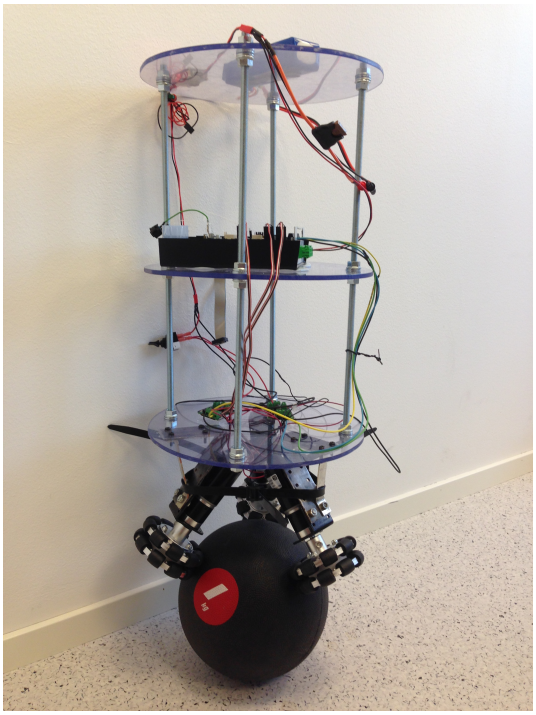
CHALMERS



Development of a mechatronical platform for AUTOSAR

The ball-balancing robot

Master of Science Thesis

ANDRÉ ALSTRIN
EMIL SUNDELL

Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015

EX016/2015

Development of a mechatronical platform for AUTOSAR

The ball-balancing robot

ANDRÉ ALSTRIN
EMIL SUNDELL

Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015

Development of a mechatronical platform for AUTOSAR
The ball-balancing robot
ANDRÉ ALSTRIN
EMIL SUNDELL

Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone +46 (0)31–772 1000

Cover:
A picture of the constructed mechatronical system.

Chalmers Bibliotek, Reproservice
Göteborg, Sweden 2015

Development of a mechatronical platform for AUTOSAR
The ball-balancing robot

ANDRÉ ALSTRIN
EMIL SUNDELL
Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
Chalmers University of Technology

Abstract

The goal of this project was to develop and construct a mechatronical platform that uses the software architecture AUTOSAR and allows the user to use generated software components created in a model based development tool. The platform itself has been developed in the form of a robot that balances on a ball. An extensive model of the robot and a Linear-Quadratic-Regulator have been developed in order to control it. The system has been simulated in Simulink and the controller has been code generated and incorporated in the AUTOSAR environment. The robot has been built from scratch as well.

Setting up and configuring AUTOSAR on the platform proved to be quite a challenge. Since AUTOSAR is very uncommon outside of the automotive industry the documentation about how to implement it is very limited. Once the platfrom was properly configured, the control algorithms from Simulink were integrated and run on the ECU. The concept works perfectly. However, due to lack of precision and some poor choices of hardware components, the robot is not able to apply the small torques required close to the equilibrium point. This makes it at the moment not possible for the robot to balance for more than a few seconds.

Index Terms: Control theory, embedded systems, code generation,
Ballbot,SIMULINK, AUTOSAR

Acknowledgements

This report documents the design process, construction and evaluation of a mechatronical system, in the form of a robot that can balance on a ball, with the purpose to serve as a teaching platform for the software architecture AUTOSAR. The project is carried out as a master thesis at the Division of Automatic control, Automation and Mechatronics at Chalmers University of Technology. The development of the system has taken place at the company QRTECH AB, Mölndal.

We would like to thank QRTECH for giving us the opportunity to do this project and also for all the technical and economical support they gave us.

We especially want to thank our supervisor at QRTECH, **Andreas Käck**, for his invaluable help and expertise together with great dedication to this project.

Furthermore, we would like to thank **Joakim Plate** and **Peter Dahlin** at QRTECH for their great knowledge and help concerning AUTOSAR, programming and PCB design. We also appreciate all the advice and ideas **Christian Ekman** and **Carl Petersson** shared with us.

Last but not least we want to thank our examiner at Chalmers, **Jonas Fredriksson**, for his support and feedback concerning the planning and thesis report.

André & Emil
Göteborg, Sweden, 2015

Contents

Chapter 1

Introduction

1.1 Background

In the beginning of the 21st century the number of computer controlled functions in cars started to become very large. Typically each major function, such as automatic transmission and traction control, used separate Electronic Control Units (ECUs). The hardware and the software were tightly connected which made it difficult to re-use or move the software to another hardware platform. More and more automotive manufacturers realised the problem and decided to do something about it. This cooperation resulted in the creation of the AUTomotive Open System ARchitecture (AUTOSAR).

Today, most automobile manufacturers, suppliers and tool developers use and jointly develop AUTOSAR. As a result, it has become important to establish and maintain competence of AUTOSAR within companies related to automotive industry. A possible method to train employees in the matter is through internal courses. In order for such a course to be stimulating and exciting the scope could be moved from pen and paper and pure theoretics to working with an actual hardware system.

1.2 Aim

This project aims at developing a system that can be used as a teaching platform for AUTOSAR. The shape of the system should be that of a so called *Ballbot* i.e. a dynamically stable robot balancing on a single spherical body, i.e. a ball. The system should make use of a computing platform developed at the company as well as the software architecture AUTOSAR. It should be able to serve as an evaluation platform for different AUTOSAR software applications, e.g. a control algorithm developed in MATLAB/Simulink using code generation.

The system is then to be used as a part of internal courses in AUTOSAR at a company as well as for demonstration purposes, e.g. at a business fair. As for the project's

contribution to the field of study, AUTOSAR is not a common software architecture outside of the automotive industry, to the best of the authors' knowledge. Hence it could serve as an evaluation of the suitability of using such an architecture within other disciplines.

1.3 Goal

The main goal is to construct and develop a fully working and running system. This means that the system should run on the architecture AUTOSAR with the possibility to upload and run programs and algorithms on it developed in Simulink. The system should also be able to read data from various sensors connected to it and respond by sending out control signals to its actuators in an appropriate way.

A fully working control method for stabilising a Ballbot that works at least in simulations should be developed as well. These simulations shall be used when determining parameters and components in the hardware design.

A secondary goal, depending on the time at hand, is to make the constructed robot balance on a ball without support. Specifically, the system should be able to balance without assistance and become stable when it is perturbed from its equilibrium point. The system should also manage to return to its starting position when moved from this point. Furthermore, the system should handle disturbances such as light pushes and shoves without falling as well as carrying an additional load which alters the centre of gravity, inertia and other model specific parameters.

1.4 Problem

Due to the nature of the project, the main scientific challenge is that of system design. Through extensive research and motivated by current engineering methods decisions about different aspects of the final system is to be made. These decisions are mainly dictated by the desired functionality of the final system. The system design can be divided into a software and a hardware part.

The software design of the system involves development of a suitable control algorithm and implementing it using the software architecture AUTOSAR. Hence, the software part could be seen as related to two research questions, that of control theory (modelling, control design, robustness analysis) and software development (structuring, efficiency and conformity to the industry standard).

As for the hardware design, it involves construction of chassis, choice of actuators and sensors as well as the electronics surrounding these parts to name a few points. A small form factor computer-on-module developed at the company using Smart Mobility ARChitecture (SMARC) will be the central computing platform.

1.5 Scope

The AUTSOSAR system should work in the sense that the system is to be able to run. It shall also be able to perform calculations on data gathered by different sensors as well as controlling actuators. However it is not in the scope of this thesis to make the system and software follow all the standards and requirements that are present in the automotive industry.

The main objective for the control algorithm is to stabilise the robot i.e. keeping it from falling. Navigation with respect to the environment, remote controlling etc. will not be considered unless time permits.

Disturbances are considered to lie within a reasonable value such that it can be included into the modelling error. This reduces the sensors to those only related to the stabilisation, i.e. force sensors measuring the increase in mass when the robot is loaded is not considered.

The robot is assumed to act in a standard environment i.e. robustness and other performance parameters are not investigated in undulated terrain.

1.6 Method

To have a detailed understanding of the problems to be solved within the project, an extensive literature study is to be made regarding existing systems of the same type and other relevant subjects in connection to the project. Specifications of functionality for the finished product is then listed.

The whole system is then to be designed on a block level, i.e. decide which parts that are needed and the requirements they need to fulfil. The work is later on divided into two paths; hardware and software construction. Before any placements of equipment orders, the system is modelled and simulated together with the control algorithm in MATLAB/Simulink in order to validate that the proposed design is possible to implement, and if not, the system design is altered.

Given that the simulations are successful, the system is constructed and tested iteratively. Tasks also connected to the construction are basic software configuration of the hardware platform as well as setting up the code generation from Simulink to the AUTOSAR target.

When the whole system is completed it is to be evaluated by comparison between simulations from a model of the system and real-world measurements. The work and result is to be documented in a master thesis report as well as presented at the completion of the project.

1.7 Previous work

In 2005 Prof. Ralph Hollis at Carnegie Mellon University, Pittsburg USA, developed the first working *Ballbot*, which later on in 2010 also was patented [1]. This *Ballbot* developed by Hollis and his group was designed to be of human size and able to withstand disturbances such as light kicks and shoves, as well as collisions with obstacles such as walls [2]. Hollis describes the driving mechanism as an inverted mouse-ball drive. Instead of letting the ball's movements provide computer signals through the mouse's rollers, the idea is to send computer signals to the roller in order to control the ball. To achieve this the *Ballbot* is equipped with four steel rollers placed orthogonal to each other around the centre of the ball, with actuators connected to two of them.

Despite the fact that the Ballbot is such a relatively recent invention several other groups around the world have developed similar balancing robots of their own. One example is the *BallIP* (Ball Inverted Pendulum) developed by Prof. Masaaki Kumagai at Tohoku Gakuin University, Japan, in 2008 [3]. *BallIP* is much smaller than the *Ballbot* and instead of using steel rollers to balance it uses three omnidirectional wheels that all are connected to actuators. This makes it possible for the *BallIP* to pivot around its vertical axis such that an arbitrary heading can be specified [3], in contrast to the original *Ballbot*.

Even though some functional Ballbots obviously exists around the world, none of them are built upon the software architecture AUTOSAR. Thus this is could be the first project that tries to combine these two parts.

1.8 Outline of thesis

Essential background theory for the thesis is covered briefly in Chapter 2. Methods of modelling and control is dealt with as well as more project specific theory regarding AUTOSAR. In addition, concepts used in the hardware design are also presented.

Next, the modelling and simulations of the system is presented in Chapter 3. A mathematical model of the system is derived and different simulation scenarios of the system's behaviour in Simulink are shown.

Chapter 4 then describes the system design in detail, covering both software and hardware design as well as the construction of the robot.

The results achieved in the project is then presented and evaluated in Chapter 5. Following in Chapter 6, is a discussion of the results which covers both successful parts and possible further improvements that could have been done. At last, a conclusion regarding the project as a whole is drawn in Chapter 7.

Chapter 2

Theory

This chapter deals with theoretical background regarding modelling and control theory as well as software architecture and electronics. Since most of the subjects are vast topics which can not by any means be presented fully in this thesis, the focus lies on results and implementation techniques from the theory.

2.1 Lagrangian mechanics

Whilst the Newtonian mechanics approach is the most well-known method for modelling mechanical systems, the alternative of using the Lagrangian approach could be beneficial in many cases. The former uses relations between vectors of force, momentum and acceleration. In contrast, the latter derives the equations of motion by basic steps based on scalar quantities such as energy, work and power, regardless of the geometrical aspects of the system. Hence, as systems become more complex, the Newtonian approach involves intricate derivations of forces, reaction forces and constraints on the different bodies involved whereas the Lagrangian approach is relatively easy. In the latter case the derivations of the equations of motions is a totally analytical procedure which demands no analysis of forces in vector form. The general outline of the Lagrangian procedure is given in the list below [4].

1. **Choose a set of generalised coordinates** q_i
   A system of n masses can be fully described by $3n$ independent coordinates also denoted as degrees of freedom (DOF). In a case of a constrained system the DOF are less than $3n$. Generalised coordinates describe the configuration in all DOF relative to some reference system which can be arbitrarily chosen. Often several different reference systems are used which then are related through binding equations.

2. **Form the kinetic energy $T(\dot{q}_i)$, potential energy $U(q_i)$ and the Lagrangian $L = T(\dot{q}_i) - U(q_i)$**
   The kinetic energy, consisting of both translational and rotational kinetic energy, is expressed for each mass as a function of the time derivative of the generalised coordinates and added together. In the same way, the potential energy as a function of the generalised coordinates is expressed for each mass and added together. From the expressions of the kinetic and potential energy the Lagrangian is formed.

3. **Solve the Euler-Lagrange equations $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = F_i, \quad i = 1, ..., n$**
   The Lagrangian is substituted in the Euler-Lagrange equations and the equations are solved, resulting in a set of differential equations, each describing the dynamics of a generalised coordinate, also known as the equations of motion. The term F_i denotes generalised forces, i.e. non-potential forces acting on the system like input forces and friction.

2.2 Control Theory

Control theory can be viewed as a collection of methods from the field of mathematics and engineering for controlling systems such that they follow a desired behaviour. The idea is to give the system an appropriate input signal in order for the output to follow a given reference, despite the presence of various disturbances. This is achieved by feedback where the designed controller gets the difference between the measured output and the reference, the error signal, as input. A conceptual block diagram describing this is shown in Figure 2.1.

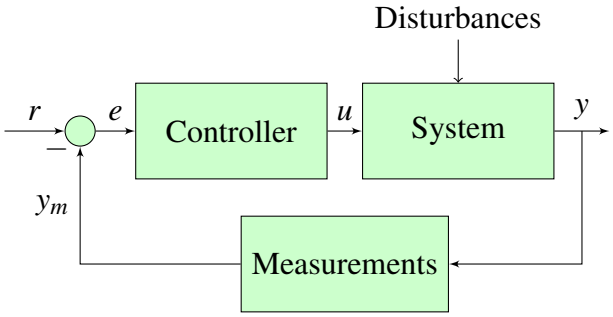


Figure 2.1: Overall control system concept.

2.2.1 Proportional-Integral-Derivative controller

The most common types of controllers are the so called Proportional-Integral-Derivative controllers (PID controllers) which are used in many industrial applications. The PID controller has historically been considered to be the most usable type of controller when there is no deeper knowledge of the underlying process to be controlled. The PID controller tries to minimise the error signal in three different aspects where it takes the present error, past error and future error into account. The P-part is proportional to the error at present time, the I-part proportional to the integral of the error up to present time and the D-part proportional to the derivative of the error up to present time [5]. A typical PID controller can be formulated as [6]

$$u(t) = K_p\Big(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\,\mathrm{d}\tau + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}\Big), \tag{2.1}$$

where

K_p = proportional gain,

T_i = integral gain,

T_d = derivative gain.

Even though in many cases PID controllers can be tuned in a manner such that the output follows the reference in a good way it is worth to note that the algorithm does not guarantee optimal control of a system or a system's stability. For large and complex systems the PID method can be very difficult to implement and tune in a satisfactory way. For cases like this an alternative control method is preferred. One example of such a method is described in section 2.2.2.

Systems are usually split into two main categories, depending on how many outputs and inputs they have [7]:

- **Single-input single-output (SISO)** - These systems have one input and one output and are the simplest ones. They are usually controlled with the help of a PID controller.
- **Multiple-input multiple-output (MIMO)** - These systems have several inputs and outputs and are generally much more complex. A MIMO system is beneficially represented as a state space model and the control method is usually some kind of state feedback/optimal control.

2.2.2 Linear-Quadratic-Gaussian controller

Linear-Quadratic-Gaussian (LQG) control is in the field of optimal control one of the most elementary problems. LQG handles the case with linear systems disturbed by additive white Gaussian noise, commonly without information about all states, i.e. some states cannot be measured and therefore have to be estimated. An LQG controller consists of a combination between two major parts; a feedback controller, namely a *Linear-Quadratic-Regulator*, for regulation of the plant and a *Kalman filter* for estimation and filtration of noisy states [8]. These two parts are going to be described further in the following subsections.

Linear-Quadratic-Regulator

A Linear-Quadratic-Regulator (LQR) is a state feedback controller and therefore has the capacity to handle large and complex MIMO systems. Instead of manually choosing the feedback gains such that the poles of the closed loop system ends up in desired locations (which is a common approach for other types of feedback controllers) the LQR method uses a different approach. The idea is to optimise a cost function. The infinite horizon LQR problem can be described as follows [8]:

Given a continuous-time linear system in state space form

$$\begin{aligned} \dot{x} &= Ax(t) + Bu(t), \\ y &= Cx(t) + Du(t), \\ x &\in \mathbb{R}^n, u \in \mathbb{R}^p \end{aligned}$$

where x is a vector containing the states, u is a vector of input signals and y a vector of outputs, the idea is to minimise the quadratic cost function

$$J = \int_0^\infty \left(x^T Q_x x + u^T Q_u u\right) \mathrm{dt}, \tag{2.2}$$

with respect to the system dynamics. $Q_x \geq 0$ and $Q_u > 0$ are symmetric, positive (semi-) definite matrices on the form

$$Q_x = \begin{pmatrix} q_{x1} & & 0 \\ & \ddots & \\ 0 & & q_{xn} \end{pmatrix}, \quad Q_u = \begin{pmatrix} q_{u1} & & 0 \\ & \ddots & \\ 0 & & q_{up} \end{pmatrix}$$

These matrices are so called weight matrices. This is where the user can tune the system. For example by setting a higher weight on for instance the first state (q_{x1}) it is much more beneficial for the system to take this state as close to zero as possible. It is also possible

to decide how much the input signals should be penalised compared to the states. The control law

$$u = -Q_u^{-1} B^T P x = Kx$$

is the solution to the LQR problem. The symmetric, positive definite matrix $P \in \mathbb{R}^{nxn}$ can be found by solving the *algebraic Riccati equation* [9] :

$$PA + A^T P - PBQ_u^{-1} B^T P + Q_x = 0. \tag{2.3}$$

The control law for the corresponding **discrete-time** system can be found in the following way:

Given the discrete-time state space model:

$$\begin{aligned} x[n+1] &= A_d x[n] + B_d u[n], \\ y[n] &= C_d x[n] + D_d u[n], \end{aligned}$$

and the quadratic cost function to be minimised with respect to the system dynamics

$$J(u) = \sum_{n=1}^{\infty} \left(x[n]^T Q_x x[n] + u[n]^T Q_u u[n] \right), \tag{2.4}$$

the control law

$$u[n] = (B_d^T S B_d + Q_u)^{-1} B_d^T S A_d x[n] = K_d x[n], \tag{2.5}$$

is the solution to the discrete-time LQR problem. S is a matrix found by solving the *discrete-time algebraic Riccati equation* [10]:

$$A_d^T S A_d - S - A_d^T S B_d (B_d^T S B_d + Q_u)^{-1} B_d^T S A_d + Q_x = 0. \tag{2.6}$$

Integral action

Since the model used in the LQR does not describe the true nonlinear plant perfectly there will be some steady-state errors. These can be compensated for by introducing integral action in the LQ controller. This is achieved by augmenting the state space model with the integral states $x_I = \int_0^t (r - y) \mathrm{d}\tau$ [8]. The resulting augmented state space model becomes:

$$\begin{bmatrix} \dot{x} \\ \dot{x}_I \end{bmatrix} = \underbrace{\begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}}_{A_{tot}} \begin{bmatrix} x \\ x_I \end{bmatrix} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{B_{tot}} u + \begin{bmatrix} 0 \\ I \end{bmatrix} r$$

By minimising the cost function

$$J = \int_0^\infty \Big(\begin{bmatrix} x^T & x_I^T \end{bmatrix} \begin{bmatrix} Q_x & 0 \\ 0 & Q_I \end{bmatrix} \begin{bmatrix} x \\ x_I \end{bmatrix} + u^T Q_u u \Big) \, \mathrm{d}t, \tag{2.7}$$

subject to the augmented system's dynamics, the LQ-regulator with integral action can be calculated.

