
                motor_out[1] = data[1];

                rt_mutex_release(&motor_out_mutex);
                rt_task_resume(&update_motor_task);
}


//init the motor driver, open all devices
int init_interface_driver(){
                motor_out[0] = 0;
                motor_out[1] = 0;
                acc_dev = open("/dev/spidev0.1", O_RDWR);
                if (acc_dev < 0) {
                        perror("open spidev0.1");
                        return 1;
                }
```

```
		gyro_dev = open("/dev/spidev0.5", O_RDWR);
		if (gyro_dev < 0) {
			perror("open spidev0.5");
			return 1;
		}

		motor_dev = fopen("/dev/ttySS1","w");
		if (motor_dev < 0) {
			perror("open ttySS1");
			return 1;
		}

		led_red_dev = open("/dev/gpio21", O_WRONLY);
		if (led_red_dev < 0) {
			perror("open gpio21");
			return 1;
		}

		led_yellow_dev = open("/dev/gpio20", O_WRONLY);
		if (led_yellow_dev < 0) {
			perror("open gpio20");
			return 1;
		}

		com_dev = fopen("/dev/ttyBF1", "r+");
		if (com_dev < 0) {
			perror("open ttyBF1");
			return 1;
		}

		//create mutex for data get/set loops
		int mux_err = rt_mutex_create(&acc_sample_mutex, "ACC_SAMPLE_MUTEX");
		if (mux_err < 0) {
			perror("mutex ACC_SAMPLE_MUTEX");
			return 1;
		}

		mux_err = rt_mutex_create(&gyro_sample_mutex, "GYRO_SAMPLE_MUTEX");
		if (mux_err < 0) {
			perror("error GYRO_SAMPLE_MUTEX");
			return 1;
		}
		mux_err = rt_mutex_create(&motor_out_mutex, "MOTOR_OUT_MUTEX");
		if (mux_err < 0) {
			perror("error MOTOR_OUT_MUTEX");
			return 1;
		}

		//spwan tasks
		rt_task_spawn(&update_acc_task, "update_acc_task", 0, SENSOR_PRIO, 0,
&update_acc_sensor_values, NULL);
		rt_task_spawn(&update_gyro_task, "update_gyro_task", 0, SENSOR_PRIO, 0,
&update_gyro_sensor_values, NULL); //disabled due to chip_select problem
		rt_task_spawn(&update_motor_task, "update_motor_task", 0, MOTOR_PRIO, 0,
&send_motor_data_task, NULL);
		return 0;
}

//end the interface driver, close all devices
int end_interface_driver() {
		end_interface_tasks = 1;

		//sleep 1 sec
		rt_task_sleep(END_DELAY);

		//close devs
		close(acc_dev);
		close(gyro_dev);
		fclose(motor_dev);
		close(led_red_dev);
		close(led_yellow_dev);
		fclose(com_dev);
		close_log_data();

		//delete rt_tasks
		rt_task_delete(&update_acc_task);
```

```
    rt_task_delete(&update_gyro_task);
    rt_task_delete(&update_motor_task);

    rt_task_delete(&rt_led_on_red_task);
    rt_task_delete(&rt_led_off_red_task);
    rt_task_delete(&rt_led_on_yellow_task);
    rt_task_delete(&rt_led_off_yellow_task);

    //delete mutex
    rt_mutex_delete(&acc_sample_mutex);
    rt_mutex_delete(&gyro_sample_mutex);
    rt_mutex_delete(&motor_out_mutex);

    return 0;
}
```