Kalman filter

Typically when using a state feedback controller not all states are available, only the measured output y disturbed by noise. In order to reconstruct the wanted information about the missing states an appropriate observer has to be designed. One common choice of observer is the *Kalman filter*. This is an optimal observer that minimises the estimation error $\tilde{x}(t) = \hat{x}(t) - x(t)$. Even in the case where all states are measured the *Kalman filter* is often used because of its ability to optimally filter away Gaussian noise.

The *Kalman filter* is based on both measurements and a model of the system's dynamics. In this way it is possible for the user to determine how much the filter should trust the model compared to the measurements. This results in a trade-off between how sensitive the user allows the system to be against measurement disturbances and how fast the observer should track changes in the states [7].

In 1961 R. E. Kálmán and R. S. Bucy presented the method in the following way (for simplicity the matrices presented are renamed to correspond with the notation used above) [11]:

Given a linear dynamical system in state space form:

$$\begin{aligned} \frac{d}{dt}x(t) &= Ax(t) + Bu(t) + Fw(t), \\ z(t) &= y(t) + v(t) = Cx(t) + v(t), \end{aligned}$$

where $z(t)$ is the observed signal, the optimal estimator has the form

$$\frac{d}{dt}\hat{x}(t) = A\hat{x}(t) + Bu(t) + L\big(y(t) - C\hat{x}(t)\big). \tag{2.8}$$

The intensities of the two white noise terms $w(t)$ and $v(t)$ are represented as $R_w(t)$ and $R_v(t)$ and the covariance matrix of the estimation error $\tilde{x}(t)$ as P(t), where P(t) satisfies:

$$\frac{d}{dt}P(t) = AP + PA^T - PC^T R_v^{-1} CP + FR_w F^T, \tag{2.9}$$

$$P(0) = E\{x(0)x^T(0)\}$$

When P converges and the system is stable, equation (2.9) corresponds to the *dual algebraic Riccati equation*:

$$0 = AP + PA^T - PC^T R_v^{-1} CP + FR_w F^T. \tag{2.10}$$

The Kalman observer gain is then given as

$$L = PC^T R_v^{-1}. \tag{2.11}$$

For the corresponding **discrete-time** case [12], given the discrete plant:

$$\begin{aligned} x[n+1] &= A_d x[n] + B_d u[n] + F_d w[n], \\ z[n] &= y[n] + v[n] = C_d x[n] + D_d u[n] + v[n]. \end{aligned}$$

The Kalman estimator is given as

$$\hat{x}[n+1|n] = A_d \hat{x}[n|n-1] + B_d u[n] + L\big(y[n] - C_d \hat{x}[n|n-1] - D_d u[n]\big), \tag{2.12}$$

where

$$L = A_d P C_d^T \left(R_v + C_d P C_d^T\right) \tag{2.13}$$

and P is the estimation error covariance matrix given by [8]:

$$P = A_d P A_d^T + F_d R_w F_d^T - A_d P C_d^T \left(R_v + C_d P C_d^T\right)^{-1} C_d P A_d^T. \tag{2.14}$$

2.3 Complementary filter

When facing the task to measure angles there are a few ways to go. A gyroscope sensor is very good at measuring quick changes of the angle. By integrating the angular velocity given by the gyroscope it is possible to get the angle as well. However, since gyroscopes have a drift problem that makes the "zero angle" drift away from its original position, only using a gyroscope for a balancing application like this would eventually make the robot fall. Therefore the drifting problem has to be handled. An accelerometer can also measure angles and does not have a drift problem like a gyroscope. Accelerometers have in fact a very stable steady state behaviour, but on the other hand they are pretty slow to follow rapid changes and are sensitive to measurement noise [13].

One simple and cheap solution is therefore to low pass filter the accelerometer value, high pass filter the gyroscope value and combine them in a so called complementary filter. The complementary filter has the form:

$$angle[n] = (1-\alpha)(angle[n-1] + gyro[n] \cdot dt) + \alpha \cdot acc[n] \tag{2.15}$$

where α is usually in the interval $0 < \alpha < 0.5$ and is a tuning parameter. A higher value of α makes the system less sensitive to the gyroscope drift, on the other hand it also gets less sensitive towards quick changes and introduces more measurement noise. Like many other control applications there is a trade-off. Generally, the notion of a complementary filter is that if one measurement is filtered by a filter with the frequency function $G(s)$, then the other measurement is filtered by its compliment $1 - G(s)$ and they are summed. The specific case of Equation 2.15 is the result of a simplification where the original filter $G(s)$ is an exponential moving average acting on the accelerometer measurements and its compliment is acting on the gyroscope measurements.

2.4 AUTOSAR

As mentioned before, the key goal with AUTOSAR is to separate the software applications from the hardware and infrastructure. This makes it possible to develop the software applications independently of which hardware that is going to be used. By doing this it is much easier to re-use software as well as moving applications between different ECUs. Furthermore every application uses a standardised AUTOSAR interface which simplifies the integration between applications developed by different companies [14]. In Figure 2.2 the basic structure of AUTOSAR is shown.

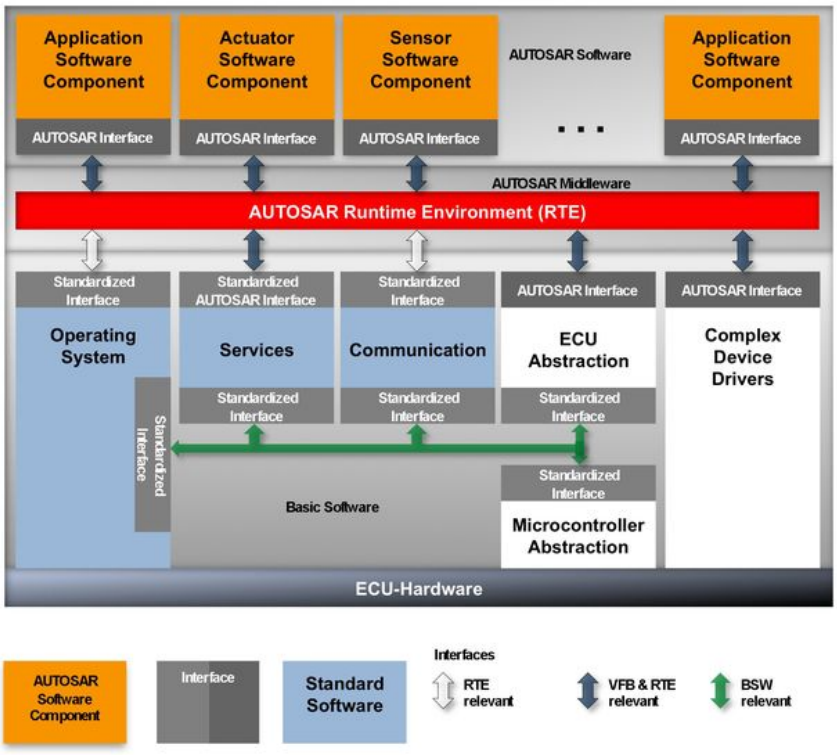


Figure 2.2: Overview of the AUTOSAR structure [15].

2.4.1 Software Component

A *Software Component* (SWC) in AUTOSAR is an application or part of an application that is designed for a particular purpose, e.g. locking a car (or controlling the movement of a Ballbot). The *Software Components* communicate with their surroundings through a standardised interface which makes it easy to re-use them on different platforms.

Inside the *Software Components* there is one or many runnables that execute specific tasks. They are triggered by different events from the *Runtime Environment* (RTE) that could be of different types. For example *Timing Events* are used when a runnable is to be executed periodically, *Operation Invoked Events* when a client calls a server for a service and *Data Received Events* when the triggering should be made by received data [16].

2.4.2 Virtual Functional Bus

In order to achieve the goal with SWCs that are independent of the underlying hardware, the *Virtual Functional Bus* (VFB) was created. It serves as an abstraction where the components can be integrated in a virtual AUTOSAR system and the communication relationship between components can be verified. The idea with this approach is to break down the complexity of a system very early in the design phase [17].

2.4.3 Runtime Environment

In the centre of the AUTOSAR ECU architecture the *Runtime Environment* (RTE) can be found. Through the RTE the VFB is realised for a specific ECU. The RTE makes the communication between different SWCs possible. Furthermore, it provides a way for the components to access the *Operating System* (OS), communication services and other *Basic Software Modules*.

For each ECU a new tailor-made RTE is generated. It is based on the ECU configuration settings, the SWCs mapped to the ECU as well as configurations and settings from basic layers. Among other things the RTE is responsible for executing tasks and running runnables [18].

2.4.4 Basic software

Below the RTE the *Basic Software* (BSW) is located and consists of several services and component modules. The BSW does not perform any functional job itself, however it provides services to the *Software Components*. The modules in the BSW are very ECU-specific and include among other things memory management, communication framework, operating system, microcontroller abstraction and ECU abstraction. It also includes support for *Complex Device Drivers*, these are drivers that not are in the AUTOSAR standard and handles specific hardware dependent features [19].

2.4.5 Operating system

The *Operating System* (OS) used in AUTOSAR is an extension of the OSEK OS, which is an industry standard. The OS is responsible for scheduling of the tasks that have runnables mapped to them by the RTE. The tasks and resources that should be mapped to the OS are set in the configuration of the OS [19].

2.5 Electronics

This section introduces some common electronic concepts within the area of mechatronic design that have been utilised throughout the project.

2.5.1 Microcontroller signal types

A microcontroller is a small specialised computer very common in embedded systems. In order to communicate with its surrounding it uses dedicated input and output pins where different types of signals are received and sent. The most common signals used are described briefly below.

Digital Input/Output Signals

These are the simplest kinds of signals that the microcontroller understands. The signals are either "low" or "high". "Low" means a voltage level close to ground potential and is interpreted as a logical zero in the processor. "High" is interpreted as a logical one and corresponds to a voltage level close to +3.3 V or +5.0 V, depending on the manufacturing technique. The measured voltage on an input pin is compared to a predefined threshold in order to determine the logical level.

Digital signals can be used for example to control a light emitting diode, reading a pulse or influence peripheral devices connected to the microcontroller in other ways.

Pulse Width Modulation

Pulse Width Modulation (PWM) is a method that can be used for controlling the amount of power delivered to a load (the user of the power), with very small losses. The idea is to periodically switch on and off the voltage in a certain pattern in order to achieve a desired average output voltage. This is done by controlling what percentage of the period the voltage is "high", the so called duty cycle [20]. Figure 2.3 shows an example of a PWM signal where the period is 4 ms and the duty cycle 75%. This means that during 75% of the period, equal to 3 ms, the output is 5 V and during the last 1 ms 0 V. Thus, the average output voltage is $5\,\text{V} \cdot 0.75 = 3.75\,\text{V}$.

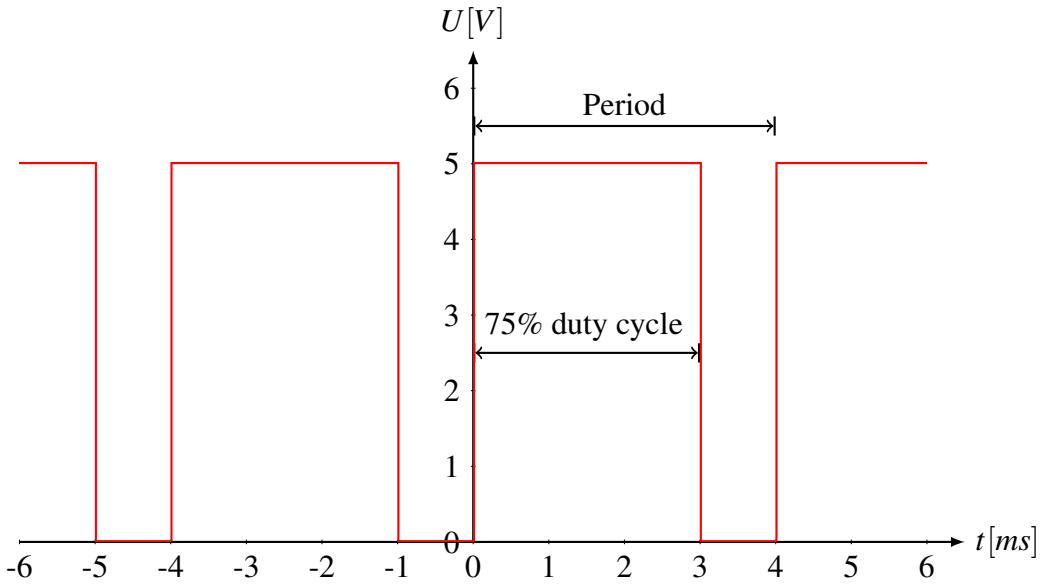


Figure 2.3: PWM signal with 75% duty cycle.

The period and duty cycle can continuously be updated with new values by a microcontroller depending on the required output at the moment. Typically the period is much shorter than the time constant of the load in order to get a smoother response. If the response still is not smooth enough one could for example lead the PWM signal through an analog filter before it reaches the load. A well designed filter more or less transforms the PWM signal into a DC signal centred around the average voltage level.

PWM is often used together with H-bridges when controlling motors. This is further described in section 2.5.2.

Analog-to-Digital Conversion

If the microcontroller should be able to use and calculate on the analog signals used by devices connected to it, the analog signals have to be converted into digital representations. This is done by an Analog-to-Digital-Converter (ADC). The two main parts of the conversion is sampling and quantisation. According to the Nyquist/Shannon theorem the signal must be sampled with at least twice the frequency of the highest frequency in the input signal, to avoid aliasing [21].

Quantisation is the process of mapping an analog value to a corresponding digital one. Due to rounding or truncation of the digital values some small errors appear between the digital values and the true analog ones, this is called quantisation error. The resolution of the ADC says how many bits that are used to represent the digital value. The higher the resolution, the more precise the mapping from an analog value to a cor-

responding digital one can be. Thus, a higher resolution results in a lower quantisation error. [21].

2.5.2 DC motor

There are many types of electronic actuators today in direct current (DC) powered applications, especially since the introduction of power electronics and inverters paved way for new types like the brushless DC motor and switched reluctance motor. These motors have many advantages such as higher torque density, increased efficiency and less mechanical wear. However, the traditional brushed DC motor is still common in many applications because of its low cost and simple control electronics [20].

Brushed DC motor model

The brushed DC motor consists of two circuits, the field windings which builds up the flux in the air-gap of the motor and the torque-producing armature windings in the rotor. Brushes are used to transfer the current to the armature circuit and always have it flowing in the right direction in order to create a continuous rotation in either direction. In many motors the flux is established by permanent magnets instead, reducing the connections, size and energy consumption. This relatively simple description of the physics behind the brushed DC motor gives a very convenient equivalent circuit, shown in Figure 2.4.

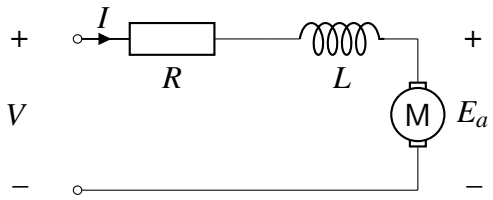


Figure 2.4: Equivalent circuit of the brushed DC motor.

In addition to the equivalent circuit the following equations hold

$$T = k_\phi I \tag{2.16}$$

$$E_a = k_\phi \omega \tag{2.17}$$

where T is the produced torque, ω the speed of the motor and E_a the counter-electromotive force, a voltage proportional to the speed. The motor constant k_ϕ is determined by the physical aspects of the motor including the strength of the earlier mentioned flux.

DC motor drivers

From Equations 2.16, 2.17 and the equivalent circuit it is evident that in order to control the motor, the variable to control is the input voltage V. In the early days of the DC motor this was often achieved by a variable resistance in series with the motor. With the introduction of power electronics, the same result can be achieved much more efficiently by the use of an H-bridge converter which is shown in Figure 2.5. By controlling the duty cycle of the transistors 1-4 with the aid of a microcontroller the voltage V_{out} can be of any value between $\pm V_{in}$.

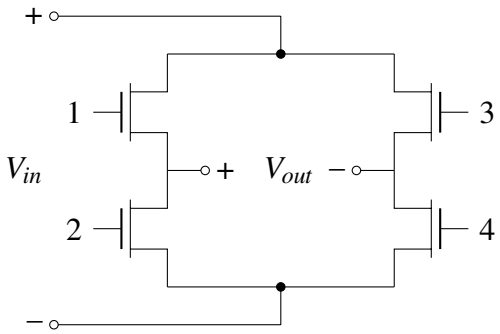


Figure 2.5: H-bridge converter circuit.

2.5.3 Current sensing

As can be seen from 2.16 the output torque of the motor is proportional to the current through the motor. Hence, in order to control the torque of the motor it is sufficient to measure the current and calculate the new input signal to the motor accordingly. The most common way to measure the current is through a shunt resistor, often at mΩ values, in order to reduce the impact of the measurement on the rest of the circuit. To have a low potential at the resistor it is often placed at the low side of the load close to circuit ground, see Figure 2.6. The voltage across the resistor is then proportional to the current in the circuit as $V_{sense} = R_{sense} \cdot i$. Since the resistance R_{sense} has such a low value, the voltage has to be amplified in order to increase the resolution. This can either be accomplished through an operational amplifier circuit or a Hall effect sensor [22].

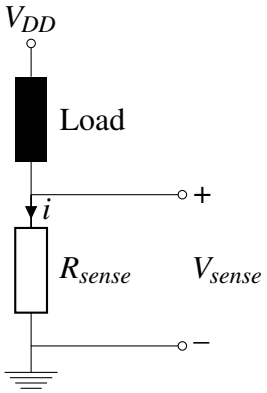


Figure 2.6: Low side current sensing.

2.5.4 Quadrature Encoders

To fully observe the state of a DC motor, the speed of the shaft is of interest. A common way of acquiring the speed is through an optical quadrature encoder. Such a device consists of a rotating disc, a light source and a light sensor. The disc has a defined pattern which either blocks or passes the light emitted from the source, resulting in pulse trains at the light sensor output. By displacing two pulse trains by 90° it is possible to determine not only the speed, i.e. frequency, but also the direction of rotation. Figure 2.7 depicts such signals at counterclockwise rotation since channel B is low at the rising edge of channel A [23].

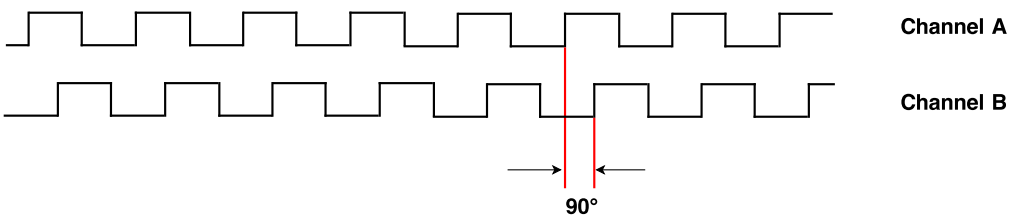


Figure 2.7: Quadrature encoder output channels A and B in counterclockwise rotation.

2.5.5 Inertial Measurement Unit

Measurements of the velocity and orientation of a body can be made with the aid of an Inertial Measurement Unit (IMU), commonly referred to as an electronic device with a three-axis accelerometer, three-axis gyroscope and, in some cases, a three-axis magnetometer. The outputs of the sensors are then combined with some kind of sensor fusion algorithm, e.g. the complementary filter discussed in section 2.3. Typical application areas of an IMU is air crafts, unmanned control, navigation and more recently, handheld devices like smartphones and portable gaming platforms.

Accelerometers

Modern accelerometers are often based on micro electro-mechanical systems (MEMS). This is technology that simply put consists of a damped mass on a spring. When subjected to acceleration the mass is displaced, enabling the displacement to be measured by a capacitive sensor [24]. By placing three such masses orthogonal to each other the acceleration around all three axes can successfully be measured. Depending on the application, a trade-off between sensitivity and maximum measurable acceleration have to be made.

Gyroscopes

Gyroscopes are common as sensors of angular rate in navigational applications. There are many different types of gyroscopes relying on different physical phenomenons. The vibratory MEMS rate gyroscope is perhaps the most common in embedded control applications. The principle of operation relies on the Coriolis Effect which causes vibration inside the gyroscope when it is rotated. The vibration is then detected capacitively and the signal is conditioned to produce a voltage proportional to the angular rate [24]. It is important to note that since the gyroscope measures the angular rate, the angle will drift over time if it is used as the sole sensor for angle approximation. This is due to the fact that even at zero rate there will be a small value present due to noise, calibration errors and temperature variations which will accumulate quickly as the velocity is integrated.

2.5.6 Analog filtering techniques

Actual electrical circuits will always be subjected to noise at some level. Either from the 50 Hz electrical mains or other close-by signals by capacitive or inductive coupling. Some circuits will themselves emit noise, such as logic and other high frequency switching equipment. By using analog or digital filters with a proper cut-off frequency these effects can be mitigated. Low pass filters can also be used to turn a PWM signal into a DC-level value. Two types of analog filters used for these purposes are described below.

RC-filters

The most simple type of analog filter is the RC-filter (resistor/capacitor), shown in Figure 2.8 configured as a low pass filter.

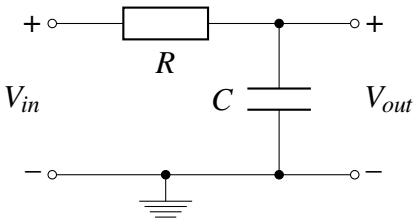


Figure 2.8: First order low-pass RC-filter circuit.

The transfer function for such a filter is given by

$$\mathrm{H}(j\omega) = \frac{1}{1 + j\omega RC} \tag{2.18}$$

which places the cut-off frequency at $\omega_c = \frac{1}{RC}$. To have a steeper attenuation several RC-filters can be cascaded. In some cases an operational amplifier at unity gain is placed as a buffer on the output.

Common Mode Choke

Signal cables can be subjected to common-mode interference, i.e. interference appearing on multiple signals simultaneously. Winding the signal cables in coils around the same core effectively cancels the interference by presenting high impedance against common mode signals and low impedance against differential mode signals.

Chapter 3

Modelling and Simulations

This chapter covers the simulations of the proposed system. It starts out by describing the modelling of the system using the Lagrangian mechanics approach outlined in section 2.1. With the model as a basis, the control algorithm is developed using control theory and the MATLAB/Simulink environment which is described in the second part of the chapter. Finally, the results from the simulations are presented from which the construction design is then built upon.

3.1 Modelling

In order to develop and evaluate the performance of a control algorithm acting on a dynamical system an intimate knowledge of the system is desirable. In general, a model consists of a set of differential equations describing the evolution of the system in time. From a control theory point of view, it is beneficial to denote the differential equations on a state space form as it enables the use of sophisticated methods of control design as described in section 2.2.2. This section deals with the derivation of the state space model and is inspired by the work of a bachelor's thesis at ETH Zürich [25]. Worth noting is that all algebraic calculations were made with the aid of the computational software Wolfram Mathematica since the resulting mathematical expressions consists of thousands of terms.

3.1.1 System description

The proposed design for the Ballbot robot is shown as a CAD model in Figure 3.1. The model was developed in the open-source CAD modelling software program FreeCAD. As described in Chapter 1 a Ballbot is a robot which balances on a ball. In the proposed design the means of which the robot balances is by three radially equidistantly placed actuating omniwheels. An omniwheel is a wheel with small cylindrical wheels

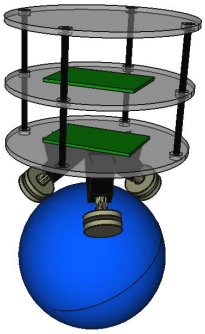


Figure 3.1: A CAD-model of the proposed robot design.

around its circumference, which enables lateral movement in the direction of the wheel axis. For reasons of simplicity, the omniwheels is modelled as regular wheels in the CAD-model. When modelling the system mathematically, the model is reduced to three simple bodies; *one hollow sphere* representing the ball, *three discs* representing the actuating omniwheels and *one solid cylinder* representing the body of the robot. Table 3.1 lists the mechanical parameters used when modelling the system.

Table 3.1: Parameters of the mechanical system

Description	Variable	Value
Ball		
Radius of ball	r_{Ba}	0.107 m
Mass of ball	m_{Ba}	1 kg
Omniwheels		
Radius of omniwheel	r_{OW}	0.05 m
Mass of omniwheel	m_{OW}	0.335 kg
Mounting angle of motors in vertical plane	w_{Inc}	55°
Mounting angle of motor 1 in horizontal plane	w_{Sep1}	0°
Mounting angle of motor 2 in horizontal plane	w_{Sep2}	120°
Mounting angle of motor 3 in horizontal plane	w_{Sep3}	240°
Moment of inertia for motor	J_{mot}	5.9 µkg m^2
Body and omniwheels		
Radius of body (approximated as cylinder)	r_{Bo}	0.1 m
Mass of body and omniwheels	m_{BoOW}	5 kg
Other parameters		
Distance between centre of ball and COG of the body	l	0.5 m
Gravitational acceleration	g	9.81 m/s^2
Gear box ratio	i_{gear}	14

Using the parameters of 3.1 the moment of inertia of the bodies can be calculated. The ball is approximated as a hollow sphere resulting in the following expression

$$I_{Ba} = \begin{bmatrix} I_{Ba_x} & 0 & 0 \\ 0 & I_{Ba_y} & 0 \\ 0 & 0 & I_{Ba_z} \end{bmatrix} \quad \text{where} \qquad I_{Ba_i} = \frac{2}{3} m_{Ba} r_{Ba}^2 \quad \text{for} \quad i = x, y, z. \tag{3.1}$$

As for the omniwheels, the moment of inertia of the motor transformed by the gear ratio of the gearbox have to be taken into account resulting in

$$I_{OW_i} = \frac{1}{2} m_{OW} r_{OW}^2 + i_{gear}^2 \cdot J_{mot} \quad \text{for} \quad i = 1, 2, 3. \tag{3.2}$$

Finally, the inertia for the body is calculated according to

$$I_{BoOW} = \begin{bmatrix} I_{BoOW_x} & 0 & 0 \\ 0 & I_{BoOW_y} & 0 \\ 0 & 0 & I_{BoOW_z} \end{bmatrix} \quad \text{where} \tag{3.3}$$

$$I_{BoOW_x} = \frac{1}{12} m_{BoOW} (3 r_{Bo}^2 + (2(l - r_{Ba}))^2) + m_{BoOW} (r_{Bo} + l)^2, \tag{3.4}$$

$$I_{BoOW_y} = I_{BoOW_x} \quad \text{and} \quad I_{BoOW_z} = \frac{1}{2} m_{BoOW} r_{Bo}^2. \tag{3.5}$$

3.1.2 State selection

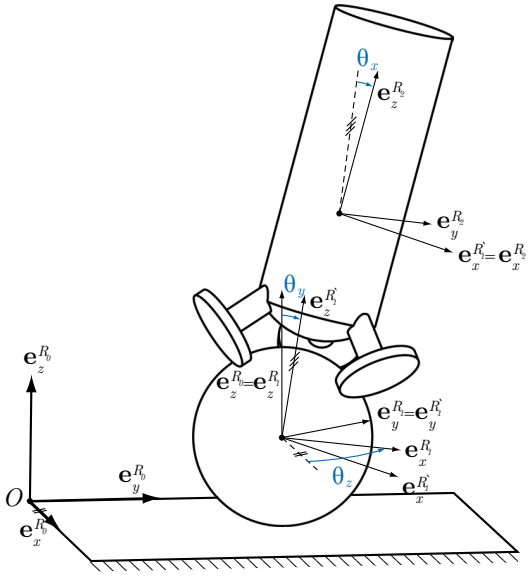


Figure 3.2: Figure: The relative reference systems and their relationships [25].

To represent the orientation of the bodies, four different coordinate reference systems R_0, R_1, R_1', R_2 are used. Using the Tait-Bryan type of Euler angles the following sequence of transformations relate the reference systems to each other:

$$R_0 \xrightarrow{\theta_z} R_1 \xrightarrow{\theta_y} R_1' \xrightarrow{\theta_x} R_2 \tag{3.6}$$

The reference systems and their relationships are shown in figure 3.2. As can be seen, R_1 is derived by rotating around the z-axis of R_0 with θ_z. In an analogous manner R_1' is derived by rotating around the y-axis of R_1 with θ_y and R_2 is derived by rotating around the x-axis of R_1' with θ_x.

The system has five degrees of freedom (DOF), the position of the ball in the plane (two DOF) and the orientation of the body (three DOF). This requires a state vector with ten states consisting of the coordinates in each DOF and their respective derivatives. To match the output of a possible sensor solution with an Inertial Measurement Unit (IMU) and a rotary encoder the following states were chosen

$$\vec{x} = [\theta_x \;\; \dot{\theta}_x \;\; \theta_y \;\; \dot{\theta}_y \;\; \theta_z \;\; \dot{\theta}_z \;\; \phi_x \;\; \dot{\phi}_x \;\; \phi_y \;\; \dot{\phi}_y]^T. \tag{3.7}$$

As the reference system R_2 is fixed relative to the body the variables θ_x, θ_y and θ_z describe the orientation of the body. The variables ϕ_x and ϕ_y are the rolled angles of the

ball in the respective direction, ϕ_x being the clockwise rotation around the x-axis and ϕ_y the counterclockwise rotation around the y-axis, both referred to the R_1 reference system. The remaining part of the system to describe is the omniwheels, whose angular velocities around their motor axis are denoted $\dot{\psi}_1$, $\dot{\psi}_2$ and $\dot{\psi}_3$ also related to the R_2 reference system. This gives the following angular velocity vectors describing the full system:

$$
\begin{aligned}
{}_{R_1}\vec{\Omega}_{Ba} &= \begin{bmatrix} \dot{\phi}_x \\ \dot{\phi}_y \\ 0 \end{bmatrix}, \qquad \begin{cases} {}_{R_2}\omega_{OW1} = \dot{\psi}_1 \\ {}_{R_2}\omega_{OW2} = \dot{\psi}_2 \\ {}_{R_2}\omega_{OW3} = \dot{\psi}_3 \end{cases}, \\
{}_{R_2}\vec{\Omega}_{Bo} &= J \cdot \dot{\vec{\theta}} = \begin{bmatrix} \dot{\theta}_x - \sin\theta_y \cdot \dot{\theta}_z \\ \cos\theta_x \cdot \dot{\theta}_y + \cos\theta_y \cdot \sin\theta_x \cdot \dot{\theta}_z \\ -\sin\theta_x \cdot \dot{\theta}_y + \cos\theta_x \cdot \cos\theta_y \cdot \dot{\theta}_z \end{bmatrix}
\end{aligned}
\tag{3.8}
$$

where the velocity vector of the body is the time variation of the Tait-Bryan angles $\dot{\vec{\theta}}$ converted to the R_2 reference system with the Jacobian matrix J.

As inputs to the system, the three torques generated by the motors are chosen. These are denoted T_1, T_2 and T_3.

3.1.3 Binding equations

To be able to describe the system fully by the chosen states in Equation 3.7, some of the variables in the final expressions in Equation 3.8 have to be substituted through binding equations which relates the bodies to each other.

Relationship between omniwheels and ball

Assuming no slip between the omniwheels and ball the velocities of the bodies can be directly related to each other. The validity of the assumption have to be considered when designing the system by seeing through that there is enough friction between the omniwheels and the ball and that the applied torques is within reasonable values. Mathematically, the assumption states that the tangential speed of the omniwheels have to be exactly the same as the speed of the ball in the same direction.

To express the surface speed of the ball, the angular velocity of the ball relative to the body have to be calculated as

$$
{}_{R_2}\vec{\Omega}_{BaREL} = T_{21} \cdot {}_{R_1}\vec{\Omega}_{Ba} - {}_{R_2}\vec{\Omega}_{Bo} \tag{3.9}
$$

where T_{21} is the transformation matrix between the ball and body reference system. Defining the vectors from the centre of the ball to the omniwheels' contact points on the

ball as ${}_{R_2}\vec{a}_{C0C1}$, ${}_{R_2}\vec{a}_{C0C2}$ and ${}_{R_2}\vec{a}_{C0C3}$ and the unit vectors for the tangential speed of the omniwheels as ${}_{R_2}\vec{u}_1$, ${}_{R_2}\vec{u}_2$ and ${}_{R_2}\vec{u}_3$ the no slip assumption can be formulated as

$$\left({}_{R_2}\vec{\Omega}_{BaREL} \times {}_{R_2}\vec{a}_{C0Ci}\right) \cdot {}_{R_2}\vec{u}_i = {}_{R_2}\omega_{OWi} \cdot r_{OW} \qquad \text{for} \quad i = 1,2,3. \tag{3.10}$$

The left hand side corresponds to the ball's surface speed in the omniwheel direction and the right hand side to the tangential speed of the omniwheel.

Speed expressions for energy calculations

The absolute value of the rotation of the omniwheels have to take into account the motion of the body as well. By using scalar projection of the body velocity vector onto the vectors of which the omniwheels rotate around results in the following set of equations

$${}_{R_2}\Omega_{OWi} = {}_{R_2}\omega_{OWi} + \frac{{}_{R_2}\overrightarrow{MW_i}}{||{}_{R_2}\overrightarrow{MW_i}||} \qquad \text{for} \quad i = 1,2,3. \tag{3.11}$$

Solving the set of equations in 3.10 for ${}_{R_2}\omega_{OW1}$, ${}_{R_2}\omega_{OW2}$ and ${}_{R_2}\omega_{OW3}$ and replacing the solutions into Equation 3.11 results in explicit expressions only dependent of the states given in Equation 3.7.

To be able to calculate the translational energy of the ball, the translational speed is needed. It is given by

$${}_{R_0}\dot{\vec{a}}_{C0} = (T_{01} \cdot {}_{R_1}\Omega_{Ba}) \times {}_{R_0}\vec{a}_{GrC0} \tag{3.12}$$

where ${}_{R_0}\vec{a}_{GrC0}$ is the vector from the ground to the centre of the ball and T_{01} the transformation matrix between the ball and inertial reference system.

3.1.4 Energies

The next step in the Lagrangian approach is to calculate the kinetic energy, both translational and rotational, and potential energy for each of the bodies. The zero point for the potential energy is chosen as the plane intersecting with the centre of the ball.

Ball

The kinetic energy for the ball is given by

$$T_{Ba} = \frac{1}{2} \cdot m_{Ba} \cdot {}_{R_0}\dot{\vec{a}}^T_{C0} \cdot {}_{R_0}\dot{\vec{a}}_{C0} + \frac{1}{2}\,{}_{R_1}\Omega^T_{Ba} \cdot I_{Ba} \cdot {}_{R_1}\Omega_{Ba} \tag{3.13}$$

and due to the choice of the zero reference through the centre of the ball the potential energy is given by

$$U_{Ba} = 0. \tag{3.14}$$

Body

As mentioned when calculating the inertia of the bodies, a simplification is made regarding the inertia of the body and the motors, they are considered as one whole body. With the help of a vector ${}_{R_2}\vec{a}_{C0CBoOW}$ from the centre of the ball to the centre of gravity of the body the kinetic energy is given by

$$\begin{aligned} T_{BoOW} = \frac{1}{2} \cdot m_{BoOW} \cdot {}_{R_0}\dot{\vec{a}}^T_{C0} \cdot {}_{R_0}\dot{\vec{a}}_{C0} + \frac{1}{2} {}_{R_2}\Omega^T_{Bo} \cdot I_{BoOW} \cdot {}_{R_2}\Omega_{Bo} + \\ m_{BoOW} \cdot \left(T_{20} \cdot {}_{R_0}\dot{\vec{a}}_{C0}\right) \cdot \left({}_{R_2}\vec{\Omega}_{Bo} \times {}_{R_2}\vec{a}_{C0CBoOW} \right) \end{aligned} \tag{3.15}$$

where the last term models the coupling introduced from the fact that the centre of ball is reference.

The potential energy is given by

$$U_{BoOW} = -m_{BoOW} \cdot G \cdot T_{02} \cdot {}_{R_2}\vec{a}_{C0CBoOW} \tag{3.16}$$

where G denotes the gravitational vector and T_{02} the transformation matrix between the body and inertial reference system.

Omniwheels

The rotational energy of the omniwheels are however modelled separately as

$$T_{OW_i} = \frac{1}{2} \cdot I_{OW} \cdot {}_{R_2}\Omega^2_{OWi} \qquad \text{for} \quad i = 1,2,3. \tag{3.17}$$

3.1.5 Equations of motion

Given the expressions derived in the previous section, the Lagrangian is formed as

$$L = T - U = T_{Ba} + T_{BoOW} + T_{OW1} + T_{OW2} + T_{OW3} - U_{Ba} - U_{BoOW}. \tag{3.18}$$

What remains before the final declarations of the Euler-Lagrange equations is the generalised forces F_i. The generalised forces considered are the actuating motor torques T_1, T_2 and T_3 which act on ${}_{R_2}\omega_{OW1}$, ${}_{R_2}\omega_{OW2}$ and ${}_{R_2}\omega_{OW3}$. Those are not functions of the states in Equation 3.7 hence a polynomial separation has to be done expressing them as

$${}_{R_2}\omega_{OWi} = J_{Ti} \cdot \dot{\vec{x}} \qquad \text{for} \quad i = 1,2,3. \tag{3.19}$$

The counter torques T_{C1}, T_{C2} and T_{C3} acting on the body in the opposite direction of T_1, T_2 and T_3 is given by

$$\vec{T}_{Ci} = -T_i \cdot \frac{{}_{R_2}\overrightarrow{MW_i}}{||{}_{R_2}\overrightarrow{MW_i}||} \qquad \text{for} \quad i = 1,2,3. \tag{3.20}$$

This gives the following expression of the generalised forces

$$F_i = J_{T1}^T \cdot T_1 + J_{T2}^T \cdot T_2 + J_{T3}^T \cdot T_3 + J^T \cdot \left(\vec{T}_{C1} + \vec{T}_{C2} + \vec{T}_{C3} \right) \tag{3.21}$$

where J_{Ti} for $i = 1,2,3$ is given from Equation 3.19 and J is the Jacobian given in Equation 3.8.

Finally, the Euler-Lagrange equations are given by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_i}\right) - \frac{\partial L}{\partial x_i} = F_i, \quad i = 1, ..., 10 \tag{3.22}$$

with L as in Equation 3.18 and F_i as in Equation 3.21. Solving these equations yields the equations of motion.

3.1.6 State-space model

The derived equations of motion is, as mentioned, a set of differential equations describing the modelled system. These are to be converted into a state-space model as

$$\dot{\vec{x}} = A\vec{x} + B\vec{u}, \quad y = C\vec{x} + D\vec{u} \tag{3.23}$$

$$\vec{x} = [\theta_x \;\; \dot{\theta}_x \;\; \theta_y \;\; \dot{\theta}_y \;\; \theta_z \;\; \dot{\theta}_z \;\; \phi_x \;\; \dot{\phi}_x \;\; \phi_y \;\; \dot{\phi}_y]^T \tag{3.24}$$

$$\vec{u} = [T_1 \;\; T_2 \;\; T_3]^T. \tag{3.25}$$

However, to be able to factor out the matrices A,B,C and D the nonlinear differential equations have to be linearised. A suitable linearisation point is the unstable equilibrium where all states are zero which also is the point around which stabilisation is desired.

3.1.7 Odometry

Since there is no direct way to measure the states ϕ_x and ϕ_y related to the movement of the ball, these have to be approximated by other available sensors via odometry calculations similar to those deriving the equations of motions. This results in expressions for $\dot{\phi}_x$ and $\dot{\phi}_y$ as functions of the angular velocities of the omniwheels, $\dot{\psi}_1$,$\dot{\psi}_2$ and $\dot{\psi}_3$ as well as the states θ_x, θ_y, θ_z and their derivatives.

3.2 Control design

The control method for regulation of the body's tilt angles and the robot's position is LQG-control (see section 2.2.2). Since the outputs from the real robot are going to be sampled measurement values the system is a discrete-time system, from the controller's point of view. The first step to develop a suitable controller is therefore to discretise the continuous-time state-space model given in section 3.1. This is easiest done in MATLAB using the command:

$$sysd = c2d(sys, Ts),$$

where *sys* is the continuous-time system model and T_s the sample time. The discrete-time LQR gain matrix K_d can then be calculated with the MATLAB command $dlqr$ which takes the matrices A_d and B_d as inputs, together with the weight matrices Q_x and Q_u.

Since the measurements most likely will have some additional noise a Kalman filter is implemented as well. The Kalman gain L is therefore calculated with the MATLAB command *kalmd*. The function takes as inputs the continuous-time system with state-space matrices $A, [B \quad F], C, D$ as well as the matrices R_w and R_v and the sample time T_s. Note that in our case, since there is no feedthrough term, the matrix D is simply zeros of the appropriate dimensions.

3.3 Simulink implementation

In order to simulate the model it is implemented in the Matlab tool Simulink. By doing this it is possible to verify and investigate if the model behaves in a realistic way. With the help of the simulations it is also easy to evaluate the developed controller and determine if it has the desired behaviour. It also makes it easier to analyse the system's robustness against disturbances. Figure 3.3 shows the overall layout of the Simulink implementation of the system displaying the main blocks. In Figure 3.4 the implementation of the LQG controller is shown and in Figure 3.5 the implementation of the Kalman filter.

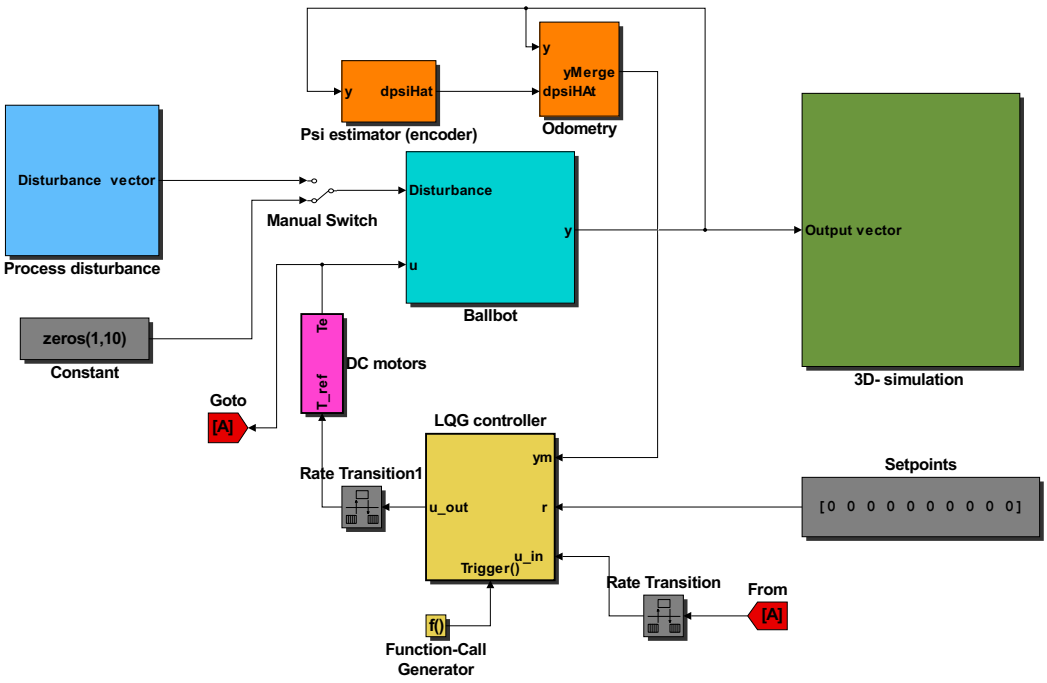


Figure 3.3: Implementation of the system in the simulation environment Simulink.

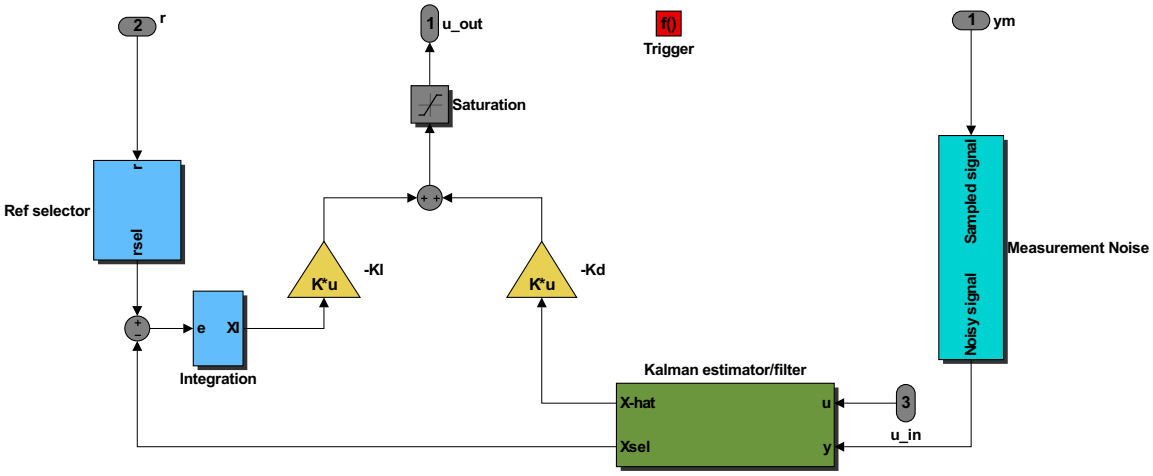


Figure 3.4: Implementation of the LQG-controller in Simulink.

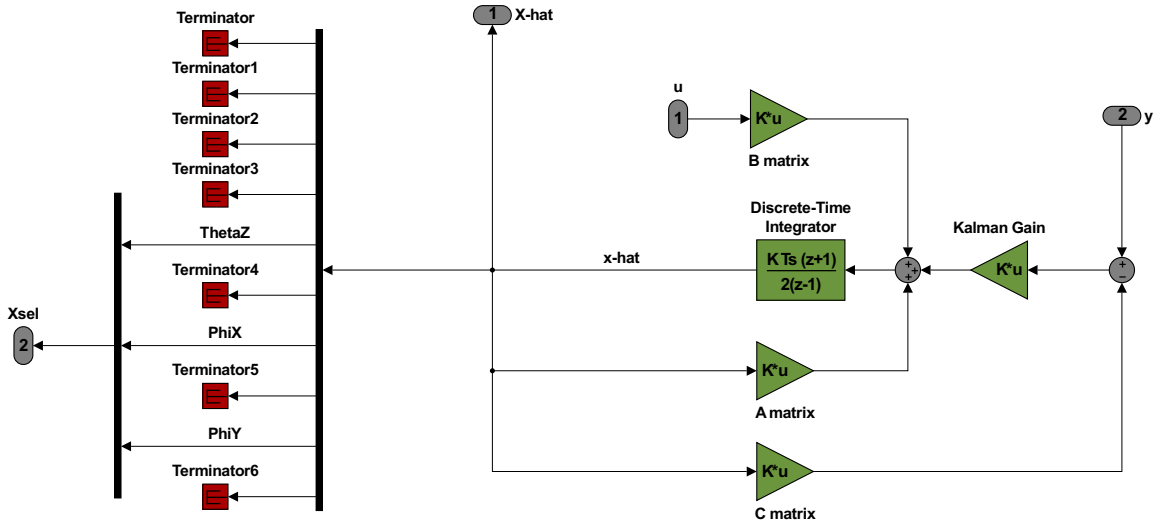


Figure 3.5: Implementation of the Kalman filter in Simulink.

3.4 Simulation results

In this section some results of the simulations performed in Simulink are going to be shown. The simulations presented in Figure 3.6 up to and including Figure 3.9 have the initial angles $\theta_x = 10^\circ$ (0.1745 rad) and $\theta_y = 5^\circ$ (0.0873 rad), all other states are zero. The set points, i.e. the references the controller wants to take the states to, are all zero as well. The variance of the added white Gaussian noise is 0.05 rad^2.

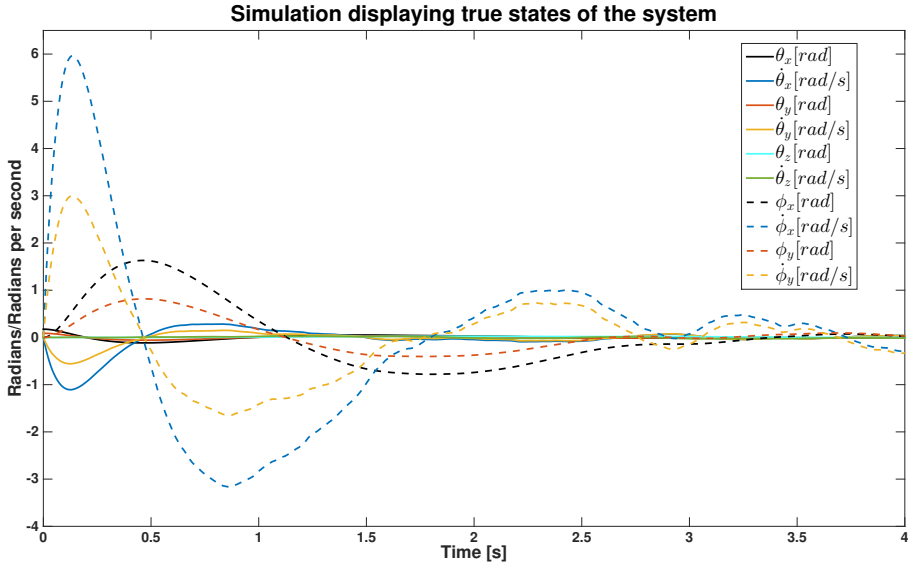


Figure 3.6: Simulation with initial angles $\theta_x = 10^\circ$ (0.1745 rad) and $\theta_y = 5^\circ$ (0.0873 rad) and all set points at zero. As can be seen the system manges to stabilise. The plotted angles and angular velocities are the ones that comes directly from the block "Ballbot" in Figure 3.3.

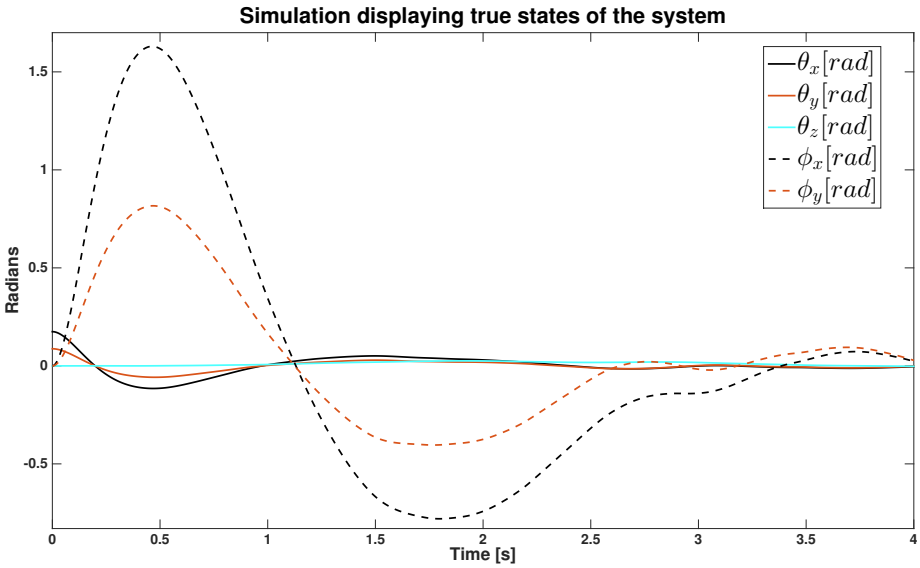


Figure 3.7: Simulation with initial angles $\theta_x = 10^\circ$ (0.1745 rad) and $\theta_y = 5^\circ$ (0.0873 rad) and all set points at zero, i.e the same scenario as in Figure 3.6. However this plot only shows the angles and not the angular velocities.

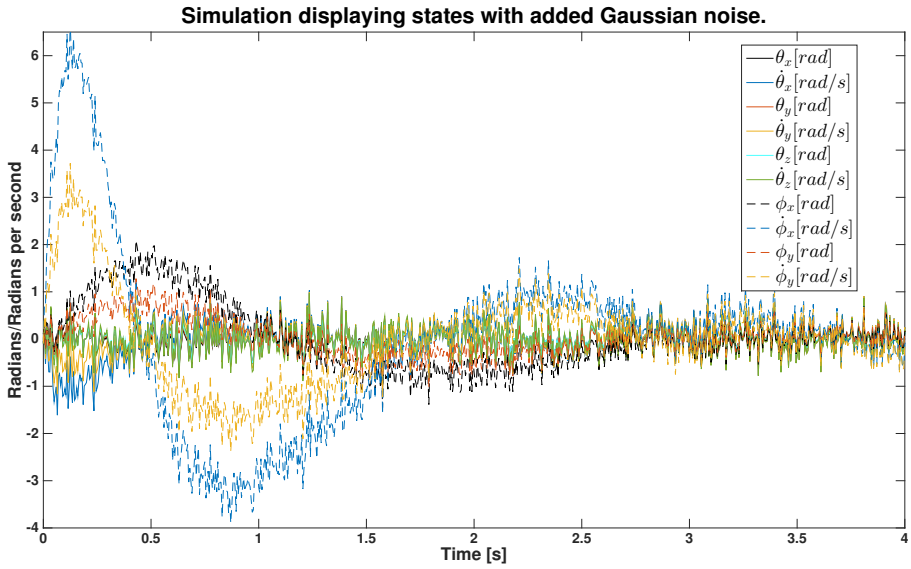


Figure 3.8: Simulation for same scenario as in Figure 3.6 and 3.7. The plot shows the signals entering the Kalman filter in Figure 3.5. The added white Gaussian noise has a variance of 0.05 rad^2.

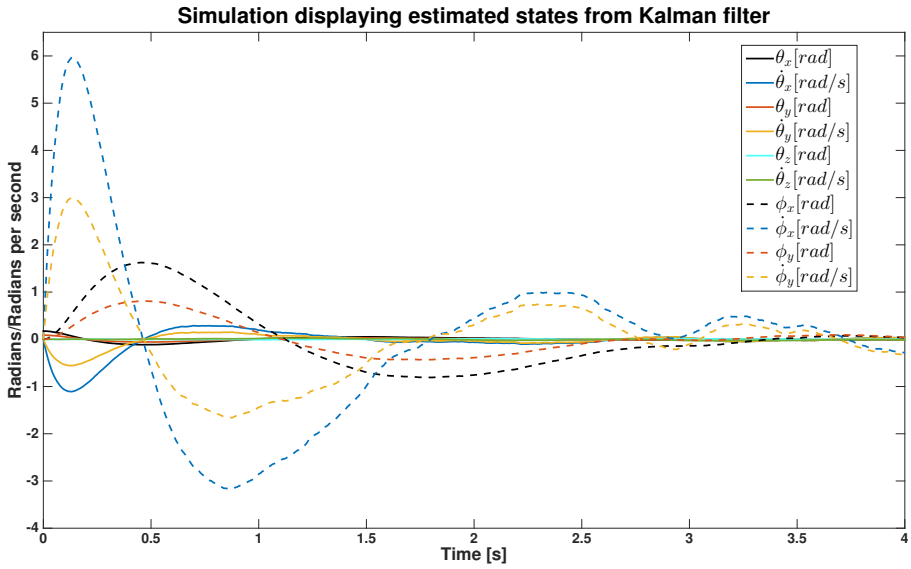


Figure 3.9: The plot shows the output from the Kalman filter in Figure 3.5 when the noisy signals in Figure 3.8 act as inputs. As can be seen the Kalman filter reduces the noise very well.

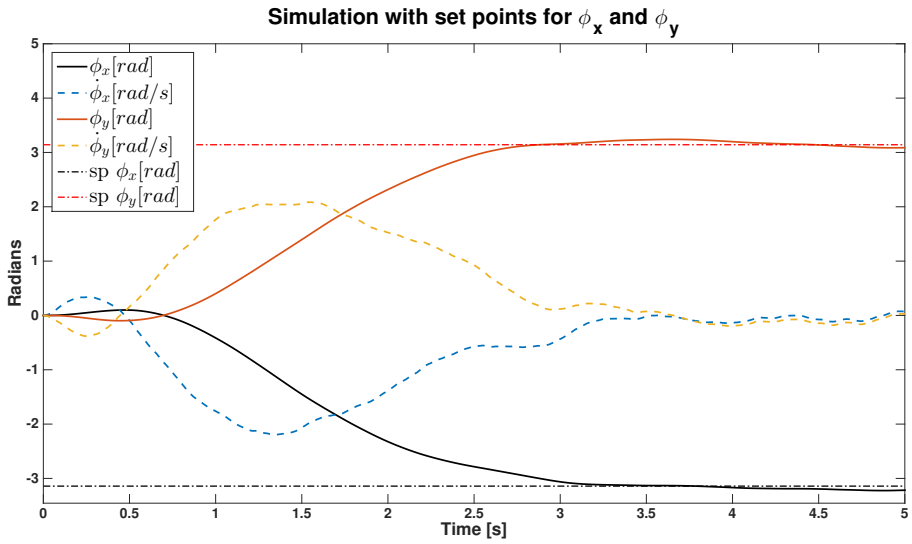


Figure 3.10: Simulation showing the translation of the ball. All initial angles at zero and set points for $\phi_x = -\pi$ rad and $\phi_y = \pi$ rad, all other at zero. As can be seen the system manges to take the states toward their respective set point in a reasonable time.

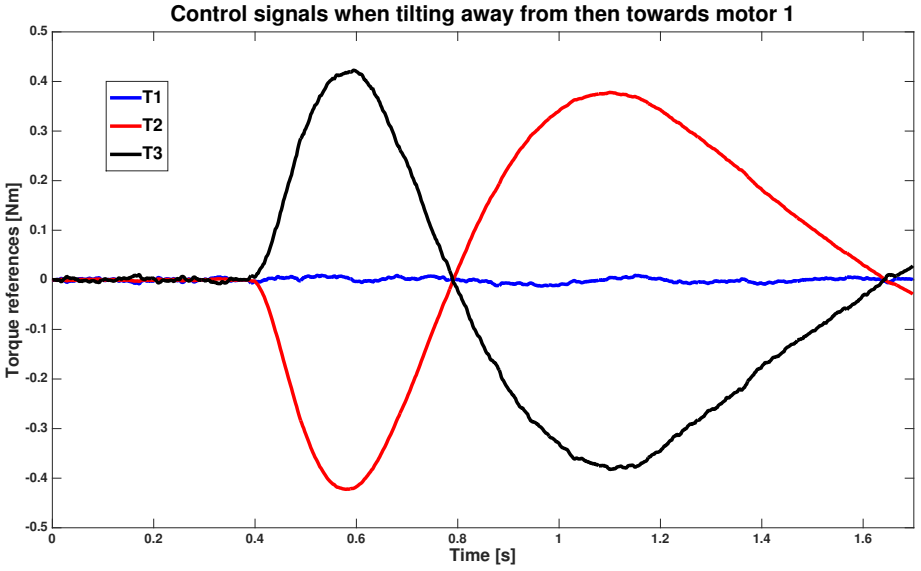


Figure 3.11: Simulation showing the output torque references from the controller when the robot is tilted away from and then towards motor 1.

Chapter 4

System Design

This chapter covers the construction and design of the robot and describes its major parts, both in hardware and software.

4.1 Hardware

This sections deals with the hardware design and motivates the choices made. It also briefly describes the already available hardware components which have been used within the project.

4.1.1 Chassis

The chassis design has been presented in the modelling subsection 3.1.1. Early on, a design using three equidistantly placed actuating omniwheels were chosen to enable the robot to turn around its own vertical axis. It also eliminates the demand for other braces to keep the body at a fixed distance from the ball. This design has been used in several other projects such as Rezero [25] and BallIP [3]. The latter also incorporates a simple storey-like solution using circular transparent hard plastic sheets. Combining these with threaded rods, washers and nuts a very flexible design could be achieved, capable of post-construction adjustments to make room for different equipment but also testing different sensor placements and weight distributions. The attachment of the flexible storey can be seen in Figure 4.1.

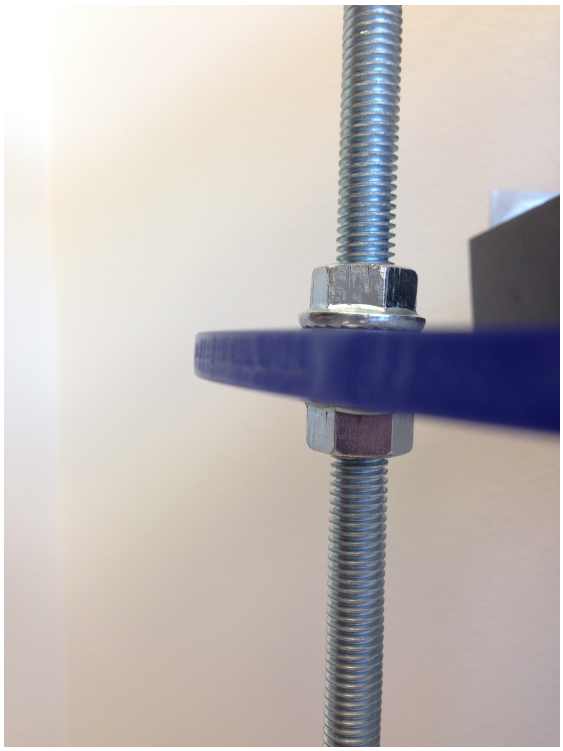


Figure 4.1: Attachment of flexible storeys within the robot body.

The actuating omniwheels were attached to the body with metal braces which were manually bent to the right angle to fit the ball at all three contact points. The ball is a high friction medicine ball specifically chosen for not having any seams which could introduce more nonlinear phenomena.

4.1.2 Microcontroller platform

The computational platform used for the project is the Open Dependable Electrical and Electronics Platform (ODEEP) developed at QRTECH AB, shown in Figure 4.2. It is built upon the automotive grade microcontroller MPC5567 from Freescale and has several communication interfaces built in such as SPI, CAN and Ethernet to name a few. Examples of other hardware integrated on the board are wide range power supply regulation, high current driver outputs and SD-card support. However, the main reason for choosing ODEEP is the possibility to generate complex algorithms from MATLAB/Simulink directly as well as it supports AUTOSAR. To program and debug the system, a Powerful Embedded Ethernet Debug Interface (PEEDI) from Ronetix where used, connecting to the platform via the Nexus interface.

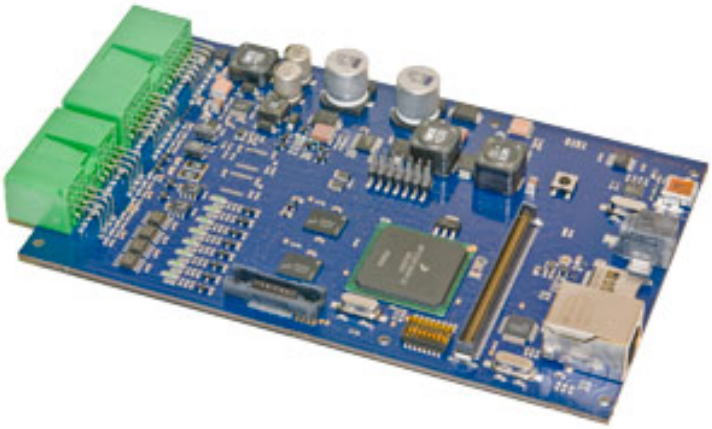


Figure 4.2: The Open Dependable Electrical and Electronics Platform (ODEEP) developed at QRTECH AB.

4.1.3 Sensors and actuators

As actuators for the system, both brushless and brushed DC motors were considered. However, since brushless DC motors require more complex and expensive control electronics, brushed DC motors were chosen. Also, the physical modelling of a DC motor is much simpler, making it easier to evaluate and simulate in MATLAB/Simulink. The specific model is 3266E_0 from Phidgets with a planetary gearbox for increased torque and encoders for control purposes. As drivers for the motors, Sabertooth 2x5 from DimensionEngineering were chosen. Each driver can supply two brushed DC motors with up to 5A and they can be operated in several different modes, analog, serial or R/C input. In analog mode, the output from the drivers is a voltage to the motors proportional to the analog voltage applied at the input. The drivers are built upon a H-bridge driver at 32 kHz, enabling operation in both directions. Details about brushed DC motors and H-bridge drivers have been further explained in subsection 2.5.2.

As for the sensors measuring the attitude of the robot, the motion processing unit MPU-6000 from InvenSense was used. It combines a three-axis gyro with a three-axis accelerometer and has built-in signal conditioning and ADC conversion. The digital signals can then be transferred over either I2C or SPI. The chip also incorporates a dedicated processor for motion related calculations called Digital Motion Processor (DMP). This could possibly reduce the main processor load significantly by letting the DMP handle the sensor fusion, however it requires knowledge about how to program the motion processor and a full documentation for the DMP is not yet available. This resulted in the sensor fusion having to be made on the main processor to save development time. Considering the fact that both the gyro and accelerometer data for all the three axes have to be transmitted, the SPI interface was chosen because its significantly higher

transmission speed compared to I2C. AUTOSAR has built-in drivers for SPI and the ODEEP platform has hardware support for SPI which makes the implementation easier. To have a better physical interface to work against an evaluation board of the chip where used with header connectors, on-board power regulator and magnetometer.

In addition to the attitude sensors, the robot needs to know its orientation in the plane since the rolled distances of the ball in two perpendicular axis are states in the model. This is achieved by the use of quadrature encoders mounted on the motor axis which emits a pulse pattern when the motor operates. More information regarding quadrature encoders can be found in subsection 2.5.4.

4.1.4 Expansion card

To have a coherent interface between the sensor/actuator components and the computational part, an expansion card was designed in Altium Designer. Using the 80 pole board-to-board connector on ODEEP and stacking the expansion board on top of it, all necessary interfaces were made available. Apart from connecting the external hardware to the MCU, some signal conditioning had to be made on the board as well. The motor drivers operate in analog mode and hence the MCU has to control an analog voltage. This is achieved by controlling the duty cycle of a 100 kHz PWM signal and then feeding it through a second order analog filter with a buffer residing on the expansion card. Also, to cancel out common-mode noise introduced by the motor on the quadrature encoders' signals, common-mode chokes were used. To relieve the MCU from completely decoding the quadrature signal, the dedicated integrated circuit HCTL-2032 from Avago Technologies was placed on the board. The circuit decodes the quadrature pulses and outputs a clock and direction signal which can be read by the MCU.

Since the control algorithm generates torque references as inputs to the system and the drivers only can control the speed of the motors, an outer PI-loop is designed to control the torque. Hence measurements of torque, or more precisely current, is needed. This is also done on the expansion board by the use of an integrated Hall current sensor, ACS718 from Allegro, in low-side current sensing mode as described in Subsection 2.5.3. The analog signal output from the sensor is then fed to the ADC-unit of the MCU, making it available to use as a control feedback. In Figure 4.3 a complete CAD model of the designed expansion board can be seen. The IMU evaluation board is mounted into the female header in the middle of the board and is fixed with screws through the holes. Details about other auxiliary components on the board such as pull-up resistors, decoupling capacitors etc. is considered to be out of the scope for this report and thus not described any further.

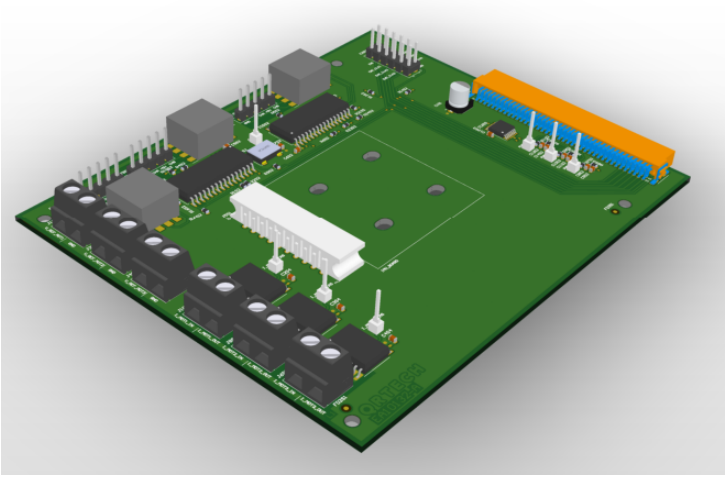


Figure 4.3: CAD model of the designed expansion board.

4.1.5 Power

Both ODEEP and the motor drivers need power in order to operate. The most cost-effective and power dense portable energy source was found to be lithium polymer batteries (LiPo). It is the most common battery type for radio controlled hobby vehicles and hence has a wide range of capacities and voltages commercially available. The battery chosen for the robot is a Zippy Flightmax 3000 mA h with four cells and a nominal voltage at 14.8 V. ODEEP can run at any voltage between 12 to 24 V but the motors is rated at 12 V and hence, a voltage converter is needed to supply them with the correct voltage level. A switching regulator, Quanum QM12V5A-UBEC, designed for radio controlled applications was thus fitted together with the battery.

4.2 Software development work flow

The software implemented in the robot is based on the software architecture AUTOSAR. To be able to develop software following this standard two different programs are used. For configuration of the AUTOSAR environment an AUTOSAR authoring tool, namely Arctic Studio, is used. To write more complex algorithms like the LQG regulator MATLAB/Simulink is used.

4.2.1 Arctic Studio and Arctic Core

Arctic Studio and Arctic Core are software developed by the company ArcCore. Arctic Studio is based on Eclipse and allows the user to configure, validate, generate and integrate BSW modules and the RTE according to the AUTOSAR standard. From Arctic Studio it is also possible to create a Software Component Description (SWCD) that can be exported to other programs. The SWCD specifies for instance the interfaces, data types and events used in the SWC and creates a skeleton that can be filled with content in for example Simulink. This is often referred to as a top-down approach when developing AUTOSAR SWCs.

Arctic Core can be seen as the heart of the program. It contains among other things predefined interfaces, data types and BSW modules. Arctic Core also includes support for several evaluation boards and target hardware [26].

4.2.2 Code generation from Simulink

Being able to use the method of code generation, writing complex algorithms like the one to stabilise the robot is made much simpler. Writing for instance an advanced control algorithm in Simulink is by many considered much more intuitive than writing it directly in for example plain C-code. This model based approach also makes it easier for others to quickly continue working in the same file. Hence the possibility to, with some minor adjustments, use an algorithm that works in simulations straight off with the help of code generation can reduce development time a lot. It also increases the chance to discover errors in the design at an early stage.

To be able to import SWCs configured in Arctic Studio as arxml files into Simulink an extra MATLAB Toolbox needs to be downloaded. With the *Embedded Coder Support Package for AUTOSAR* its possible to use the class *arxml.importer* to import these kinds of files and create a skeleton in Simulink with the correct in-ports, out-ports, data types and events defined.

With the SWC skeleton imported into Simulink the controller used in simulations is copied into it. It is important to note that all integrators and derivatives have to be discrete. To be sure that everything works as intended one should use the inbuilt function *Configure Model as AUTOSAR Component* and validate the configuration, such that it

follows the AUTOSAR standard. The Embedded Coder in MATLAB has support for a number of different processors including Freescale's MPC5567 used in this project. Hence it is possible for the compiler to generate code adjusted for the hardware the code is going to run on. When the code has been generated it is copied back into Arctic Studio where it is mapped to the corresponding task and integrated with the rest of the system.

4.3 Software implementation

To be able to get a fully working system the basic software have to be configured specifically for the processor being used. The high level control algorithms however are implemented as AUTOSAR software components, hence it is possible to re-use them on different ECUs if needed. In Figure 4.4 an overview of the all the software implemented on the robot is shown.

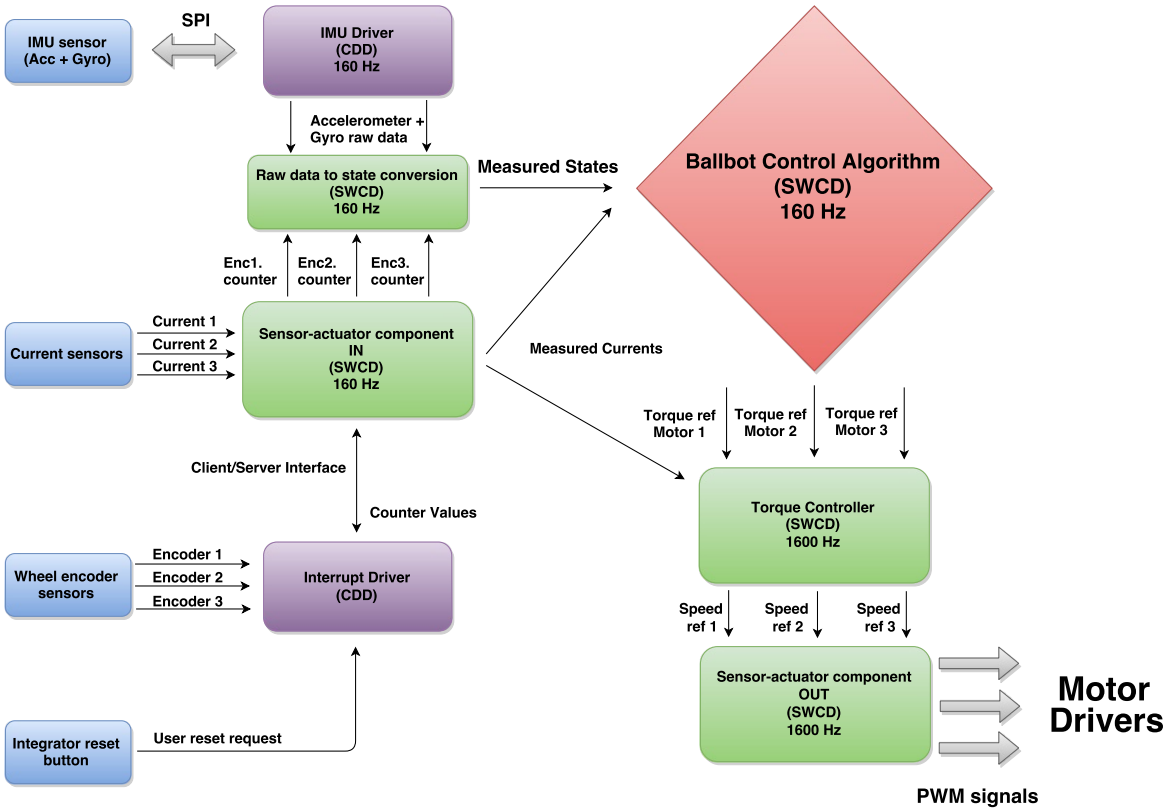


Figure 4.4: Overview of the implemented AUTOSAR structure displaying the Software Components being used and how they interact with each other.

4.3.1 AUTOSAR Software Components

This section describes the created AUTOSAR software components and complex device drivers shown in Figure 4.4.

IMU Driver

The *IMU Driver* handles the communication with the IMU sensor over SPI. Arctic Core includes basic support and functions for the SPI protocol that can be used. However the data being sent and received have to be handled manually. This is done in a so called complex device driver written in C code. The *IMU Driver* initialises the IMU sensor by writing settings to its internal registers and then periodically reads the wanted data from its output registers. The received data from the accelerometer and the gyro is then written to sender ports that through the RTE can communicate with the rest of the AUTOSAR system. The *IMU driver* operates with a frequency of 160 Hz.

Interrupt Driver

Similar to the *IMU Driver* that handles SPI communication, external interrupts also need to be taken care of in a complex device driver. The *Interrupt Driver* is a CDD written in C that is configured to act on events on seven different interrupt pins connected to the microcontroller. Six interrupts belong to the quadrature decoders and one to an external button used for resetting integrators in the control algorithms.

As mentioned in section 2.5.4, the rotated length and direction for each motor can be calculated by counting the number of pulses on the two signals coming from the corresponding quadrature decoder. One interrupt pin is connected to the decoder clock signal and one to the decoder direction signal. When a clock pulse is registered the driver checks the status register flag for the corresponding direction pin. If this flag is set a counterclockwise rotation has occurred, otherwise a clockwise. The driver updates the internal counter variable for that specific motor accordingly. This method makes sure that no pulse is missed and that the rotational states of all motors always are up to date.

The *Interrupt Driver* also handles the integrator reset request triggered when the user pushes a physical button connected to the microcontroller. When such an event is registered a status variable is set and held high until it is read by the appropriate software component.

The driver is set up with an AUTOSAR client-server interface. This means that whilst the internal counter values are updated every time a new interrupt signal is raised, output data from the driver is only sent when a request from another component is made. Hence the frequency new data comes with from the *Interrupt Driver* is determined by how often it is called upon.

Sensor-Actuator Component In

Sensor-Actuator Components are software components that allow the other software components to interact with real sensors and actuators on the hardware. In other words they provide interfaces between the physical values used by the sensors and actuators and the other SWCs. In this project two *Sensor-Actuator Components* have been written. One is handling input signals and one output signals. The input sensor-actuator component periodically fetches measured current values from the ADC unit as well as encoder counter values and reset requests from the *Interrupt Driver*. These data values are then sent on to the SWCs that require them. The update frequency is 160 Hz and the component is written directly in Arctic Studio. The C-code determining the component's behaviour is written manually without any involvement of MATLAB/Simulink.

Raw data to state conversion

The data from the *IMU Driver* and the *Interrupt Driver* consist of raw values from the accelerometer, gyro and encoder counters. This doesn't correspond straight off with the states described in section 3.1.6 that is used by the control algorithm. Thus the raw values must be converted to the appropriate form before they are sent further on.

The angular velocities $\dot{\theta}_x, \dot{\theta}_y$ and $\dot{\theta}_z$ are the simplest ones since they are the gyro values straight off. The angles however are a bit more complicated. Because of the phenomena with gyroscope drift only integrating the angular velocities don't give a result good enough. Hence a complementary filter is used where both gyroscope and accelerometer data are combined. Firstly the angles from the accelerometer are calculated as:

$$Ang_x = atan2\big(acc_y, \sqrt{acc_x^2 + acc_z^2}\big)$$
$$Ang_y = atan2\big(acc_x, \sqrt{acc_y^2 + acc_z^2}\big)$$

Since the accelerometer uses the gravitational force to calculate the angles and since a rotation around the z-axis doesn't influence the measured force in the different axes, it is unfortunately not possible to determine Ang_z in the same way. To get the wanted angles θ_x and θ_y, complementary filters are used in the following way:

$$\theta_x[n] = 0.04 \cdot Ang_x + 0.96 \cdot \big(\theta_x[n-1] + gyr_x \cdot dt\big)$$
$$\theta_y[n] = 0.04 \cdot Ang_y + 0.96 \cdot \big(\theta_y[n-1] + gyr_y \cdot dt\big)$$

θ_z is simply the integrated gyr_z value.

To recreate the rotation angles of the omniwheels (ψ_x, ψ_y and ψ_z) the corresponding counter values for each motor are divided by the number of pulses per revolution and converted to radians by multiplying with 2π.

Ballbot Controller

The *Ballbot Controller* is the main controller of the system that keeps the Ballbot balancing and moving to set points in the plane. As inputs it uses the states recreated in the *Raw data to state conversion SWC* as well as the measured applied currents for each motor. The algorithm first reproduces the states corresponding to the movement of the ball ($\phi_x, \dot{\phi}_x, \phi_y, \dot{\phi}_y$) from the wheel rotation angles ψ_x, ψ_y and ψ_z. All signals are then filtered in a Kalman filter. This is mostly done for filtering away measurement noise. Using this data the control signals corresponding to torque references for each motor are calculated by the Linear Quadratic Regulator.

The Kalman filter wants to know the previous control signals. By converting the measured currents into applied torque this can be achieved. A simpler approach would be to use the previous control outputs instead. However since these only are references to the torque controller it is more true to use the measured values instead. All this is done in Simulink and MATLAB. The *Ballbot Controller* has an update frequency of 160 Hz.

Torque Controller

The motor drivers being used cannot take torque references as inputs. Instead it uses analog input voltages as speed references, where 2.5 V corresponds to no movement, 5 V full speed forward and 0 V full speed backwards. Consequently the torque references from the *Ballbot Controller* must be converted into speed references in the form of different voltage levels. This is done in the *Torque Controller*. Since torque is proportional against current the torque references are firstly converted into current references. By measuring the real currents in the motors and feeding them back in a loop, the error between the reference and the true value can be fed into a PI-regulator. Hence an appropriate control signal and speed reference can be calculated. In other words the *Torque Controller* is really a *Current Controller*, it is developed in Simulink and has an update frequency of 1600 Hz.

Sensor-Actuator Component Out

Similar to the sensor-actuator component that handles input signals this is a component that works as an interface between the software components and the hardware. It takes the requested voltage references from the *Torque Controller* as inputs, converts them to appropriate PWM duty cycles and passes this new information forward to the PWM driver. The update frequency is 1600 Hz.

4.3.2 AUTOSAR Basic Software

This section describes briefly some of basic software modules used in this project and what they do. Support and code for these modules are included in Arctic Core and don't have to be written manually. However the modules still have to be configured and connected to each other in the Arctic Studio environment.

Operating system

The operating system contains five different tasks that run with different priorities and frequencies. There is a start up task as well as a service task that runs the basic software. Furthermore three other tasks are connected to the runnables in the SWCs. Highest priority has the service task that makes sure that all background functions run as supposed. The *IMU driver* runs on a task of its own whilst the *Sensor-Actuator Component In SWC*, the *Raw data to state conversion SWC* and the *Ballbot Controller* share one task. Similarly the *Torque Controller* and the *Sensor-Actuator Component Out SWC* share one task that runs at a higher frequency.

ADC Driver

Initialisation and control of the Analog Digital Converter units on the microcontroller is handled by the *ADC Driver*. The driver provides services for enabling, disabling, starting and stopping of conversions [27]. In this project the ADC is used when measuring the internal currents of the motors, these are the signals that are fed back to the *Ballbot Controller* and the *Torque Controller*.

Digital IO Driver

The *Digital IO Driver* handles the digital inputs and outputs from the microcontroller. The driver provides services for writing to and reading from DIO channels (pins), DIO ports and DIO channel groups. All services are synchronous [28]. The quadrature decoders used in this project require two digital signals as inputs, in order to specify their internal settings. These signals are configured as DIO channels.

PWM Driver

The PWM module (Pulse Width Modulation) links PWM channels to a hardware PWM on the microcontroller. The module provides services and functions allowing the user to select the period time of the signal as well as the duty cycle [29]. The input voltages to the motor drivers given by the *Torque Controller* are examples of low pass filtered PWM signals used in this project.

Port Driver

One pin on the microcontroller can for different applications be configured to have different functionalities. For instance, in one project a pin might be used for SPI, PWM or as a General purpose I/O (GPIO) pin. In another project however the user might want to use the pin for CAN or LIN communication instead. Depending on the wanted functionality, different initialisation routines for the pin have to be performed. This is where the Port Driver is used. By configuring a pin for a specific functionality, the Port Driver performs the appropriate initialisation [30].

SPI Driver

The SPI driver (Serial Peripheral Interface) provides services and functions for sending and receiving data on the SPI bus. The SPI bus operates with one master and one or several slaves. The bus uses the following four logical signals:

- SCK: Serial Clock
- MOSI: Master output, Slave input
- MISO: Master input, Slave output
- CS/SS: Chip Select/Slave select

The most common setup is that the SCK, MOSI and MISO signals are shared among all slaves while the CS/SS signal is individual for each slave. When the CS/SS signal goes low (active) the corresponding slave is selected [31]. In this project the processor acts as master and the IMU unit as the only slave. The Basic Software SPI Driver handles the low level communication and provides different functions in C code that can be used by the user. The high level communication concerning what to write and what to read is in this project handled in the *IMU Driver*.

Chapter 5

Evaluation

By performing a lot of different test scenarios the performance of the robot has been evaluated. Each software component has been tested individually as well as all the different hardware components. The behaviour of the whole system with all components connected to each other has also been tested in various experiments. Results from these test scenarios are presented in the following section.

5.1 Results

Initial tests with the whole system as described in the previous chapter did not work as intended. The angles seemed reasonable however the control signals calculated from the LQG controller quickly grew until max speed on the motors were reached. Consequently, the system was reduced in order to make it less complex and easier to troubleshoot. The Kalman filter as well as the reference tracking via integral states were found to introduce unpredictable behaviour and the following tests has been done without those parts. When going back to a more basic approach the robot's performance improved drastically and it could balance for a few seconds before falling. The robot compensates correctly for large movements but has difficulties when trying to apply the small torques required around the equilibrium point. When balancing a robot like this around such a small point with this many degrees of freedom a high precision for small angles is a requirement. Unfortunately this seems to be the biggest problem for this system.

5.1.1 Model and controller evaluation

To be able to evaluate the whole system and its individual software components, data i.e. the internal states, have to be collected and analysed in some way. At first this was done by in software routing the wanted signals to PWM ports on the MCU. This

is an easy way where one can look at the signals with the help of an oscilloscope. It is possible to for example determine that the signs of the signals are correct, e.g. if an angle is increasing or decreasing when tilting the sensors. However the precision is rather inaccurate and the resolution is quite poor. The number of simultaneously examined signals is also limited by the number of available PWM ports.

A better solution is to store the internal data, what the processor sees, on a buffer within the MCU and then send it to be analysed on a laptop.

Firstly a method was developed where the logging started by getting invoked by the user. A limited number of samples were collected and saved on the internal SRAM on the MCU. When the buffer was full the data was sent to the laptop through the JTAG connection used when programing the micro controller. Even though this worked the method was quite poor. Doing it this way the logging takes much time where many different steps have to be taken in both Arctic Studio's debugging environment as well as in MATLAB in order to get satisfying plots of the data. The main disadvantage was though the limited number of samples that could be collected. Due to limitations in the size of the SRAM and even more in the debugging environment only 200 samples for each internal state could be collected. This corresponds to data covering a test of 0.4 s with a sample rate of 500 Hz, this is too few for the test scenarios required.

Thus another more advanced method was developed where the data do not get stored on the SRAM but instead continuously streamed to the laptop over a CAN interface. By using the software CANalyzer the signals can be plotted in real time as well as being saved to a MATLAB data file (.mat). A lot of time was spent on making the CAN communication work. At first built-in AUTOSAR modules were used. This was however very complex and included many extra features that this project did not need. This introduced many extra problems, some of them could not be handled. At the end a manually written CAN driver was written instead that did not use AUTOSAR at all. This method worked perfectly and the the data in the plots below are collected this way.

A number of tests were performed in order to evaluate the behaviour of the implemented software components. Firstly the inputs to the control algorithm were investigated, that is the internal states recreated from the measured sensor values. In Figure 5.1 the result from one of the tests checking the angle θ_x and angular velocity $\dot{\theta}_x$ is shown. In the test the robot was tilted manually, at first to around $-40°/-45°$ and then in the opposite direction to positive $40°/45°$.

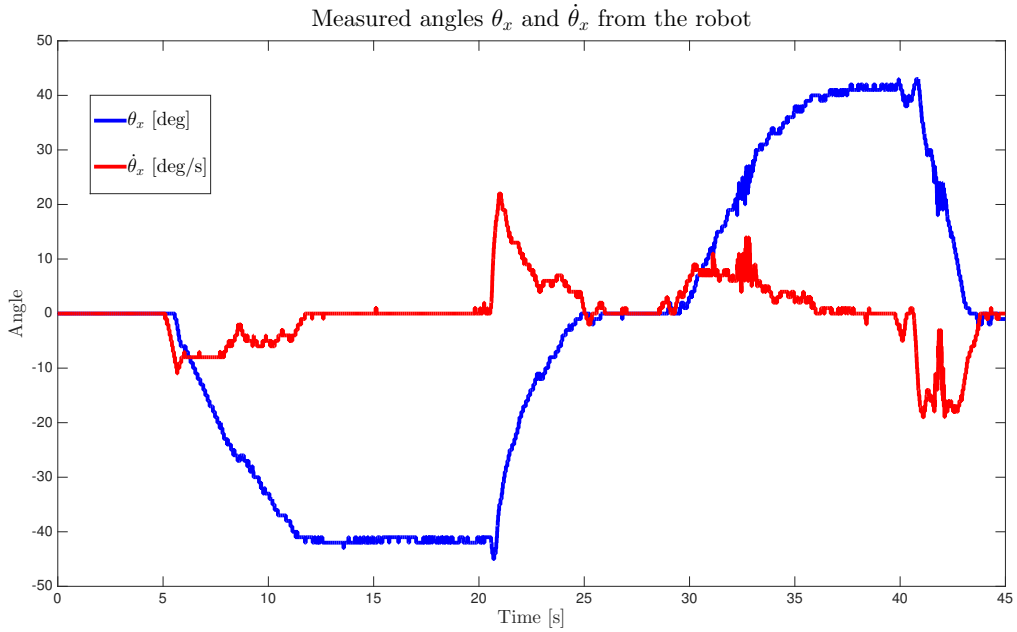


Figure 5.1: Test scenario for the angle θ_x and angular velocity $\dot{\theta}_x$. The robot is tilted approximately 40° in both directions.

As can be seen in Figure 5.1 the recreated state θ_x follows this behaviour very well. Due to limited external measurement equipment it was very difficult to tilt the robot to an exact angle, it had to be an approximation. One can also see that the angular velocity, state $\dot{\theta}_x$, follows the angle θ_x in a reasonable way. The plots are furthermore pretty smooth. This implies that there aren't too much noise on the signals.

In Figure 5.2 the result from a corresponding test scenario for states θ_y and $\dot{\theta}_y$ is shown. The robot was manually tilted to approximately -/+ 40°. Even here the concerned angle and angular velocity show a reasonable behaviour. As expected the noise level is a bit higher on the angular velocity signal. However the noise level is too low to cause any serious problems.

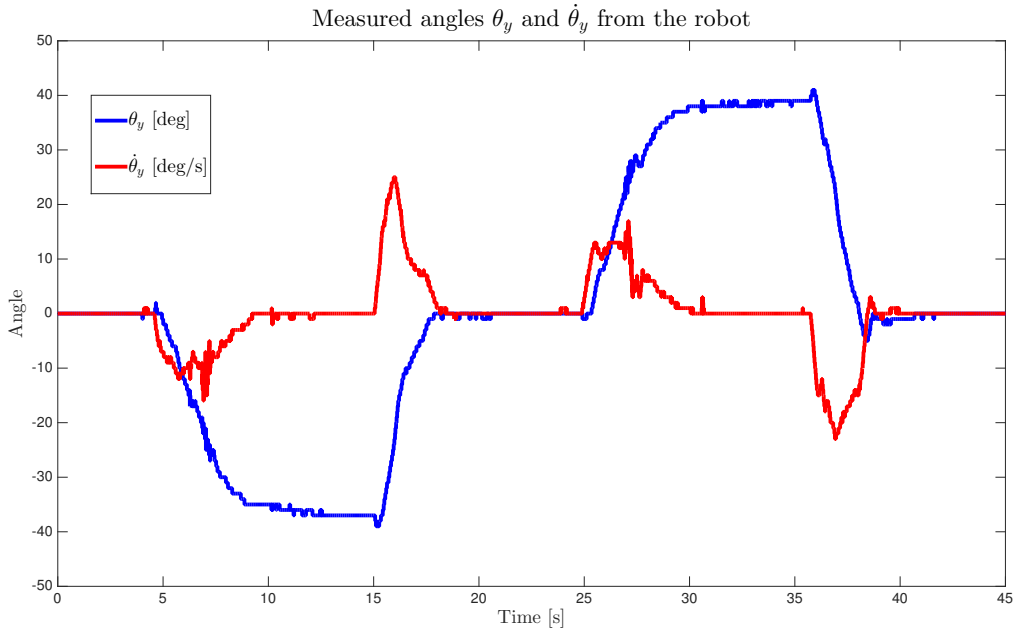


Figure 5.2: Test scenario for the angle θ_y and angular velocity $\dot{\theta}_y$. The robot is tilted approximately 40° in both directions.

In Figure 5.3 a similar test case for the angle θ_z and angular velocity $\dot{\theta}_z$ is shown. In this test the robot was rotated around the vertical axis approximately 90° in both directions. This corresponds with the data in the plot where both states show a reasonable behaviour.

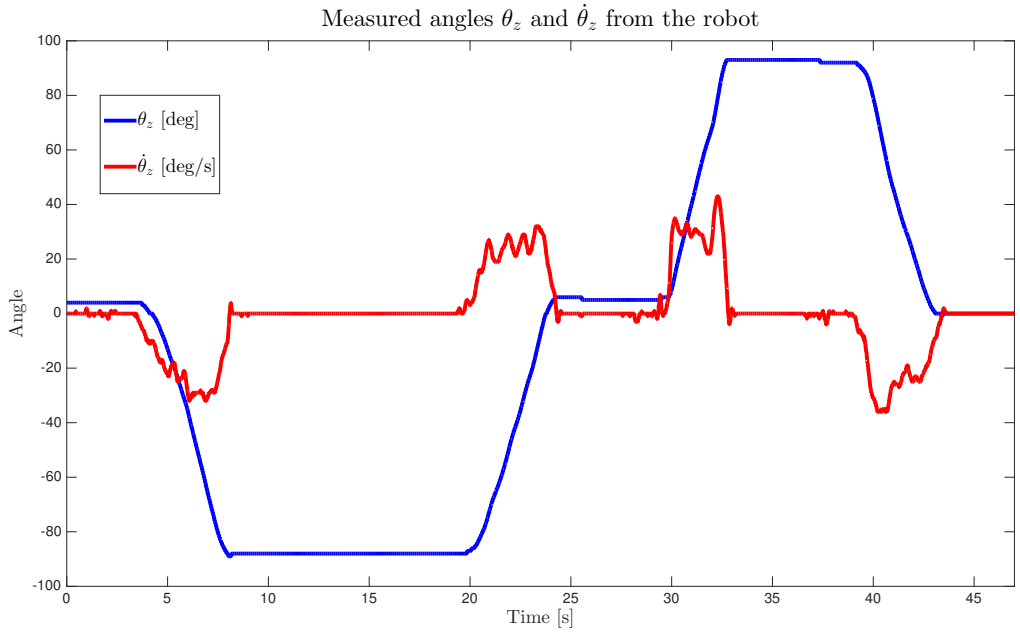


Figure 5.3: Test scenario for the angle θ_z and angular velocity $\dot{\theta}_z$. The robot is rotated approximately 90° in both directions.

The following two figures present results from test scenarios concerning the angles $\phi_x, \dot{\phi}_x, \phi_y$, and $\dot{\phi}_y$. These are the states corresponding to the movement of the ball calculated using the odometry method described in section 3.1.7. In both tests the ball was rotated somewhere between 90° - 100° around the corresponding axis.

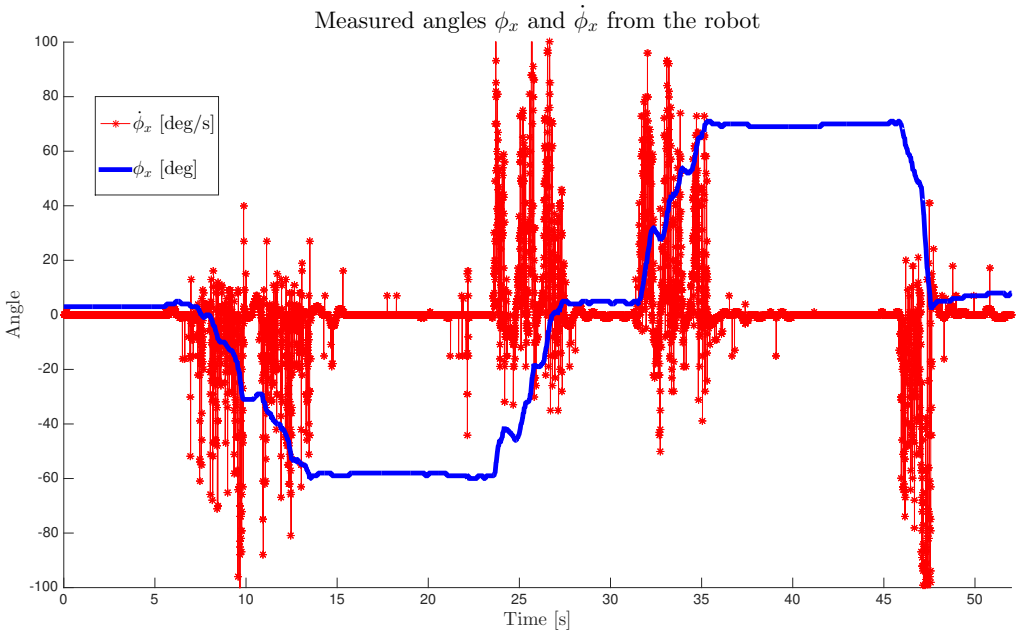


Figure 5.4: Test scenario for the angle ϕ_x and angular velocity $\dot{\phi}_x$. The ball is rotated approximately 90° - 100° around the ball's x-axis.

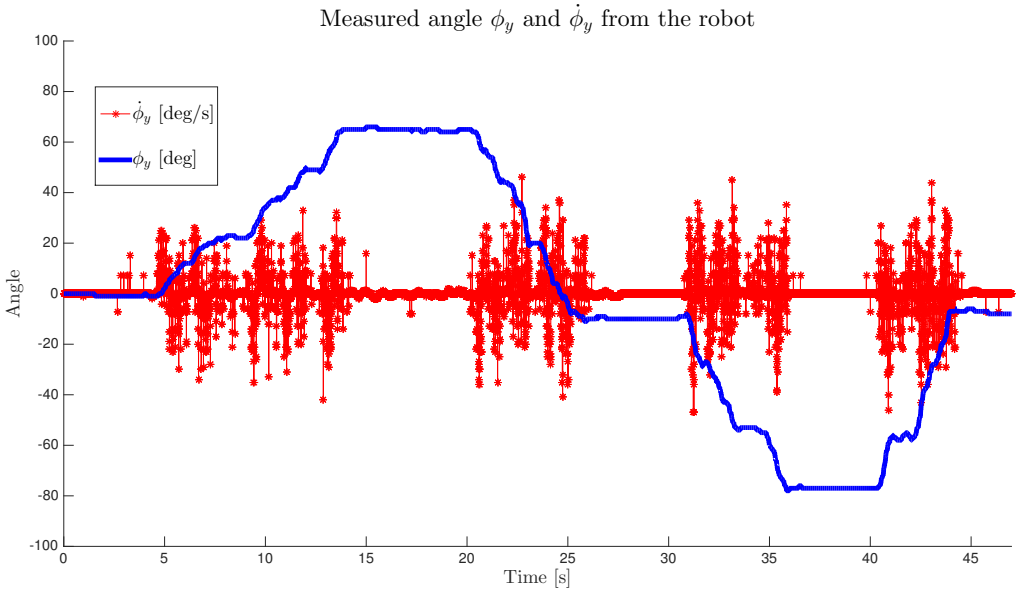


Figure 5.5: Test scenario for the angle ϕ_y and angular velocity $\dot{\phi}_y$. The ball is rotated approximately 90° - 100° around the ball's y-axis.

As can bee seen in both Figure 5.4 and Figure 5.5 the results are not as good as in the previous tests. The estimated angles of the ball's rotation are much lower then the actual value. This error could come from the test setup. When tilting the robot and rotating the ball manually the movement of the robot could possibly get a different behaviour compared to the scenario where it balances by itself. Since the odometry calculations are based on the model of the robot this could introduce some errors in the calculations when it does not behave in the way the model expects. In either way, this behaviour could have a bad influence on the control behaviour. If the controller gets wrong information regarding these states expecting the robot to have moved less than in reality wrong control outputs will most likely be calculated.

One can also notice that the angular velocities of the ball ($\dot{\phi}_x$ and $\dot{\phi}_y$) are very noisy. If these states have almost any weighting at all in the LQR control design the resulting control signal will be very noisy as well.

The next three plots show results from tests where the calculated torque references from the control algorithm are evaluated. These are the references that are sent to the motor drivers and these torque values should ideally be applied on the ball by the motors. When leaning straight towards a specific motor and the control algorithm just wants to stabilise the robot it is expected that this motor stays still while the other two motors applies torque on the ball, in opposite directions against each other.

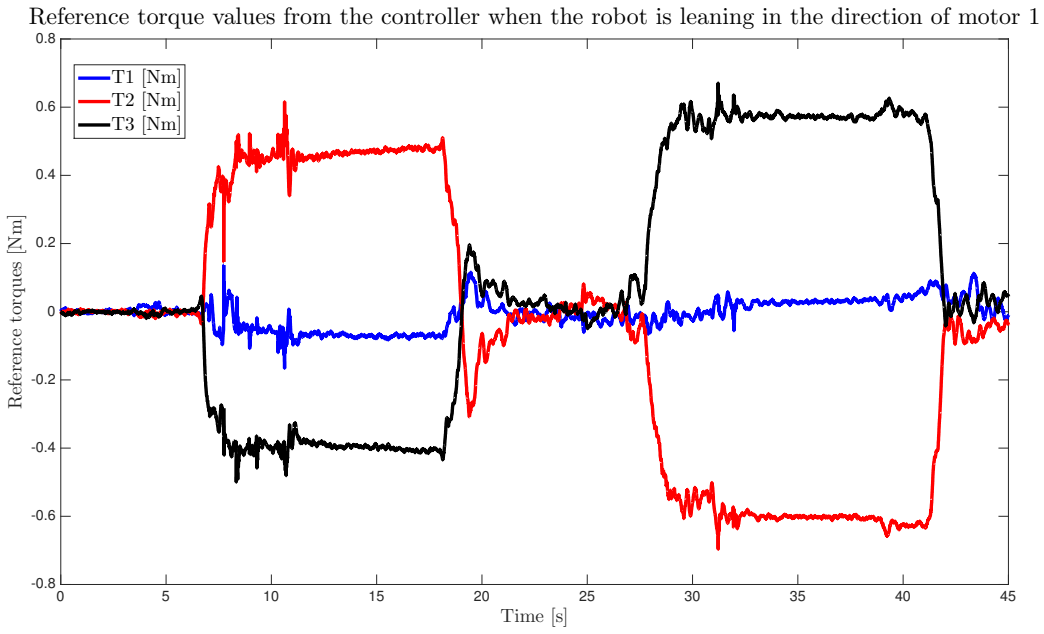


Figure 5.6: Test scenario showing showing the three control signals/torque references from the control algorithm when tilting the robot towards and away from motor 1.

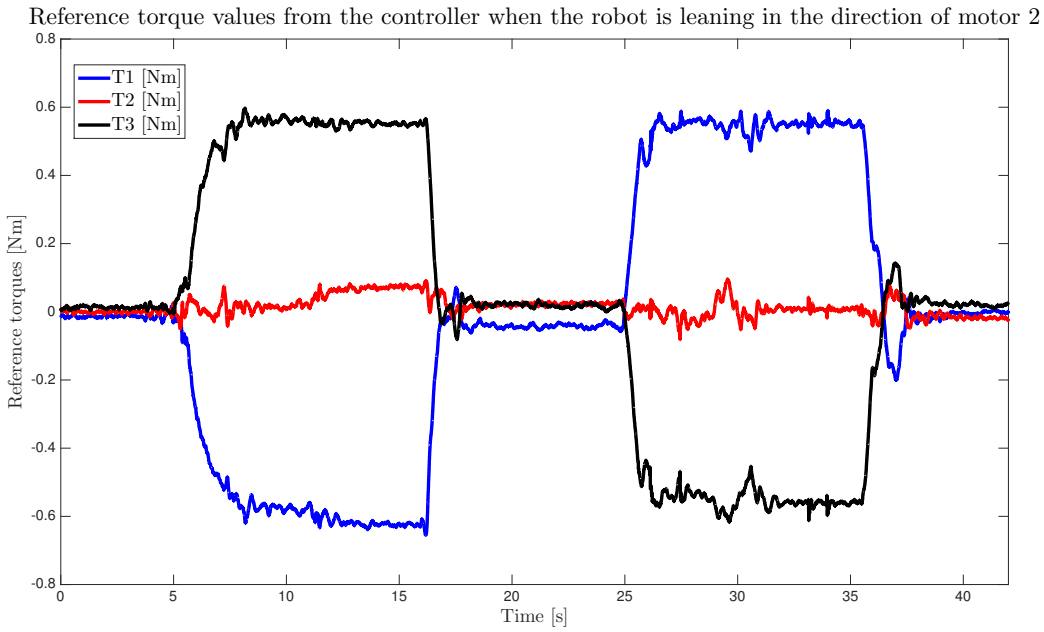


Figure 5.7: Test scenario showing showing the three control signals/torque references from the control algorithm when tilting the robot towards and away from motor 2.

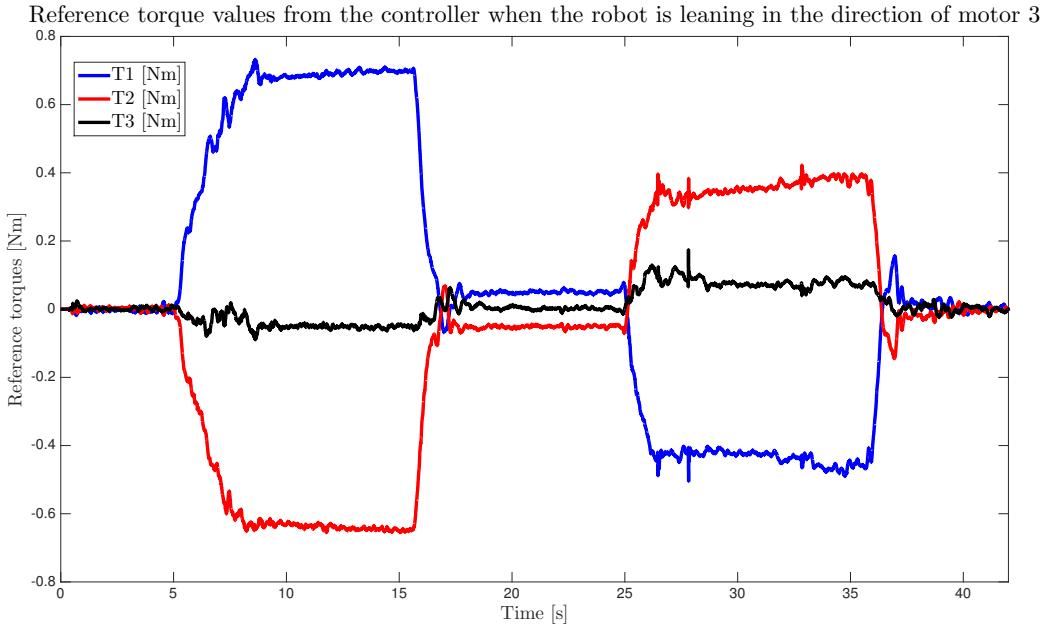


Figure 5.8: Test scenario showing showing the three control signals/torque references from the control algorithm when tilting the robot towards and away from motor 3.

As can be seen in Figure 5.6, Figure 5.7 and Figure 5.8 the control algorithm behaves exactly as anticipated and corresponds to what could be seen in similar simulations (see Figure 3.11). In these tests the weighting on the states $\dot{\phi}_x$ and $\dot{\phi}_y$ was very low in order to not introduce so much noise.

5.1.2 Torque control evaluation

As could be seen in the previous section the recreation of states and calculation of control signals work more or less exactly as intended. The remaining problem is to convert the torque references from the controller into a real applied torque on the ball as described in section 4.3.1. In order to determine how well or bad this module work tests where the measured currents and control signals were investigated. One of tests is presented in Figure 5.9. In this test a step response for one of the motors was performed. Input to the PI-controller was a step with amplitude 0.5 A.

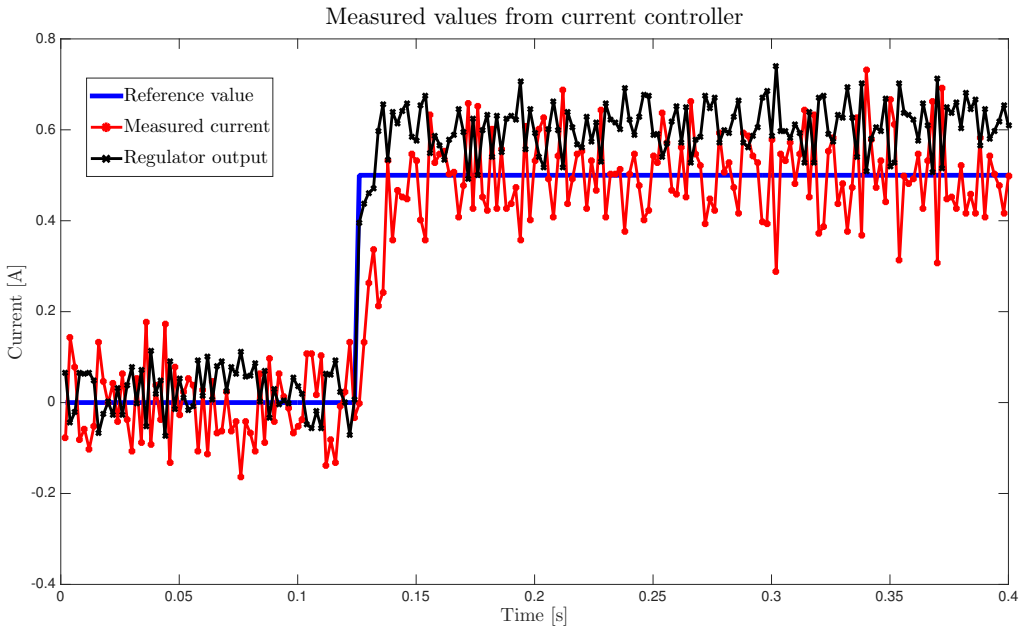


Figure 5.9: Step response with an amplitude of 0.5 A showing measured current and regulator output.

As can be seen in Figure 5.9 the measured current signal is extremely noisy. The measured current has a standard deviation of 0.25 A and a peak-to-peak value of 0.45 A in the region where it should be constant. This is very much concerning the range it should work within. With this robot's motors an applied torque of 0.25 A corresponds to a torque of 0.123 Nm on the ball and 0.45 A to 0.218 Nm. In order to keep the robot

stable around the unstable equilibrium with angles at zero it has to be possible to apply very small torques on the ball as well as not applying any torque when there shouldn't be any. This high noise level makes that impossible to control.

To try solving this problem the measured current signal was filtered in several ways, both in hardware and in software. This helped a bit but not enough. The main problem was that when the noise was filtered to a low enough level the filters introduced a phase shift and time delay such that the controller became to slow to do anything useful. More about the problem with the current measurements are discussed under section 6.

Instead the torque/current controller was finally removed completely. The torque references from the LQR controller were instead mapped to an appropriate size and routed directly to the motor drivers. This can work since the mechanical system is much slower than the electrical. This means that small rapid changes in the electrical signals will be smoothed out such that only the mean value of the signals in a certain interval will influence the behaviour of the mechanical parts. In fact this showed an improved performance. The robot was however still not able to perform the small precise changes required around the equilibrium point in order to balance by itself.

Chapter 6

Discussion

Throughout this project, a ball-balancing robot has been designed, constructed and evaluated. Using the software architecture AUTOSAR, the system runs successfully on the ODEEP platform and the integration of the high-level control algorithms, code-generated from Simulink, have been proved successful. However, the goals set at the start of the project have not all been met since, as mentioned in the previous chapter, the robot does not balance for more than a couple of seconds. This chapter will discuss the outcome of the project, issues with the current design and suggest further development to improve the system.

6.1 Project planning

During the course of the project, it became evident that the initial plan was a bit ambitious given the time at hand for a master thesis project. Not only was there a steep learning curve on different subjects such as AUTOSAR and the MCU oriented programming but the design part was very time-consuming as well and not to mention the troubleshooting. Given the experience of conducting this project the planning would have been different. Allocating more time to troubleshoot as well as the design in Altium and basic software configuration in AUTOSAR would have been better. Also, the mechanical solutions could have been simplified to better fit the time available as well. The decided dimensions of the robot demanded a high motor torque, making it more difficult to find suitable motors and finally in practise, more difficult to control. The three-part actuating design as chosen also results in a more complex system and given the outcome, a simpler design could have been more successful in reaching the final goal even though resulting in a not as elegant final product. As an example, the inverted mouse drive solution as implemented by Hollis in [2] would have been easier to model and control but on the other hand, the robot would not have been able to rotate around its own vertical axis.

6.2 Components and software

In addition to better planning, some of the choices of components and software implementation could have been better. As mentioned in the previous chapter, the measured current was way too noisy to serve as an input to a torque controller. By choosing a current sensor with an appropriate measurement range the resolution could have been increased, making the measurements less sensitive to noise. The currently used sensor has a range of $\pm 10\,\mathrm{A}$, a sensitivity value of $200\,\mathrm{mV/A}$ and $70\,\mathrm{mA}_{rms}$ noise. The motor current turned out to reside within $\pm 1.5\,\mathrm{A}$ which consequently gives very small voltage steps at the sensor output not far from the noise level. Another possible solution to the measurement noise problem would have been to use a shunt resistance and an operational amplifier circuit. Even though such a solution would have meant more components and perhaps inferior performance, it would have been adjustable to fit the motor range by a simple replacement of a few capacitors and resistors.

Even though modular tests of components were made, they could have been done in an even more detailed fashion. As an example, the motor drivers were specified in data sheets and manuals to control the speed of the motor. As it turned out after precious time of troubleshooting the whole system, the drivers did not have any feedback or control of any kind. The speed reference was simple converted to a fixed duty cycle at the output to the motors.

Another decision that should have been made earlier in the project was to exclude logging of internal states via CAN from the AUTOSAR environment and implement it manually. Given a possibility to early on analyse and ensure that the sensors and algorithms produce reasonable values could possibly have led to a more successful final product. Now, this decision was made out of desperation after countless hours trying to integrate and get CAN up and running in AUTOSAR with basically no time left to analyse the data.

6.3 AUTOSAR feasibility

One of the main research points of this thesis, was to evaluate if it is a feasible and well-functioning solution to use an AUTOSAR architecture in a mechatronical platform outside of the automotive industry. Throughout this project, AUTOSAR has proved to be very complex and time-consuming at first. It has a steep learning curve and even though the specifications are extremely well documented the actual implementation of an AUTOSAR system is not. Another problem is that since it is still confined to automotive industry, there is close to no discussions online on forums and blogs as with other open-source software architectures, making it even harder to implement in practise. Also, the flexibility of AUTOSAR comes with a lot of overhead on seemingly simple functionality. Trying to debug the code generated and available from the author-

ing tools often leads to a long chain of function calls and hidden macros. One example is the before mentioned attempt at enabling CAN communication to log internal states, which had to be abandoned after weeks of troubleshooting in favour of writing it manually in C and getting it up and running in only two days. Putting the configuration phase aside, the main idea of platform independence for higher lever tasks is a big advantage. By taking things a step further and using model based design such as in this project is an even greater advantage, possibly reducing complexity and development costs. That was also the part that worked really well, the integration of .arxml-files to Simulink and from there to AUTOSAR compliant C-code worked with a minimum amount of tweaking. It proved to be very powerful many times during the troubleshooting phase to just modify the Simulink model and regenerate the code as opposed to rewrite manually and possible loosing structure by making temporary changes.

6.4 Model validity and control theory

As pointed out in the previous chapter, the robot did not perform nearly as well as in the simulation, being unable to balance for more than a few seconds. Hence it is interesting to discuss what could have been done differently when it comes to modelling and control. The three dimensional model used in the project is complex and required a tremendous amount of computing power in Mathematica. The main motive for using such a complex model is the superior dynamical performance as described in the bachelor's thesis [25]. However, the report also states that the robot is fully functional with a much simpler decoupled two dimensional model. Perhaps the three dimensional model was a bit to ambitious and a simpler two dimensional model would have saved critical time needed to troubleshoot the system.

At the very end of this project, data collection from the robots internal states over CAN were achieved. A very interesting and rewarding point would have been to compare these results to simulations of the system with the same test scenario. Proper adjustments to the model could have been made as well as fine-tuning of the controller, possibly achieving better end results. Also, the transformations between coordinate systems could have been verified. Unfortunately, there was not enough time.

The control algorithm chosen was Linear-Quadratic-Gaussian control, which later where reduced to an Linear-Quadratic-Regulator because of the problems with the Kalman filter. Of course other methods are available, such as Model-Predictive-Control (MPC) or a plain Proportional-Integral-Derivative (PID) controller. LQG is in the case of MIMO-systems far superior to PID. However, it is more mathematically complex and heavily dependent on a good model of the system to control. MPC is perhaps the most sophisticated controller, being able of predict future behaviour, account for actuator limits and a possibility to run closer to the system constraints. With the time at hand for the project, LQG was deemed the most appropriate control method as it is a commonly

implemented method with lots of documentation. Also, it is a well known method for us personally since it has been used in several projects throughout our education.

6.5 Further improvements

Finally, there are some other points of the systems that can be improved further. As mentioned, the Kalman filter as well as the set-point tracking via integral states had to be removed due to unpredictable behaviour. A possible explanation for this is the previously mentioned noisy current measurements as those are also inputs to the filter. The fact that the simulations fail when applying the same amount of noise to the measurements as well supports this claim. Fixing the current measurement and in addition evaluating the model and making appropriate adjustments, perhaps the Kalman filter could be introduced again. This could result in a significantly increased disturbance rejection. Introducing the set-point tracking ability again will enable the robot to move in the plane according to given instructions. Combining this with set-point commands over CAN or wireless communication would expand the possible applications of the robot.

Another area that could be improved further is the calibration of the sensors to have a proper equilibrium point. At the moment, the robot is assumed to be at its equilibrium at system reset and with this method, the equilibrium point is heavily dependent on the fact that the operator holds the robot absolutely upright. Clearly, this is not optimal and the system would benefit from an alternative solution where, ideally, the equilibrium point is found automatically by either an adaptive filter or a predefined movement pattern. The deadband present at the DC motors could also be compensated for to have an appropriate response when applying small torques. This has already been tested to some extent in the present setup, however, it could be developed even further.

As mentioned in Subsection 4.1.3 the IMU includes a dedicated motion processor, DMP. By using DMP to produce the appropriate angles of the robot could decrease the ECU processor load significantly as well as increase the accuracy due to the more sophisticated algorithms available for DMP. Also, this would separate the IMU calculations from the ECU, consequently increasing the data integrity and making it fully platform independent.

Despite all the things that could have been made better during the project, a great amount of parts have successfully been implemented to great satisfaction. A full three dimensional model has been implemented in Simulink and simulations can be run in a 3D-animation environment to run different test scenarios depending on different model parameters, noise levels and sampling frequencies. The algorithms developed work without problems in the simulation environment and can successfully be code-generated into AUTOSAR and runs flawlessly. All external components such as sensors and actuators has been integrated successfully into AUTOSAR and all aids at providing the

control algorithm with sufficient data. Considering the time available this is not such a bad result after all and the amount of knowledge obtained through the project has aided in personal development for both of us.

Chapter 7

Conclusion

A mechatronical system in the form of a ball-balancing robot has been designed, simulated and constructed. The goal of the system was to serve as a test platform of the model based design capabilities of the software architecture AUTOSAR. The design uses three equidistantly mounted omniwheels and a storey-like body accommodating the ECU, IMU and other electronic components. A three dimensional mathematical model of the system was derived and used in simulations to develop a Linear-Quadratic-Regulator capable of balancing the system and following set points in the plane. Investigations of different model parameters' impact on the system and decisions regarding the final system to be constructed were made. After successful simulations of the complete closed-loop system, AUTOSAR compliant C-code of the control algorithm was generated from Simulink. To have a functional interface between the sensor-actuator part and the ECU platform, an expansion card was designed in Altium Designer incorporating the additional control electronics. In addition to the electrical and mechanical construction of the system, the basic software in AUTOSAR have been configured including drivers to connect the control algorithm to hardware. Finally, an internal logging module were implemented which communicates via CAN to debug the system.

Even though the construction and software development were finalised and the concept fully functional, the robot was not able to balance for more than a few seconds. The reasons for this were mainly poor choices of hardware in combination with not enough time allocated to troubleshoot in a structured way. Given the outcome, a less ambitious and time-consuming mechanical design could have been chosen to reach the final goal of a fully balancing robot. Also, important functions for debugging the system such as the CAN logger of the internal states, should have been implemented outside of the AUTOSAR environment at a very early stage to save time.

Regarding the main focus of the thesis, AUTOSAR suitability in model based design, the following conclusions can be made. Setting up AUTOSAR and configuring the basic software proved to be a very time consuming and challenging task due to the steep learning curve, complex design and lack of implementation documentation. Hence,

the flexibility of the AUTOSAR hardware independent structure comes at a cost of increased initial work to get the system up and running. On the other hand, the integration of Simulink and AUTOSAR enables a simple and efficient way of implementing complex control algorithms on ECUs using model based design, possibly saving valuable resources and time. Hence, a model based design development using AUTOSAR and Simulink can certainly pay off, especially on systems using a great amount of complex control algorithms which are easy to describe on a higher abstraction level such as in Simulink in contrast to implementing it in a lower level programming language.

References

[1] R. Hollis, "Dynamic balancing mobile robot," Dec. 7 2010, uS Patent 7,847,504. [Online]. Available: http://www.google.com/patents/US7847504

[2] T. Lauwers, G. Kantor, and R. Hollis, "A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, May 2006, pp. 2884–2889.

[3] M. Kumagai and T. Ochiai, "Development of a robot balancing on a ball," in *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, Oct 2008, pp. 433–438.

[4] A. Sabanovic and K. Ohnishi, *Motion Control Systems*. Wiley, 2011. [Online]. Available: http://books.google.se/books?id=1aS_OHD9PkYC

[5] M. Araki, "PID control," *Control Systems, Robotics and Automation*, vol. 2, pp. 1–23, 2002.

[6] B. Lennartson, *Reglerteknikens grunder*. Lund, Sweden: Studentlitteratur AB, 2000.

[7] T. Glad and L. Ljung, *Control theory*. London, United Kingdom: CRC press, 2000.

[8] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008, http://www.cds.caltech.edu/ murray/amwiki/Main_Page. [Online]. Available: http://press.princeton.edu/titles/8701.htm

[9] R. E. Kalman, "Contributions to the theory of optimal control," *Bol.Soc.Mat.Mexicana*, vol. 5, pp. 102–119, 1960.

[10] The MathWorks, Inc. (2014) Linear-quadratic (LQ) state-feedback regulator for discrete-time state-space system. [Online]. Available: http://se.mathworks.com/help/control/ref/dlqr.html

[11] R. E. Kalman and R. S. Bucy, "New results in linear filtering and prediction theory," *Trans. ASME, Ser. D, J. Basic Eng*, pp. 95–108, 1961.

[12] The MathWorks Inc. (2014) Kalman filter design, Kalman estimator. [Online]. Available: http://se.mathworks.com/help/control/ref/kalman.html

[13] H.-J. Lee and S. Jung, "Gyro sensor drift compensation by kalman filter to control a mobile inverted pendulum robot system," in *Proceedings of the 2009 IEEE International Conference on Industrial Technology*, ser. ICIT '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/ICIT.2009.4939502

[14] AUTOSAR. (2014) Basics. [Online]. Available: http://www.autosar.org/about/basics/

[15] ——. (2014) Technical Overview. [Online]. Available: http://www.autosar.org/about/technical-overview/

[16] ——. (2014) Software Component. [Online]. Available: http://www.autosar.org/about/technical-overview/software-component/

[17] ——. (2014) Virtual Functional Bus. [Online]. Available: http://www.autosar.org/about/technical-overview/virtual-functional-bus/

[18] ——. (2014) RTE. [Online]. Available: http://www.autosar.org/specifications/release-41/software-architecture/rte/

[19] ——. (2014) AUTOSAR Basic Software. [Online]. Available: http://www.autosar.org/about/technical-overview/ecu-software-architecture/autosar-basic-software/

[20] A. Hughes and B. Drury, *Electric Motors and Drives: Fundamentals, Types and Applications*. Elsevier Science, 2013. [Online]. Available: https://books.google.se/books?id=jjuTYtKokc8C

[21] B. Mulgrew, P. Grant, and J. Thompson, *Digital Signal Processing, Concepts and Applications*. PALGRAVE MACMILLAN, 1999.

[22] E. Ramsden, *Hall-Effect Sensors: Theory and Application*. Elsevier Science, 2011. [Online]. Available: https://books.google.se/books?id=R8VAjMitH1QC

[23] National Instruments Corp. (2013) Using Quadrature Encoders with NI X Series DAQ Devices. [Online]. Available: http://www.ni.com/white-paper/14964/en/

[24] InvenSense Inc., "MPU-6000 and MPU-6050 Product Specification Revision 3.4," Sunnyvale, USA, 2012.

[25] P. Fankhauser and C. Gwerder, "Modeling and Control of a Ballbot," Bachelor's Thesis, ETH Zürich, Zürich, Switzerland, 2010.

[26] ArcCore. (2014) Products. [Online]. Available: http://www.arccore.com/products/

[27] AUTOSAR. (2014) Specification of ADC Driver. [Online]. Available: http://www.autosar.org/fileadmin/files/releases/4-1/software-architecture/peripherals/standard/AUTOSAR_SWS_ADCDriver.pdf

[28] ——. (2014) Specification of DIO Driver. [Online]. Available: http://www.autosar.org/fileadmin/files/releases/4-1/software-architecture/peripherals/standard/AUTOSAR_SWS_DIODriver.pdf

[29] ——. (2014) Specification of PWM Driver. [Online]. Available: http://www.autosar.org/fileadmin/files/releases/4-1/software-architecture/peripherals/standard/AUTOSAR_SWS_PWMDriver.pdf

[30] ——. (2014) Specification of Port Driver. [Online]. Available: http://www.autosar.org/fileadmin/files/releases/4-1/software-architecture/peripherals/standard/AUTOSAR_SWS_PortDriver.pdf

[31] Motorola. (2000) SPI Block Guide. [Online]. Available: http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf