ISSN 0280-5316
ISRN LUTFD2/TFRT--5897--SE

Development of a ball balancing robot with omni wheels

Magnus Jonason Bjärenstam
Michael Lennartsson

Lund University
Department of Automatic Control
March 2012

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
March 2012

Document Number
ISRN LUTFD2/TFRT--5897--SE

Author(s)
Magnus Jonason Bjärenstam
Michael Lennartsson

Supervisor
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Development of a ball-balancing robot with omni-wheels (Bollbalanserande robot med omnihjul)

Abstract
The main goal for this master thesis project was to create a robot balancing on a ball with the help of omni wheels. The robot was developed from scratch. The work covered everything from mechanical design, dynamic modeling, control design, sensor fusion, identifying parameters by experimentation to implementation on a microcontroller. The robot has three omni wheels in a special configuration at the bottom. The task to stabilize the robot is based on the simplified model of controlling a spherical inverted pendulum in the xy-plane with state feedback control. The model has accelerations in the bottom in the x- and y-directions as inputs. The controlled outputs are the angle and angular velocity around the x- and y-axes and the position and speed along the same axes.The goal to stabilize the robot in an upright position and keep it located around the starting point was successfully achieved.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-63

Recipient's notes

Security classification

http://www.control.lth.se/publications/

Acknowledgement

We would like to thank our supervisor Anders Robertsson who has been very helpful and supporting all our ideas, Rolf Braun for his great assistance in hardware issues and building (and repairing) the robot and Leif Andersson for helping us with various computer problems.

Magnus & Michael

Contents

Chapter 1

Introduction

The goal of this Master Thesis was to build and stabilize a robot balancing on a ball, inspired by a Japanese project [1]. The authors' background is in mechatronical engineering and therefore this project was a suitable challenge.

The robot consists of three omni wheels in a special configuration standing on a ball which gives it inverse pendulum dynamics. The robot is stabilized by rotating the wheels which makes it move in the xy-plane.

First the kinematics of omni wheels was investigated by studying different mounting configurations on platforms moving on the ground [2]. A Lego robot was built to verify and visualize the kinematics and special properties of omni wheels.

Then a model of an inverted pendulum was developed in parallel with the kinematics of a ball actuated by three omni wheels. The inverted pendulum model in the xy-plane was developed in Dymola and exported to Matlab to perform state feedback controller design. The designed controller was then imported back into Dymola for simulation and visualization.

The model of the inverted pendulum has eight states. The states are angle, angular velocity, position, and velocity along the x- and y-axes. State feedback requires measurements from all states. The angles are estimated with sensor fusion done by a complementary filter combining gyroscope and accelerometer readings. The velocities are obtained by using motor readings and inverse kinematics which are integrated to get the positions.

The implementation was done on an Arduino microcontroller board.

Chapter 2

Hardware

In this chapter the hardware used and how it is setup is covered.

2.1 Mecanum wheel

The mecanum wheel, also called the Ilon wheel, was invented by the Swedish inventor Bengt Ilon in 1973 when he worked at the Swedish company Mecanum AB. The mecanum wheel is a conventional wheel with a series of rollers connected with an angle to the circumference. The axes of rotation for the rollers are usually in 45 degree angle to the circumference of the mecanum wheel, see Figure 2.1. This configuration of rollers enables the mecanum wheel to move in both the rotational and the lateral direction of the wheel.

On a platform with two mecanum wheel pairs in parallel there are actually two versions of the mecanum wheel, one with the roller axis mounted +45 degrees with respect to the wheels axis and the other with the roller axis rotated -45 degrees to the wheel axis, see Figure 2.2. One of the mecanum wheels in each pair has the positive angle and the other one has the negative. If that was not the case all of the resultants for the four mecanum wheels would have been parallel and the ability to move in any direction had been lost.

The mecanum wheels are used on platforms where movement in tight and narrow spaces is crucial for example on forklifts inside warehouses [3].

2.2 Omni wheels

Omni-directional wheels also have rollers connected to the circumference like the mecanum wheel, see Figure 2.3. The difference between the mecanum wheel and the omni wheel is that the axes of rotation for the rollers is parallel to the circumference of the wheel instead of 45 degrees as for the mecanum wheel. This design with the rollers enables the omni wheel to move freely in two directions. It can either roll around the wheel axis like a regular wheel

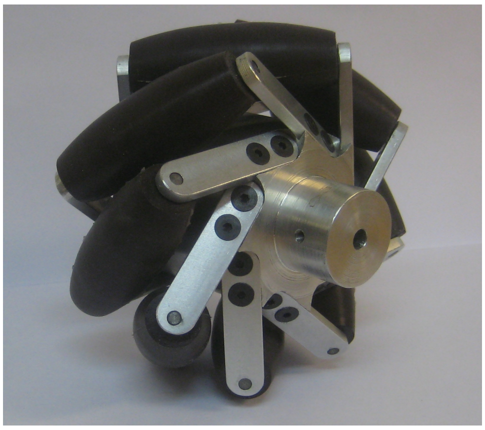

Figure 2.1: Mecanum wheel.

Figure 2.2: The two different types of the mecanum wheel.

or roll laterally using the rollers connected to the circumference or both at the same time.

When the omni wheel is moving the contact point between the ground and the omni wheel will not be directly under the wheel centre at all times as it is for a regular wheel. For example when the omni wheel is shifting between two rollers there are actually two contact points on the ground at the same time. Moving on a flat surface this will make the movement bumpy and not as smooth as for an ordinary wheel. To increase the performance manufacturers have developed omni wheels with two or three rows of rollers placed side by side to bridge the gap between the rollers so the transition when the wheel switches between rollers are smoother, see Figure 2.4. This will give a less bumpy performance but the problem of having two contact points will still occur and now the contact point cannot only drift along the circumference but also laterally.

Further improvements have been made to the omni wheel by a Japanese

Figure 2.3: Picture of omni wheel, single row.

Figure 2.4: Picture of omni wheel, double row.

university [4]. They have bridged the gaps between the rollers by cleverly inserting smaller rollers in the gaps between the larger rollers. This solution gives a smooth transition between the rollers and thereby a smooth movement translational as an ordinary wheel with the properties of an omni wheel.

Commercial applications for the omni wheels are mainly in different kinds of trolleys and conveyor transfer solutions.

2.3 Lego Mindstorms

Lego Mindstorms is a product series from Lego which contains hardware and software that is needed to create own projects such as robots [5]. The hardware consists of Lego bricks for building the structure, gears and wheels, different sensors, motors and the NXT micro computer unit.

The software used is called NXT-G and is a so called "drag-and-drop" based programming language.

Lego Mindstorms are used both by hobbyists and for educational purposes at universities.

2.4 Arduino Mega 2560

The Arduino Mega 2560 microcontroller board is based on the ATmega2560 microcontroller from Atmel, see Figure 2.5 [6]. Arduino is a cheap, open-source, cross-platform solution [7]. The programming language is very easy, well documented and it can be expanded through C++ libraries. There are several boards to choose from, the Arduino Mega 2560 was chosen mainly because it has four UART-modules for communication.

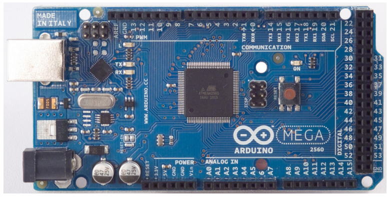

Figure 2.5: The Arduino Mega 2560 board.

2.5 ArduIMU+ V3

To measure the orientation of the robot an IMU (inertia measurement unit) board was used, ArduIMU+ V3 see Figure 2.6. It is developed by 3D Robotics and the DIY Drones community [8]. The board features a 6-axis accelerometer and gyroscope MPU-6000 chip from InvenSense and a 3-axis magnetometer HMC-5883L from Honeywell [9] [10]. An ATmega328P from Atmel running Arduino is used to interface the sensor chips and to run custom code [6] [7]. The preloaded code is open-source.

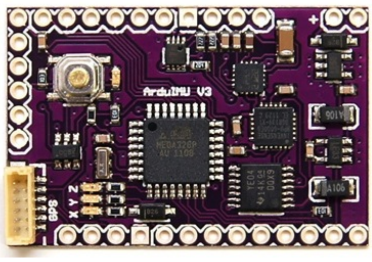

Figure 2.6: The ArduIMU+ V3 board.

2.6 Faulhaber MCDC 3006S & 3257G012CR

The motion controller MCDC 3006S and DC motor 3257G012CR from Faulhaber were used as actuators, see Figure 2.7 [11]. The motor has a maximum torque of 70 mNm. The motion controller can be set in various operation modes such as positioning mode (PID) and velocity mode (PI). All communication with the unit is done by a RS232 interface (serial communication) with up to 115 kBaud. For more information see the manual for the unit [12]. The motor is connected to the wheel using a cog belt with a 3:1 ratio.

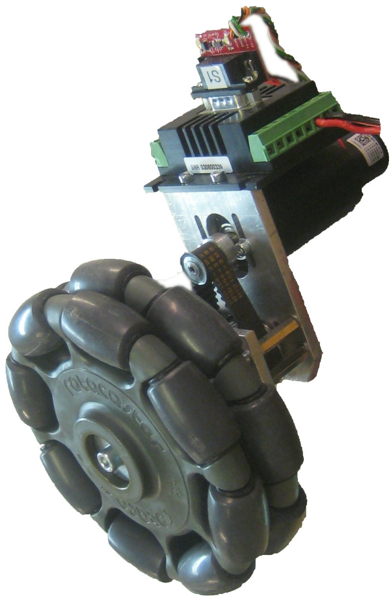

Figure 2.7: The Faulhaber motion controller and motor.

Chapter 3

Theoretical Background

3.1 State feedback control

Assume the process that is supposed to be controlled is described by the state space equation

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{Ax(t)} + \mathbf{Bu(t)} \\ \mathbf{y} &= \mathbf{Cx(t)}\end{aligned} \tag{3.1}$$

The transfer function of the process is then given by

$$\mathbf{Y(s)} = \mathbf{C(sI - A)^{-1}BU(s)}$$

and the poles of the process are given by the roots of the characteristic equation

$$\det(s\mathbf{I} - \mathbf{A}) = \mathbf{0}.$$

Also assume that all the states in the process are measurable and that the system is controllable. Controllable means that the matrix $\mathbf{W_c}$ has full rank, where $\mathbf{W_c}$ is given by

$$\mathbf{W_c} = \begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A^2B} & \cdots & \mathbf{A^{n-1}B}\end{bmatrix}$$

where n is the order of the system. If both these conditions are satisfied the control law

$$\mathbf{u} = \mathbf{l_r r} - \mathbf{Lx} \tag{3.2}$$

can be applied. The vectors $\mathbf{L}$ and $\mathbf{x}$ are given by

$$\mathbf{L} = \begin{bmatrix} l_1 & l_2 & \cdots l_n \end{bmatrix} \text{ and } \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

By combining the state space Equation 3.1 and the control law Equation 3.2 then the closed loop state space equations is given by

$$\begin{aligned} \dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{BL})\mathbf{x} + \mathbf{B}\mathbf{l_r}\mathbf{r} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} \end{aligned} \tag{3.3}$$

where r is the new reference signal. The new transfer function is now given by

$$\mathbf{Y}(\mathbf{s}) = \mathbf{C}(\mathbf{sI} - (\mathbf{A} - \mathbf{BL}))^{-\mathbf{1}}\mathbf{B}\mathbf{l_r}\mathbf{R}(\mathbf{s}).$$

The poles of the closed loop system are the roots of the characteristic polynomial

$$\det(s\mathbf{I} - (\mathbf{A} - \mathbf{BL}))^{-\mathbf{1}}.$$

The vector $\mathbf{L}$ is a design parameter and for a controllable system $\mathbf{L}$ can always be found so that the close-loop poles can be placed as desired.

3.2 Linear Quadratic Optimal Control

Consider a continuous time linear system described by Equation 5.1 and a cost function described by

$$\int_0^\infty \Big(\mathbf{x}(\mathbf{t})^\intercal \mathbf{Q_1}\mathbf{x}(\mathbf{t}) + \mathbf{2}\mathbf{x}(\mathbf{t})^\intercal \mathbf{Q_{12}}\mathbf{u}(\mathbf{t}) + \mathbf{u}(\mathbf{t})^\intercal \mathbf{Q_2}\mathbf{u}(\mathbf{t})\Big)\mathrm{d}\mathbf{t}, \tag{3.4}$$

where $\mathbf{Q_1}$ is positive semi-definite and $\mathbf{Q_2}$ is positive definite. The control law that minimizes the value of the cost is Equation 3.2, where $\mathbf{L}$ is given by

$$\mathbf{L} = \mathbf{Q_2}^{-\mathbf{1}}(\mathbf{B}^\intercal\mathbf{S} + \mathbf{Q_{12}}^\intercal). \tag{3.5}$$

$\mathbf{S}$ is found by solving the continuous time algebraic Riccati equation

$$0 = \mathbf{Q_1} + \mathbf{A}^\intercal\mathbf{S} + \mathbf{SA} - (\mathbf{SB} + \mathbf{Q_{12}})\mathbf{Q_2}^{-\mathbf{1}}(\mathbf{SB} + \mathbf{Q_{12}})^\intercal \tag{3.6}$$

[13].

3.3 Complementary filter

Complementary filter is a technique used to estimate some signal z using two measurements of the signal, x_l and x_h, with low respectively high frequency noise [14][15][16]. The idea is to let the high frequency noise measurement x_h pass through a low-pass filter $F_1 = G(s)$ and the low frequency measurement x_l pass through a complementary filter $F_2 = 1 - G(s)$ which corresponds to a high-pass filter. By adding them together the estimate $\hat{z}$ of the signal is obtained, see Figure 3.1.

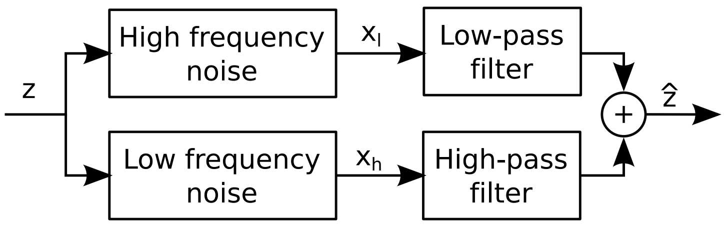

Figure 3.1: A complementary filter.

3.4 Kinematics

In this section the kinematics of omni and mecanum wheels, the test rig, a ball and the robot will be derived.

3.4.1 Kinematics of omni and mecanum wheels

Consider an omni wheel or a mecanum wheel placed on a platform moving on a level ground as shown in Figure 3.2. There are four systems involved. The terrain Σ_0, the vehicle Σ_1, the wheel Σ_2 and the roller Σ_3. The roller is always in contact with the ground at the contact point C. In reality the contact point will drift along the roller axis when the wheel is rotating around the wheel axis. For simplicity the contact point is assumed to always be located below the wheel center A.

The vehicle centre O_1 is chosen for the origin of the coordinate of the analytic description. The x- and y-axes are parallel to the ground. The wheel centre has the x- and y-coordinates a_x and a_y and α is the angle between the extended wheel axis a and the e_{1x} axis. The wheel axis is considered always to be parallel with the ground and therefore the z-component is zero.

$$\mathbf{a} = \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \\ 0 \end{bmatrix} \tag{3.7}$$

is the direction of the vector **a**. The vector **b** is the roller axis and it depends on the angle δ and the angle α. Also here the z-component is zero due to the earlier assumption that the contact point between the roller and the ground is always directly beneath the wheel center and this occurs when the roller axis is parallel to the ground.

$$\mathbf{b} = \begin{bmatrix} \cos(\alpha + \delta) \\ \sin(\alpha + \delta) \\ 0 \end{bmatrix} \tag{3.8}$$

The contact point and the wheel centre, A, are assumed to have the same coordinates because the motion is always parallel to the ground and thus

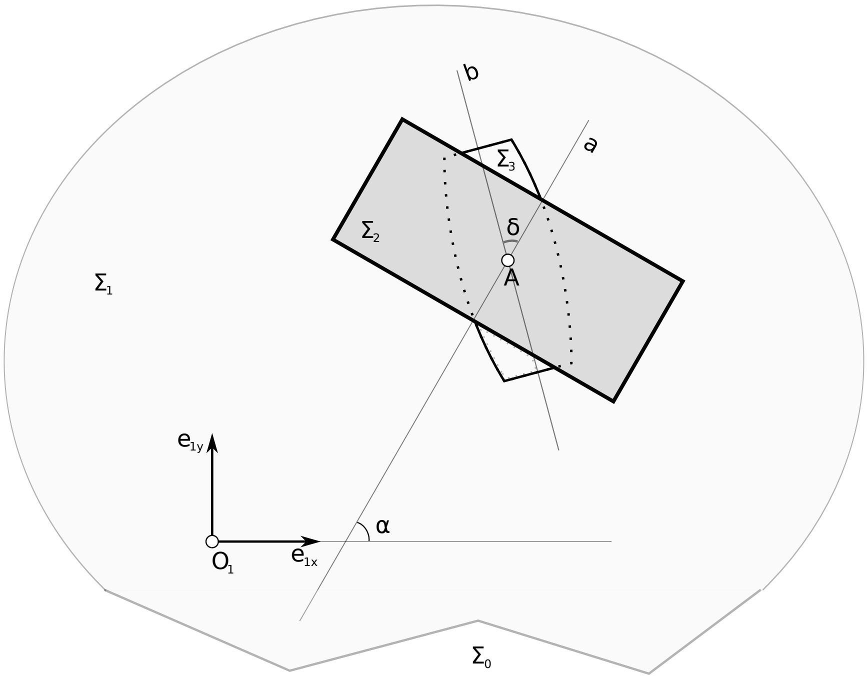

Figure 3.2: One omni or mecanum wheel moving on level ground.

the z-component can be neglected. Thus

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} c_x \\ c_y \end{bmatrix} \tag{3.9}$$

ω is the angular velocity of the motion Σ_1/Σ_0 (vehicle/ground) and $v_{O_1,01} = (v_x, v_y)^\top$ the velocity vector O_1. Then the vectorial velocity of the contact point $C(c_x, c_y)$ relatively Σ_1/Σ_0 is

$$\mathbf{v_{A,01}} = \begin{bmatrix} v_x - \omega a_y \\ v_y + \omega a_x \end{bmatrix} \tag{3.10}$$

The motion Σ_2/Σ_1 (wheel/vehicle) is the rotation around the axis a, $\dot{u}$ is the angular velocity around the wheel axis and r is the wheel radius. The velocity vector at the contact point C is then

$$\mathbf{v_{A,12}} = \mathbf{\dot{u}r} \begin{bmatrix} -\sin(\alpha) \\ \cos(\alpha) \end{bmatrix} \tag{3.11}$$

The motion Σ_3/Σ_2 (roller/wheel) is the rotation around the roller axis **b**. The motion is perpendicular to the vector **b** hence the velocity vector is

$$\mathbf{v_{A,23}} = \lambda \begin{bmatrix} -b_y \\ b_x \end{bmatrix} \tag{3.12}$$

Since the model is assumed to be non slippage the motion Σ_3/Σ_0 (roller/ground) has to be zero. Thus the vector describing the velocity is

$$\mathbf{v_{A,30}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.13}$$

Using the additivity rule for velocities of composed motions we obtain the condition

$$\mathbf{v_{A,01}} + \mathbf{v_{A,12}} + \mathbf{v_{A,23}} + \mathbf{v_{A,30}} = (\mathbf{0}, \mathbf{0})^\top$$

and by substitution of Equation 3.10 - Equation 3.12 leads to the following expression

$$\left.\begin{aligned} v_x - \omega a_y - \dot{u} r \sin(\alpha) - b_y \lambda = 0 \\ v_y + \omega a_x + \dot{u} r \cos(\alpha) + b_x \lambda = 0 \end{aligned}\right\}$$

Elimination of λ gives the following differential equation

$$r\dot{u}(b_x \sin(\alpha) - b_y \cos(\alpha)) - b_x(v_x - \omega a_y) - b_y(v_y + \omega a_x) = 0 \tag{3.14}$$

describing the relations between the angular velocity of the wheel and the movement of the vehicle. Further simplification gives

$$\dot{u} = -\frac{1}{r\sin(\delta)}[\sin(\alpha + \delta)(v_y + \omega a_x) + \cos(\alpha + \delta)(v_x - \omega a_y)] \tag{3.15}$$

This equation gives the angular velocity for the wheel as an output with the x-, y- and z-velocities as inputs to the vehicle. Rewriting this equation gives the final expression

$$\begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \vdots \\ \dot{u}_n \end{bmatrix} = -\frac{1}{r\sin(\delta)}\mathbf{M}\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{3.16}$$

where $\mathbf{M}$ is

$$\mathbf{M} = \begin{bmatrix} \cos(\alpha_1 + \delta) & \sin(\alpha_1 + \delta) & a_{1x}\sin(\alpha_1 + \delta) - a_{1y}\cos(\alpha_1 + \delta) \\ \cos(\alpha_2 + \delta) & \sin(\alpha_2 + \delta) & a_{2x}\sin(\alpha_2 + \delta) - a_{2y}\cos(\alpha_2 + \delta) \\ \vdots & \vdots & \vdots \\ \cos(\alpha_n + \delta) & \sin(\alpha_n + \delta) & a_{1n}\sin(\alpha_n + \delta) - a_1\cos(\alpha_n + \delta) \end{bmatrix}. \tag{3.17}$$

3.4.2 Kinematics of the Test Rig

In this section equations for the kinematics of the test rig are derived. The input is a vector $\boldsymbol{\omega_b}$ that describes the desired angular rotation of the ball and the output will be the required angular velocities of the omni wheels. The ball has free rotational motion around its centre and it is assumed that

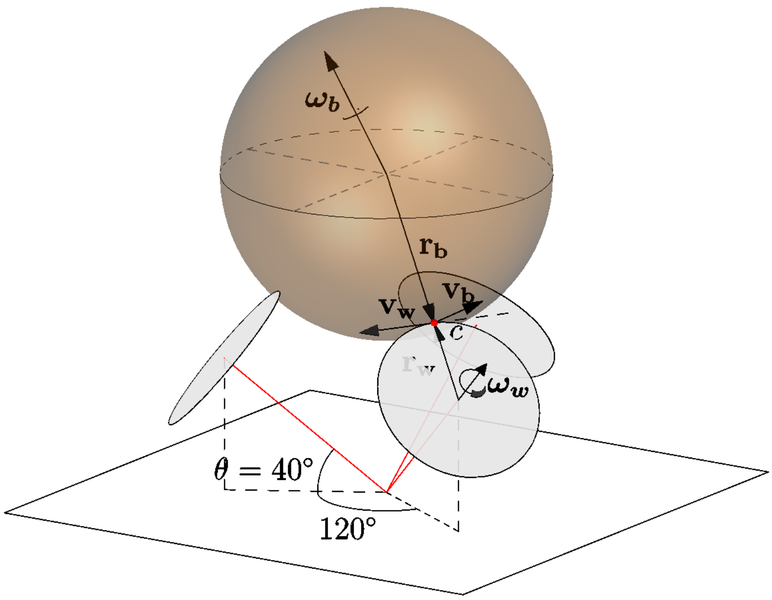

Figure 3.3: Illustrates the vectors and the contact point used in the derived equations for one of the wheels.

the three fixed omni wheels always have contact with the ball, thus the ball has three degrees of freedom. The omni wheels have double rows of rollers and thus in reality the contact point will jump between them when rotating. In order to simplify the kinematic model it is assumed that the wheels are perfectly circular and that there is a single contact point in the middle between the two rows of rollers. A three dimensional Cartesian coordinate system will be used. Consider one of the wheels, see Figure 3.3. The rotational velocity of it can be described by a rotational vector $\boldsymbol{\omega_w}$ along its rotational axis. The circumferential speed $\mathbf{v_w}$ perpendicular to the wheel axis at the contact point c is

$$\mathbf{v_w} = \boldsymbol{\omega_w} \times \mathbf{r_w} \tag{3.18}$$

where $\mathbf{r_w}$ is the radial vector from the wheel centre to the contact point. This is the velocity that can be actuated by this wheel alone. The contact point is in reality on a roller on the omni wheel which have one more degree of freedom because it can rotate around its own axis so in reality the contact point can have a velocity in any direction in a plane with its normal to the surface of the ball at the contact point. That is the same periphery velocity as the contact point on the ball $\mathbf{v_b}$ which is

$$\mathbf{v_b} = \boldsymbol{\omega_b} \times \mathbf{r_b} \tag{3.19}$$

where $\mathbf{r_b}$ is the radial vector from the centre of the ball to the contact point. As mentioned the actuated speed of the contact point on the wheel is not equal to the speed of the contact point on the ball, $\mathbf{v_w} \neq \mathbf{v_b}$, due to the rollers on the omni wheel. The only exception is if the desired rotational axis of the ball is in the same direction as the rotational axis of the wheel. If the speed of the contact point on the ball is projected in the direction of the speed of the contact point on the wheel, equality will be obtained. The direction of the actuated speed $\mathbf{v_w}$ can be calculated as

$$\mathbf{v_{wu}} = \boldsymbol{\omega_{wu}} \times \mathbf{r_{wu}} \tag{3.20}$$

where $\boldsymbol{\omega_{wu}}$ and $\mathbf{r_{wu}}$ are the unit vectors in the direction of the wheel axis and $\mathbf{r_w}$ respectively. The actuated speed can now be expressed as

$$\mathbf{v_w} = \mathbf{v_b}\mathbf{v_{wu}^2} \tag{3.21}$$

A vector can be rewritten as the scalar length multiplied with the unit vector of it, i.e. $\mathbf{r_b} = r_b\mathbf{r_{bu}}$. Using this and combining Eqs. (3.18-3.21) the final equation is formed

$$\omega_w = \frac{r_b}{r_w}\left(\boldsymbol{\omega_b} \times \mathbf{r_{bu}}\right)\left(\boldsymbol{\omega_{wu}} \times \mathbf{r_{wu}}\right) \tag{3.22}$$

Due to the orthogonal orientation of the vectors the equation can be simplified to

$$\omega_w = -\frac{r_b}{r_w}\boldsymbol{\omega_{wu}}\boldsymbol{\omega_b} \tag{3.23}$$

This is valid for an arbitrary number of wheels

$$\begin{bmatrix} \omega_{wi} \\ \vdots \\ \omega_{wn} \end{bmatrix} = -\frac{r_b}{r_w}\begin{bmatrix} \boldsymbol{\omega_{wui}} \\ \vdots \\ \boldsymbol{\omega_{wun}} \end{bmatrix}\boldsymbol{\omega_b} \tag{3.24}$$

The test rig setup will then yield

$$\begin{bmatrix} \omega_{w1} \\ \omega_{w2} \\ \omega_{w3} \end{bmatrix} = -\frac{r_b}{r_w}\begin{bmatrix} 0 & \cos(\theta) & \sin(\theta) \\ -\frac{\sqrt{3}}{2}\cos(\theta) & -\frac{\cos(\theta)}{2} & \sin(\theta) \\ \frac{\sqrt{3}}{2}\cos(\theta) & -\frac{\cos(\theta)}{2} & \sin(\theta) \end{bmatrix}\begin{bmatrix} \omega_{bx} \\ \omega_{by} \\ \omega_{bz} \end{bmatrix} \tag{3.25}$$

3.4.3 Ball translation

The ball is assumed to roll without slip on a horizontal plane and the coordinate system is fixed to the center of the ball with z-axis in the opposite direction of gravity, see Figure 3.4. The rotation of the ball is described by the vector $\boldsymbol{\omega_b}$ and the velocity of the center of the ball is described by the

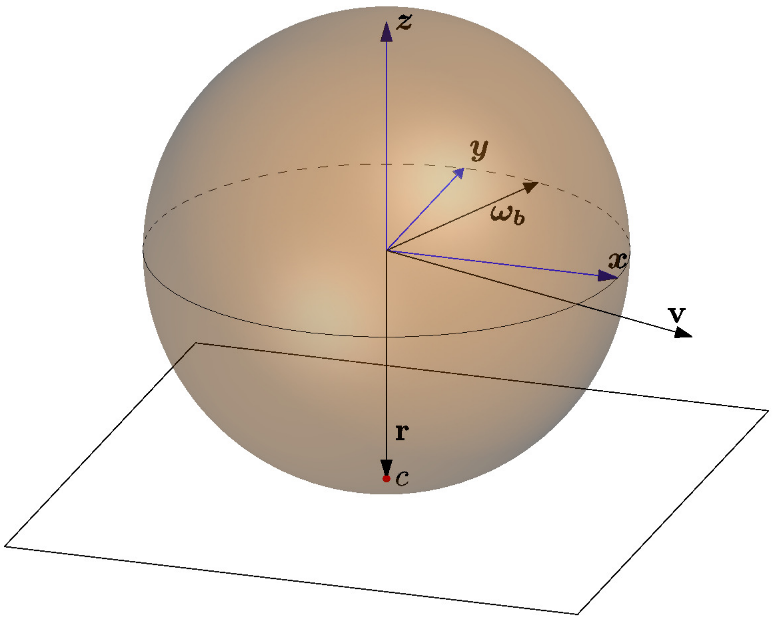

Figure 3.4: Vectors used to describe the translation of the ball on a horizontal plane. Note that the vectors are not scaled correctly.

vector $\mathbf{v}$. The relation between them will now be derived. The speed $\mathbf{v_c}$ at the contact point c is zero since the ground is not moving

$$\mathbf{v_c} = \mathbf{v} + \boldsymbol{\omega_b} \times \mathbf{r} = 0 \tag{3.26}$$

where $\mathbf{r}$ is the vector from the centre of the ball to the contact point. Solving for $\mathbf{v}$ gives

$$\mathbf{v} = -\boldsymbol{\omega_b} \times \mathbf{r} \tag{3.27}$$

It can be rewritten as a matrix product as

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = - \begin{bmatrix} 0 & r_z & -r_y \\ -r_z & 0 & r_x \\ r_y & -r_x & 0 \end{bmatrix} \begin{bmatrix} \omega_{bx} \\ \omega_{by} \\ \omega_{bz} \end{bmatrix} \tag{3.28}$$

where v_i, r_i, and ω_i are elements of $\mathbf{v}$, $\mathbf{r}$, and $\boldsymbol{\omega_b}$ respectively. The assumption that the ball is rolling on a horizontal plane without slip gives restrictions to both $\mathbf{v}$ and $\boldsymbol{\omega_b}$ which must be parallel with the xy-plane and furthermore $\mathbf{r}$ must be perpendicular to it. Moving in a horizontal plane is described by

$$\mathbf{r} = -r_b \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

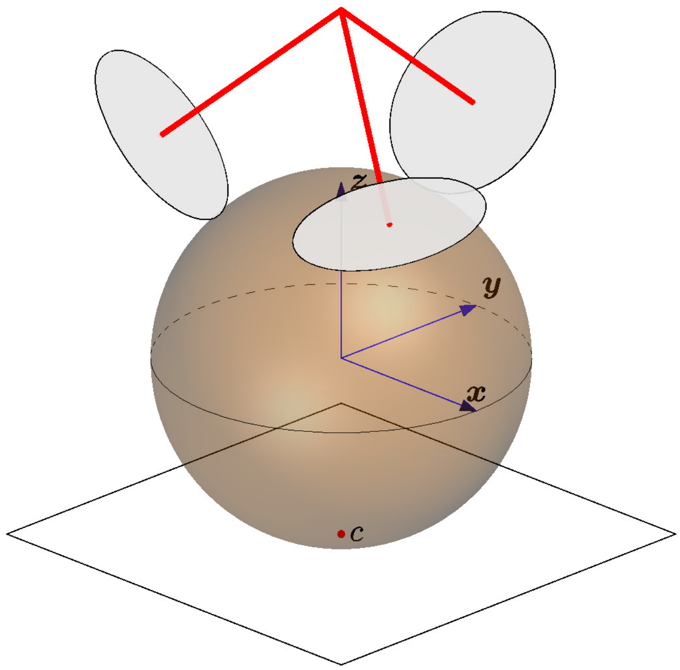

Figure 3.5: The robot standing upright.

where r_b is the radius of the ball. Using this in Equation 3.28 and solving for $\boldsymbol{\omega_b}$ yields

$$\begin{bmatrix} \omega_{bx} \\ \omega_{by} \\ 0 \end{bmatrix} = -\frac{1}{r_b} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{3.29}$$

Since it is impossible to get a velocity in the z-direction, which can be seen in the elements in the third row and column as they are all zeros, it is possible to optionally keep the rotation around the z-axis

$$\begin{bmatrix} \omega_{bx} \\ \omega_{by} \\ \omega_{bz} \end{bmatrix} = -\frac{1}{r_b} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -r_b \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_{bz} \end{bmatrix} \tag{3.30}$$

3.4.4 Robot translation

Combining the kinematics of the test rig (see Subsection 3.4.2) and the ball translation (see Subsection 3.4.3) it is now easy to formulate the equation describing the kinematics of the robot. Equations (3.24) and (3.30) will then

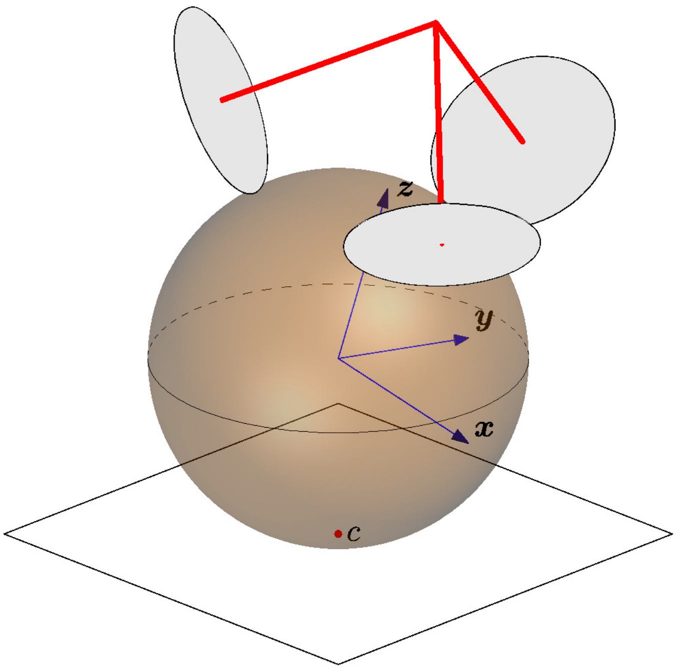

Figure 3.6: The robot tilted.

yield

$$\begin{bmatrix} \omega_{wi} \\ \vdots \\ \omega_{wn} \end{bmatrix} = -\frac{1}{r_w} \begin{bmatrix} \boldsymbol{\omega_{wui}} \\ \vdots \\ \boldsymbol{\omega_{wun}} \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -r_b \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_{bz} \end{bmatrix} \tag{3.31}$$

This is valid as long as the robot is standing upright, see Figure 3.5, but what if it is tilted? Then the positions of the contact points between the ball and the wheels will of course change and thus the model is no longer valid, see Figure 3.6.

The reason for this is due to when the robot is standing upright the world frame is parallel to the robot frame but when the robot is tilted that is no longer the case. Since the robot is only intended to move on a horizontal plane, the easiest way to get a correct model is to change basis of $\boldsymbol{\omega_b}$ from the coordinate system of the ball to the coordinate system of the robot. That is done by multiplying with the inverse rotation matrix $\mathbf{R}^{-1}$ which has the property $\mathbf{R}^{-1} = \mathbf{R}^\intercal$. The error is zero when the coordinate systems are parallel and will of course increase when the robot is tilted. The final expression for the kinematics of the robot with tilt correction is

$$\begin{bmatrix} \omega_{wi} \\ \vdots \\ \omega_{wn} \end{bmatrix} = -\frac{1}{r_w} \begin{bmatrix} \boldsymbol{\omega_{wui}} \\ \vdots \\ \boldsymbol{\omega_{wun}} \end{bmatrix} \mathbf{R}^\intercal \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -r_b \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_{bz} \end{bmatrix} \tag{3.32}$$

The robot will then yield

$$\begin{bmatrix}\omega_{w1}\\ \omega_{w2}\\ \omega_{w3}\end{bmatrix} = -\frac{1}{r_w}\begin{bmatrix}0 & \cos(\theta) & -\sin(\theta)\\ -\frac{\sqrt{3}}{2}\cos(\theta) & -\frac{\cos(\theta)}{2} & -\sin(\theta)\\ \frac{\sqrt{3}}{2}\cos(\theta) & -\frac{\cos(\theta)}{2} & -\sin(\theta)\end{bmatrix}\mathbf{R}^{\intercal}\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & -r_b\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ \omega_{bz}\end{bmatrix} \tag{3.33}$$

The unit vectors of the wheels axes will change sign at the z-elements compared to the test rig since it is "upside down".

Chapter 4

Methodology

4.1 Platforms

Omni and mecanum wheel platforms are platforms that use omni and mecanum wheels in different configurations to achieve omnidirectional movement. In these platforms the wheels have fixed positions and cannot turn like for example an ordinary car with Ackermann steering. A platform with the constraints that it is always parallel to the ground and that all the wheels are always in contact with the ground has three degrees of freedom. These degrees of freedom are movement in the plane and rotations around its own axis. An ordinary car cannot instantly move in any direction or rotate around its vertical axis and is therefore non-holonomic. A platform fitted with either mecanum or omni wheels can do so and is thereby holonomic.

4.1.1 Omni wheel platform

An omni wheel platform is usually fitted with three or four omni wheels. Since the rollers on the omni wheel are parallel to the circumference of the wheel it is important that all the wheels are not parallel with each other, as in this case the ability to move in any direction is lost. It is also important that the wheels are placed so they are not close to being parallel. If so some directions will require a lot more control signal than others to move.

In the case where the platform is fitted with three wheels it is called kiwi drive. When the omni platform is fitted with three wheels the configuration is most often in a shape of a triangle with 120 degrees between the wheel axes that passes through the vehicle centre, see Figure 4.1.

The other configuration often used is fitted with four wheels. The wheels are now placed in pairs that are perpendicular to each other.

Figure 4.2 shows three examples of movement for a platform fitted with kiwi drive. The thick arrow shows the direction of movement from the vehicle centre. The arrow that is parallel with each omni wheel shows in which direction the wheel is rotating. The thicker arrow from the wheel

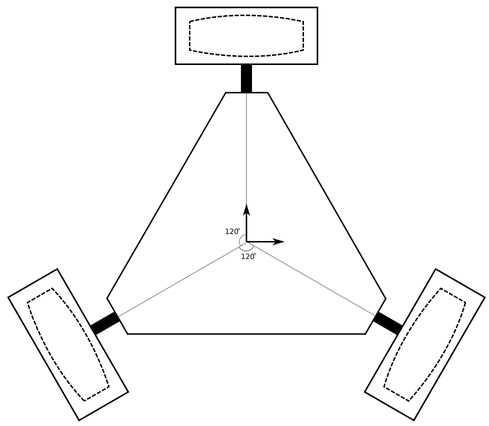

Figure 4.1: Omni-wheel platform fitted with kiwi drive.

illustrates in which direction the wheel is going when rotating, the thinner arrows show the x- and y-components.

In Figure 4.2.a the platform is moving horizontally in the x-direction by rotating wheel number one counterclockwise and wheel number two and three clockwise. The rotation of wheel number one only contributes in the x-direction while wheel number two and number three contribute in both directions. Since the platform is symmetric around the y-axis the components in the y-direction will cancel each other out at the same angular velocity. If the sum of the two x-components from wheel number two and number three times the distance to the centre of the platform in the y-direction is the same as the distance to wheel number one times the velocity the platform will move in the x-direction.

Figure 4.2.b illustrates the platform moving vertically in the y-direction by rotating wheel number two counterclockwise, wheel number three clockwise and wheel number one standing still. The components from wheel number two and number three are canceling each other out in the x-direction and are the same in the y-direction. Wheel number one does not contribute to the platform moving in the y-direction and is therefore standing still.

Figure 4.2.c shows the platform rotating around the platform centre by rotating all the wheels counterclockwise. All the wheels have the same distance to the platform centre and therefore the contribution from each wheel is the same and thus the platform will rotate around its centre.

4.1.2 Mecanum wheel platform

The mecanum platform has four mecanum wheels placed in pairs parallel to each other. Since the roller axes of the mecanum wheels are placed in ± 45 degrees to the circumference of the mecanum wheels there is no

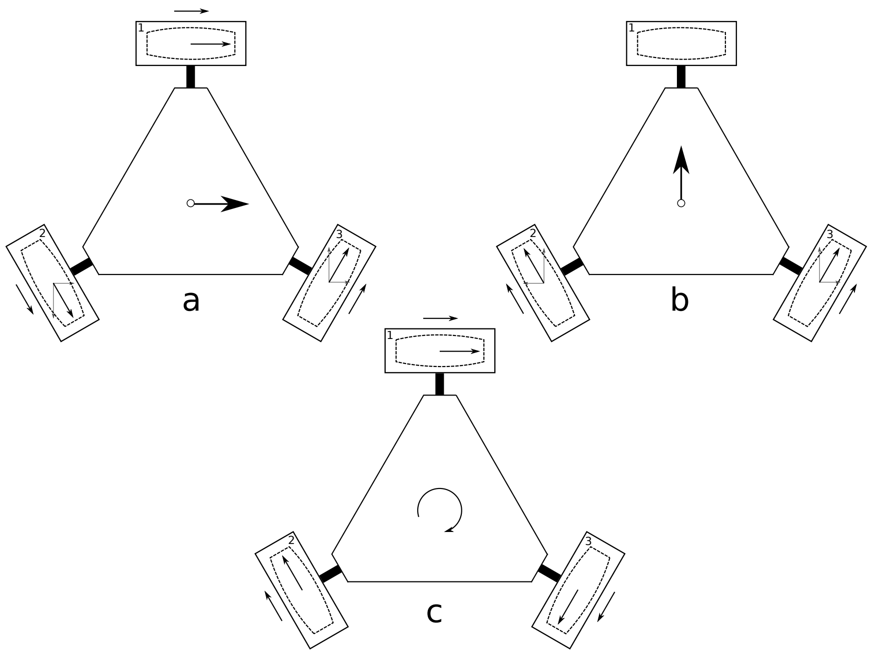

Figure 4.2: Different types of movement for an omni wheel platform

problem placing the wheels in parallel compared to the omni wheels. Now the problem of lost maneuverability occurs when the rollers of the macanum wheels are placed parallel to each other.

Figure 4.4 illustrates four different examples of different movements of the mecanum platform. The notation of the movement for the mecanum wheel platform is the same as for the omni wheel platform.

Figure 4.4.a shows the platform moving in the y-direction by rotating wheel number one and number three clockwise and wheel number two and number four counterclockwise. The components in the x-direction for each

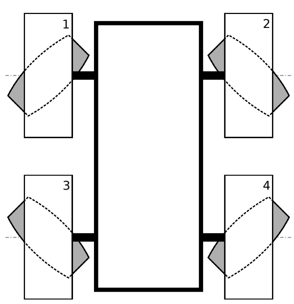

Figure 4.3: Mecanum-wheel platform.

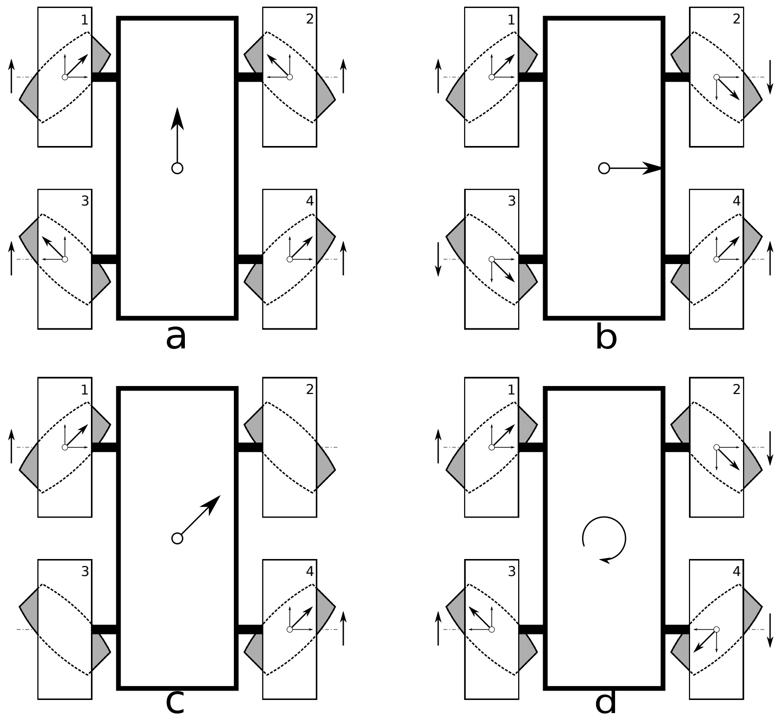

Figure 4.4: Different types of movement for an mecanum wheel platform.

wheel pair will cancel each other out and the y-component for all four wheels are in the same direction and will drive the platform forward.

In Figure 4.4.b the platform is moving in the x-direction. Instead of having the x-components cancel each other out as in Figure 4.4a here the y-components are eliminated and all the x-components contribute in the same direction. This is done by rotating wheel number one and number two in counterclockwise direction and wheel number three and four clockwise.

Figure 4.4.c illustrates the platform travelling at the same speed in the x- and y-direction by only rotating wheel number one counterclockwise and wheel number four clockwise. No components in either the x- and y-direction will be eliminated because wheel number one and four give the same contribution. Wheel number two and number three are standing still. The reason for this is that the roller axes for the mecanum wheel is perpendicular to the movement of the mecanum platform. In other words the roll of the mecanum wheel will rotate around its own axes while the actual wheel is standing still, there will be no friction in that direction.

The Figure 4.4.d illustrates the platform rotating around centre of the platform by rotating all the wheels counterclockwise. By doing this all the x-components cancel each other out and the y-components of the left side are contributing in the opposite direction as the right side, this makes the

Figure 4.5: The Lego platform and a joystick.

platform rotate.

4.1.3 Lego Mindstorms Platform

Lego Mindstorms was used to create an omni wheel platform with kiwi drive, see Figure 4.1. The purpose of the Lego Mindstorms platform was to verify the kinematics of Equation 3.16. Figure 4.5 shows the Lego platform fitted with omni wheels specially designed for Lego and a joystick used for driving the platform. The y-axis is pointing upwards and the x-axis is pointing to the right.

As shown in Figure 3.2 α is the angle between the wheel axis and the x-axis. For this platform $\alpha_{\mathbf{1}} = \mathbf{90°}$, $\alpha_{\mathbf{2}} = \mathbf{210°}$ and $\alpha_{\mathbf{3}} = \mathbf{330°}$. δ is the angle between the wheel axis and the roller axis, for an omni wheel $\delta = \mathbf{90°}$. The length from the wheel centre to the platform centre is L=0.06 meter. Plugging these values into Equation 3.17 results in the following **M** matrix:

$$M = \begin{bmatrix} \cos(180°) & \sin(180°) & -L\cos(180°) \\ \cos(300°) & \sin(300°) & -L\cos(30°)\sin(300°) + L\sin(30°)\cos(300°) \\ \cos(420°) & \sin(420°) & L\cos(30°)\sin(420°) + L\sin(30°)\cos(420°) \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & L \\ \frac{1}{2} & -\frac{\sqrt{3}}{2} & L \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & L \end{bmatrix}.$$

This matrix can be inserted into Equation 3.16 to get the angular velocity of each omni wheel.

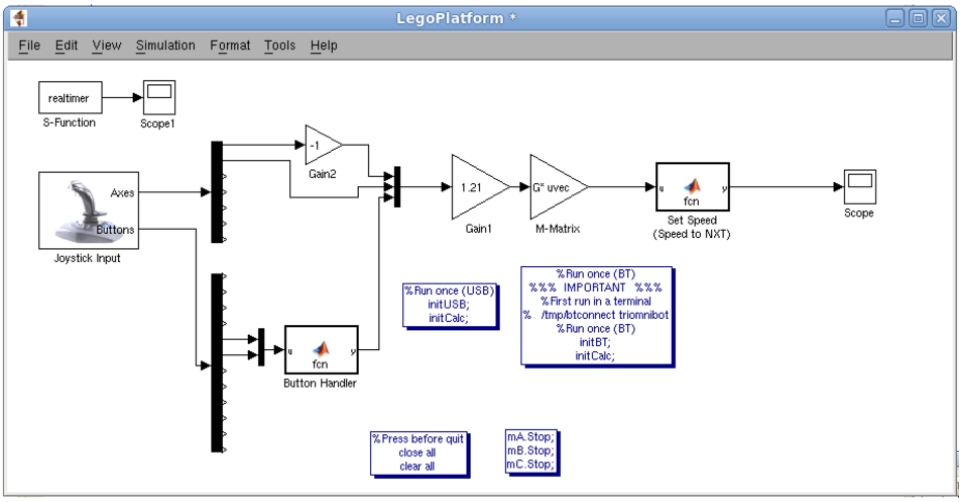

Figure 4.6: Simulink model with the RWTH - Mindstorms NXT Toolbox.

The Lego platform consists, except from the Lego bricks and the omni wheels, of three motors and one Lego Mindstorms NXT brick. The NXT brick is a micro computer that is the brain in Lego Mindstorms projects. In this case the platform is controlled by a special toolbox developed for Matlab/Simulink, the toolbox is called "RWTH - Mindstorms NXT Toolbox for MATLAB". The purpose of the toolbox is to control Lego Mindstroms NXT robots with Matlab/Simulink via a wireless Bluetooth connection or via an USB cable.

Figure 4.6 shows the Simulink model of the platform. From the block *Joystick* input comes three inputs, two inputs are the velocities along the x- and y-axes from the analog stick. The third input is the orientation around the z-axis from two buttons. After the inputs are multiplexed into a vector it is multiplied with *Gain1*. *Gain1* is only a scaling so full throttle on the joystick will give full angular velocities to the motors. The *M-matrix* is the matrix gain calculated above. The *Set Speed* function just sends the calculated angular velocities to the motors. The *realtimer* block, developed here at LTH, enables Simulink to run in real time.

4.2 The Test Rig

This chapter will describe the test rig and its design with three omni wheels with a ball and the kinematics of it will be derived. The first step to investigate the possibilities to build a robot balancing on a ball was to build a test rig. The test rig was built “upside down” with the ball in the air resting on the omni wheels, see Figure 4.7.

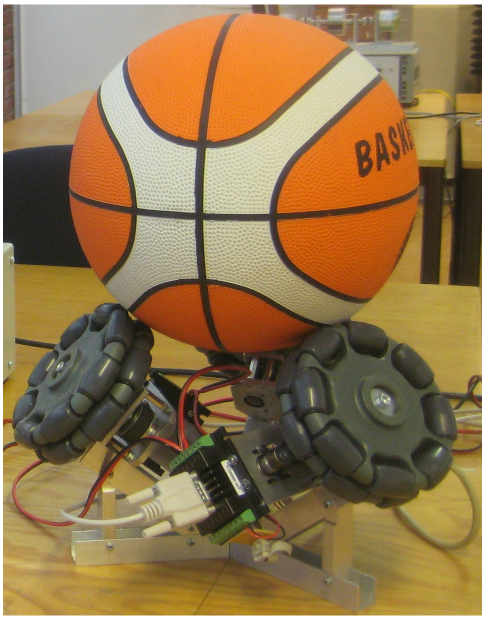

Figure 4.7: Picture of the test rig.

4.2.1 Geometry and design

The omni wheels are evenly spaced around the vertical z-axis with 120 degrees between them and are positioned perpendicular to surface of the ball, see Figure 4.8. That and the angle θ will define the configuration of the wheels. A larger θ will make the wheels come closer together and thus the ball will have a smaller support area. It will also affect the kinematics of the rig, see Section 3.4.2. An angle of 40 degrees was chosen.
The omni wheels used are the "125 mm double omni directional wheel" manufactured by Rotacaster [17]. They are double in the sense that there are double rows of rollers. This gives a smoother run compared to a single row wheel.

4.2.2 Verification

The kinematic model was verified by simple experiments on the test rig. The ball was given a desired rotational speed of 0.25 revolutions per second in different directions, see Figure 4.9. When the ball was assumed to be in steady state after the initial acceleration the rig was recorded with a camera. In the film one can see how long time it takes until a marked point on the ball has completed one revolution. The mean time of three revolutions was used.

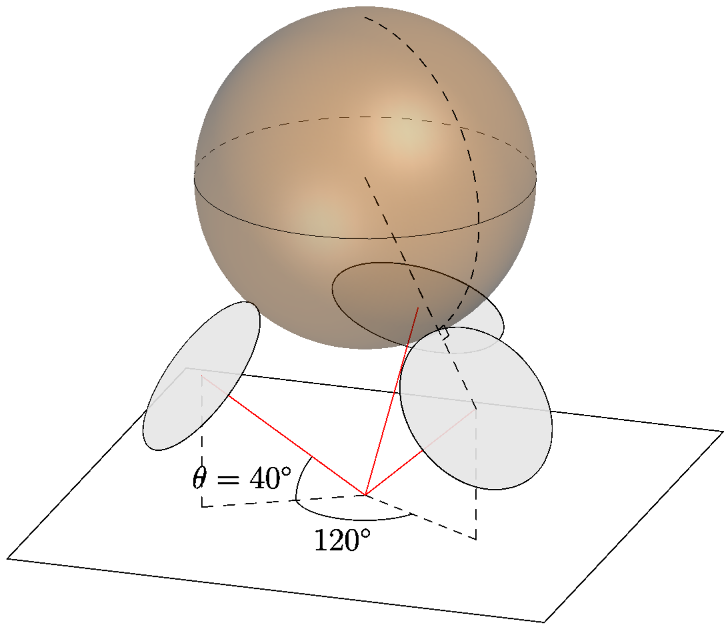

Figure 4.8: The geometry of the test rig, compare with Figure 4.7

4.3 The Robot

In this section the kinematics and design of the robot will first be described. Then the control design of the robot will be derived.

4.3.1 Dynamics of the Robot

To be able to stabilize the robot upon the ball some kind of control design needs to be performed, therefore a model of the actual robot was created. A simplified way of looking at the robot is as an inverted pendulum in the xy-plane. To be able to create a model, the dynamics of the real robot needs to be known and therefore an experiment was performed.

When an inverted pendulum falls from an upright position the curve between the angle and the time is approximately proportional to the curve e^{kt} for some positive constant k. The length of the pendulum determines how fast it will fall. A long pendulum has greater inertia than a short pendulum constructed in the same material and with the same diameter. Therefore a pendulum will fall slower the longer it gets. The curve e^{kt} will increase more rapidly with a greater coefficient k. This means that a greater coefficient will correspond to a shorter pendulum. The goal is to find the coefficient so the curve e^{kt} is as close to the behavior of the real robot as possible.

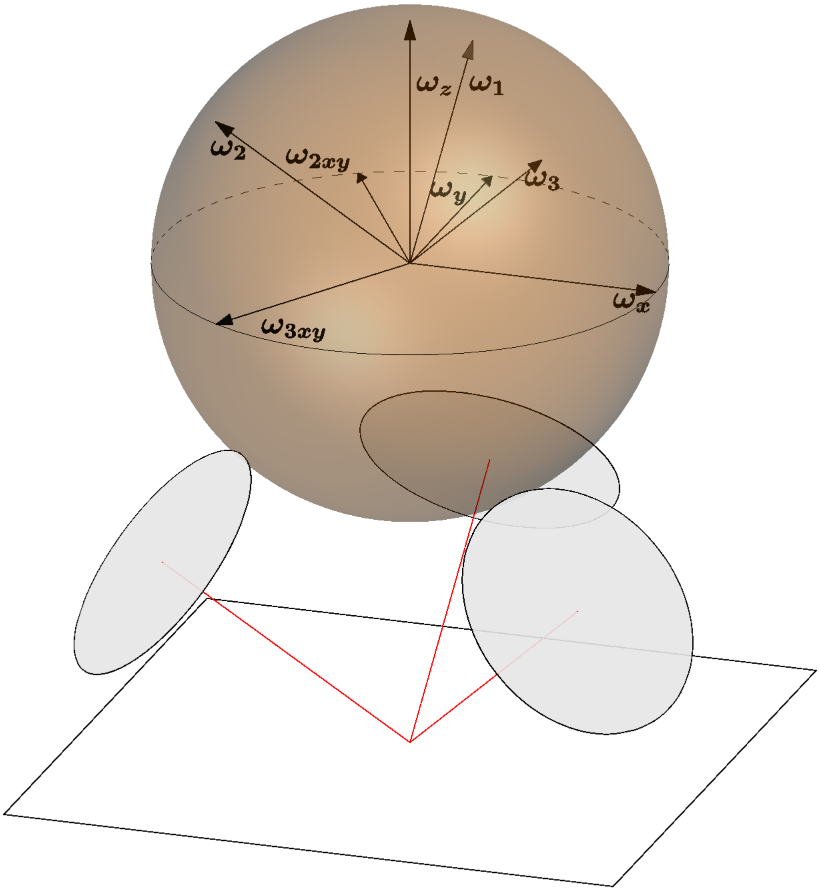

Figure 4.9: The rotational directions used when verifying the kinematic model.

The robot was placed upon the basketball and then tilted a few degrees and then released. The angles and time were measured as the robot was falling freely. The result was plotted in Matlab together with the curve e^{kt} to try to get a good match. Finally the $e^{2.9t}$ was decided as a close result. The result is shown in Figure 4.10.

The dynamics of the robot upon the basketball is now determined. Now a model with similar properties as the real robot can be created.

4.3.2 Dymola Model

The model is created in Dymola with the multi-body package. The Dymola environment uses the open Modelica modeling language [18]. Modelica is a non-proprietary, object-oriented, equation based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, elec-

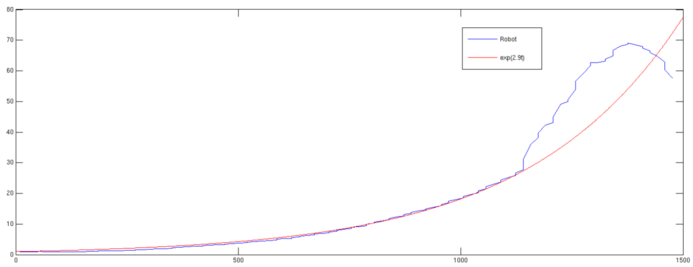

Figure 4.10: Comparing the curve e^{kt} with the actual dynamics of the robot.

tronic, hydraulic, thermal, control, electric power or process-oriented subcomponents [19].

Figure 4.11 shows the Dymola-model of the robot. The solid blue triangles are the acceleration inputs in the x- and y-directions to the pendulum. The smaller white triangles are the outputs of the model. There are four outputs in each direction, the angle and angular velocity around both the x- and y-axes and also the position and the speed along the x- and y-axes. The *prismatic_X* and the *prismatic_Y* components enable the inverted pendulum to move along the x- and y-axes. The prismatics are connected to each other and then to the sensors that are measuring the positions and the velocities in the xy-plane. Inputs to the prismatics are the parts *speed_X* and *speed_Y*. The reason for having speed components instead of accelerate components is that the inputs for the real robot are speed references. The components *speed_X* and *speed_Y* give the prismatics exactly the desired velocities but in reality that is not the case. There is a delay in the motors so the actual velocity does not follow the reference signal perfectly. To compensate that, the first order systems are added to introduce a delay. The components *revolute_X* and *revolute_Y* are also connected with the prismatics and the sensors. The components *revolute_X* and *revolute_Y* enable the pendulum to rotate around the x- and y-axes respectively. The pendulum is connected to the revolutes. The pendulum is modeled with a component called body-cylinder. It is modeled as a solid steel rod with the diameter of 0.1 meter and height of 1.25 meter. The dimensions are set to give similar dynamics as the actual robot. Sensors for measuring the angles and angular velocities are finally connected to the pendulum.

The Dymola model can then be linearized to get the state space model for the robot. The state space model can be exported to Matlab to try out different control designs. Figure 4.12 shows the pendulum with state feedback control.

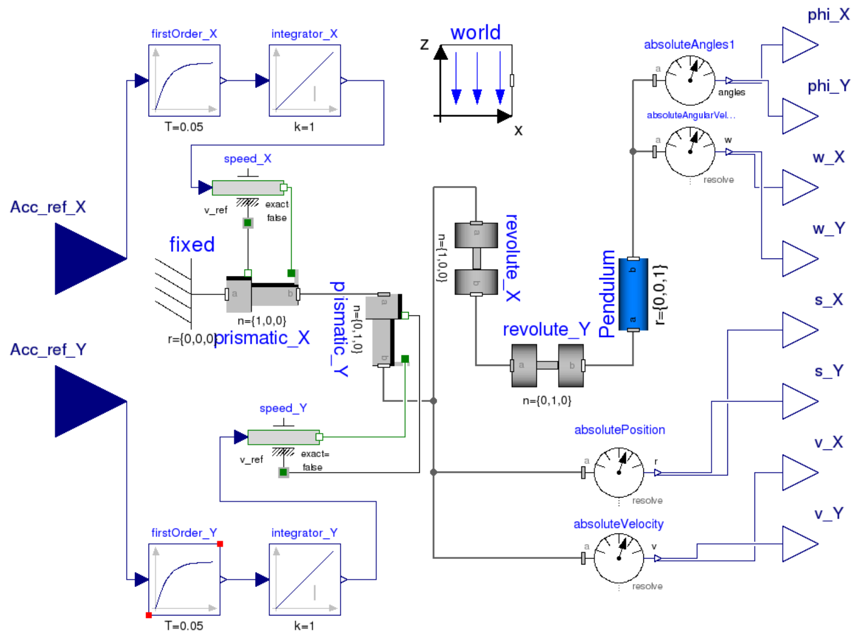

Figure 4.11: Dymola model of the robot.

4.3.3 Robot design

The basic geometry is the same as for the test rig, see Subsection 4.2.1, but now it is turned upside down with the robot standing on the ball. Thus it is no longer a stable system. If the robot's center of mass is not exactly above the contact point between the ball and the ground, the ball will start to rotate and the robot will follow as long as the wheels are not moving and eventually fall over. The robot now also has a rod mounted in the center which will be used to make the system slower, see Figure 4.13.

4.3.4 Implementation

This section will give a brief overview of the implementation on the Arduino Mega 2560 since the source-code is fairly rich commented, see Appendix A. The Arduino Mega 2560 is the heart of the system and does all calculations and communications, see Figure 4.14. All communication is done with UART serial communication at 115200 baud. The sampling interval of the system is driven by the IMU which sends new data every 20 ms. When new IMU data is received query commands are sent to the motion controllers to get updated motor positions. Then all states are calculated and then the

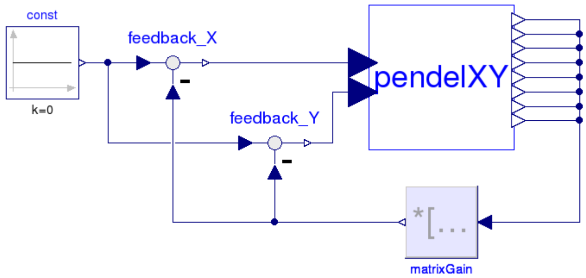

Figure 4.12: Dymola model of the state feedback controlled pendulum.

control algorithm calculates new control signals for the motors. Last at each period, after control signals have been sent to the motors, variables etc are updated and information is optionally sent to a computer for debugging and logging.

Calculating attitude

A complementary filter was used to calculate the attitude of the robot since the sensor fusion algorithm on the IMU did not perform good enough. An accelerometer and a gyroscope are used to estimate the attitude. To get the angle ϕ from an accelerometer is done by measuring the influence of gravity along the x- and y-axis, see Figure 4.15. The angles from the xy-plane around the x- and y-axis are calculated as

$$\phi_x = \arcsin\left(\frac{\ddot{y}}{g}\right) \tag{4.1}$$

$$\phi_y = \arcsin\left(\frac{-\ddot{x}}{g}\right) \tag{4.2}$$

where $\ddot{x}$ and $\ddot{y}$ are the accelerometer measurements and g is gravity. The sign change is due to the right hand rule. A gyroscope measures the angular velocity $\dot{\phi}$ around an axis. The angle is then easily calculated as

$$\phi = \int \dot{\phi} dt \tag{4.3}$$

The angle calculated using accelerometer measurements is very sensitive to disturbances. This is due to any acceleration, for instance made caused by the motors moving the robot, or ordinary measurements disturbances, which will make it deviate from measuring only the gravity will cause an erroneous result. The latter can often be regarded as high frequency noise. The gyro

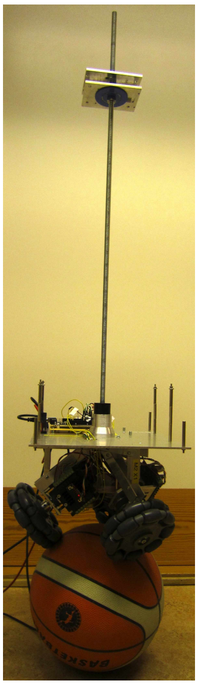

Figure 4.13: Picture of the Robot balancing on a basket ball.

is in general very accurate and will slowly drift over time. This can be regarded as a low frequency noise. Thus to get a good estimate of the angle a complementary filter can be used, where the accelerometer estimation will be low-pass filtered and the gyro estimation will be high-pass filtered and then added together. The filter is implemented as

```
phi = alpha * (phi_old + gyro*dt) + (1-alpha)*acc;
```

where `gyro` and `acc` are readings from the gyroscope and accelerometer respectively, `phi_old` is the previous value of `phi` and `alpha` is a value between zero and one which can be regarded as how much one rely on each sensor, one meaning completely rely on the gyroscope [20]. Since it was not possible to measure the dynamics of the sensors the cross frequency for the filters was chosen by practical experiments.

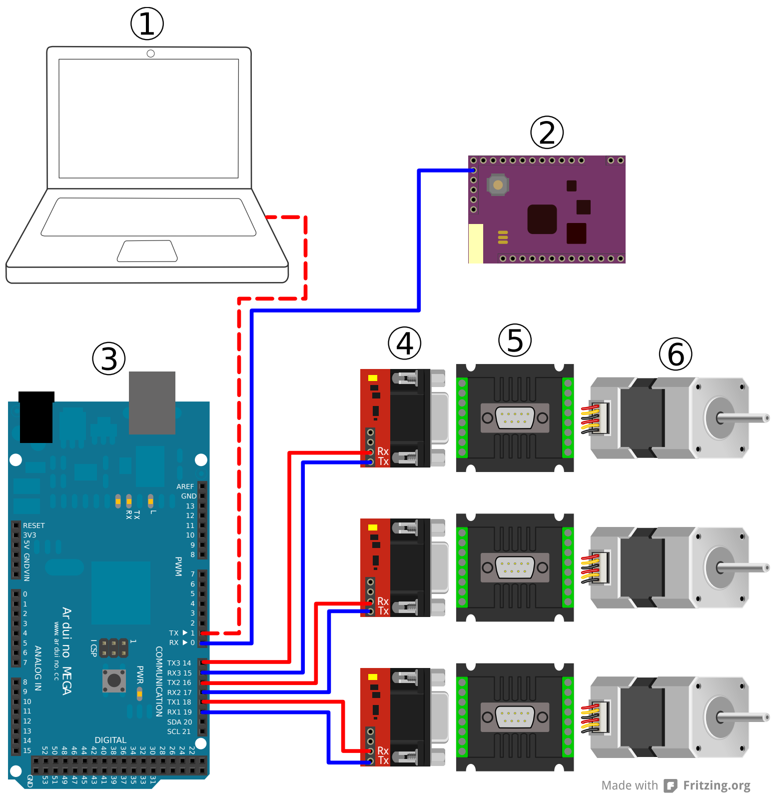

Figure 4.14: Overview of the network between components. Red and blue lines indicates receiving and sending from the Arduino Mega 2560. *1* Computer, *2* IMU, *3* Arduino Mega 2560, *4* TTL to RS-232 converter, *5* Motion controller, *6* Motor. Note that connecting a computer is optional.

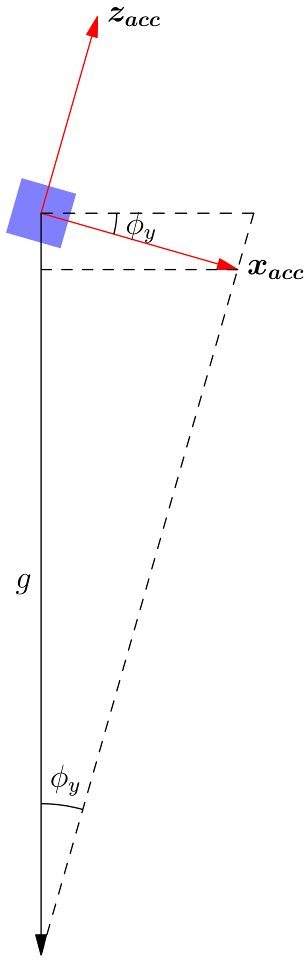

Figure 4.15: Accelerometer tilted $\phi_y = 16°$ around the y-axis. Note that the value of the accelerometer measurement is negative.

Chapter 5

Results

5.1 Lego Robot

The reason for building the Lego Robot was to visualize and get a basic understanding of the possibilities using omni wheels. The robot behavior agrees with the kinematic model. It was easy to verify by driving the robot as a RC car with a joystick. The platform performed as expected and the authors were pleased with the performance and did not investigate it further.

5.2 Test Rig Kinematics

The measured results can be seen in Table 5.1. The largest deviation is less than 3 % which is considered acceptable. The slight deviation might be due to not perfect assembly and construction, rounding error due to the Faulhaber speed controller can only handle integers in rotations per minute and finally the camera has a resolution of 30 frames per second.

$\boldsymbol{\omega_b}$	Time [s]	Deviation [%]
$\boldsymbol{\omega_1}$	4.045	1.12 %
$\boldsymbol{\omega_2}$	4.055	1.38 %
$\boldsymbol{\omega_3}$	4.056	1.39 %
$\boldsymbol{\omega_{1xy}}$, **x**	3.900	-2.50 %
$\boldsymbol{\omega_{2xy}}$	3.999	-0.03 %
$\boldsymbol{\omega_{3xy}}$	3.944	-1.39 %
y	4.044	1.11 %
z	4.078	1.95 %

Table 5.1: Measured times and deviations. Note that x is in the same direction as ω_{1xy}.

5.3 The Robot

5.3.1 Linear Model

Figure 4.11 displays the model of the robot modeled in Dymola. By using the *linearize* command in Dymola the state space model of the robot can be obtained. Equation 5.1 and Equation 5.2 show the state space model of the robot.

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 11.76 & 0 & 0 & 0 & -1.12 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 11.76 & 0 & 0 & -1.12 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -50 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 50 & 0 \\ 0 & 50 \end{bmatrix} \mathbf{u} \tag{5.1}$$

$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} \tag{5.2}$$

Dymola automatically selects the states when a model is linearized. The states selected are displayed in Table 5.2. The poles and zeroes of the open loop system are shown in Figure 5.1. There are four poles at the origin, one double pole in -50 and two double poles in ± 3.429. Because of the double pole in the right half plane the robot is not stable and will not stay in an upright position. To stabilize the system an optimal feedback matrix is calculated with the help of LQR. The **Q** and **R** matrices are weighted to get the desired behavior of the robot. The most important task for the controller is to keep the robot in an upright position, therefore deviations in rotation around the x- and y-axes are penalized the highest. To keep the robot at the origin is also important. Therefore the positions states are also penalized but not as highly as the angle deviation. The **Q** and **R** where

State	Physical variable
x_1	Position along the x-axis
x_2	Velocity along the x-axis
x_3	Position along the y-axis
x_4	Velocity along the x-axis
x_5	Rotation around the y-axis
x_6	Angular velocity around the y-axis
x_7	Rotation around the x-axis
x_8	Angular velocity around the x-axis
x_9	FirstOrder X acceleration
x_{10}	FirstOrder Y acceleration

Table 5.2: States of the linear model

finally chosen as

$$\mathbf{Q} = \begin{bmatrix} 50 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 50 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}.$$

The optimal feedback matrix, $\mathbf{L}$, with the weighted $\mathbf{Q}$ and $\mathbf{R}$ matrices is then

$$\mathbf{L} = \begin{bmatrix} -2.24 & -3.56 & 0 & 0 & -39.58 & -11.51 & 0 & 0 & 0.23 & 0 \\ 0 & 0 & -2.24 & -3.56 & 0 & 0 & 39.58 & 11.51 & 0 & 0.23 \end{bmatrix}$$

Figure 5.2 shows the poles and zeroes for the closed loop system with optimal control. Now all the poles are in the left half plane and thus the system is stable.

5.3.2 Model Simulations

Figure 5.3 shows the disturbance rejection of the Dymola model with the optimal feedback matrix calculated above. The initial inclination angle of the pendulum is 0.1745 rad (10 degrees) around the y-axis. To compensate for this the pendulum has to move along the x-axis. Figure 5.4 shows the disturbance rejection along the y-axis. The inclination angle is now 0.0873 rad (5 degrees) around the x-axis.

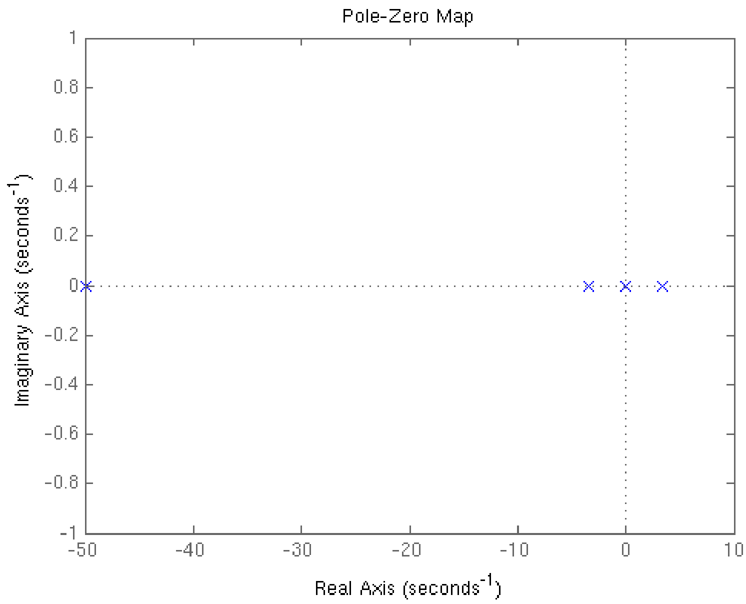

Figure 5.1: Pole-zero map for the open loop system.

5.3.3 Complementary Filter

In order to chose α an experiment was done. The robot was fixed on a rolling cart at an angle of approximately 3.5 degrees around the x-axis. The cart was then moved back and forth along the y-axis while measuring the acceleration and the angular velocity, see Figure 5.5 The angle was calculated from the measurements using Matlab for different α, see Figure 5.6. Since the gyro is not affected by acceleration it gives far better readings then the accelerometer. One might be tempted to set $\alpha = 1$. There are two good reasons for not doing so, the first reason is that the gyro drifts over time and secondly pure integration of the gyro to calculate the angle will require perfect initialization. Assuming that the accelerometer is fairly well calibrated an α less than one will have the effect that the initialization of the integration is less important and it will also correct the drift. With this in mind α was chosen to 0.995.

5.3.4 Robot Performance

Figure 5.7 shows the inclination angles around the x- and y-axes of the robot when balancing. Figure 5.8 shows the position of the robot along the x- and y-axes when balancing and trying to stand at the origin. A video of the robot is available at YouTube, `www.youtube.com/watch?v=eqhnZmMAU6M`

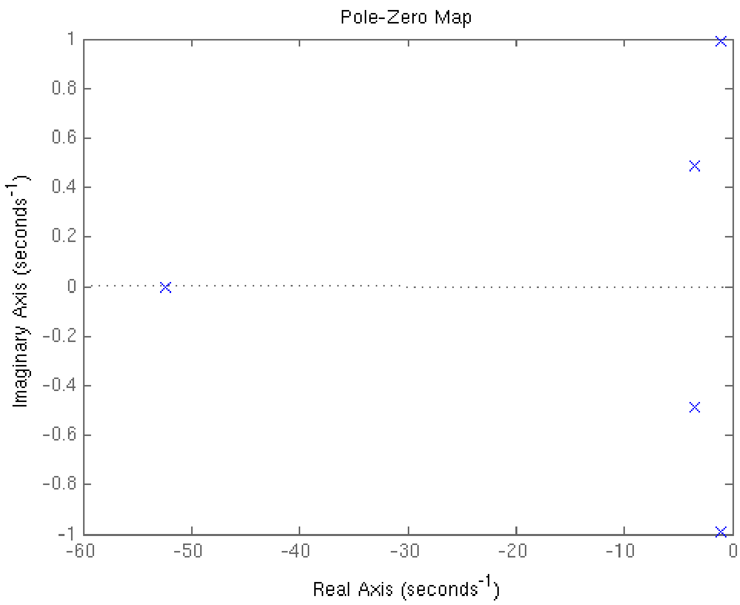

Figure 5.2: Pole-zero map for the closed loop system with LQR.

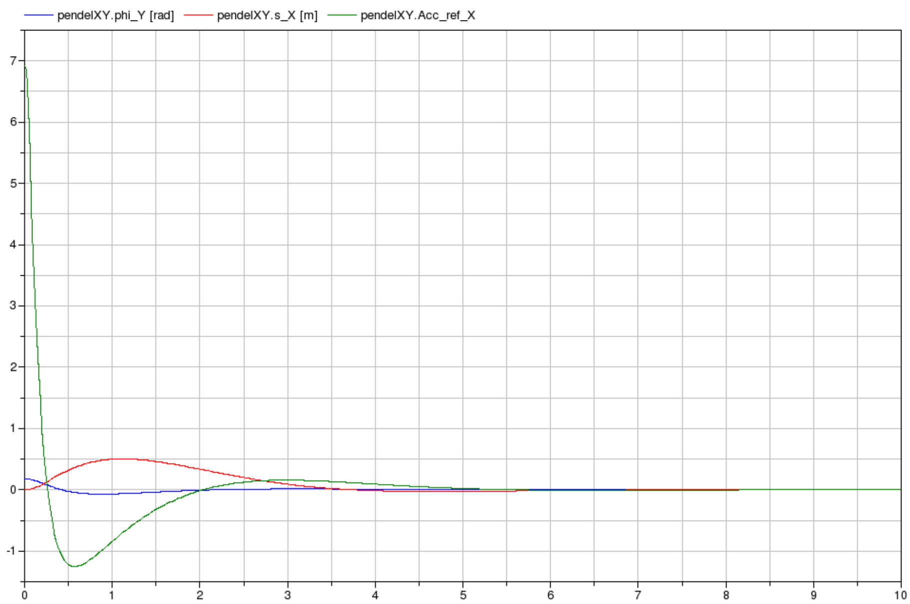

Figure 5.3: Initial disturbance rejection with 0.1745 rad (10 degrees) degree angle around the y-axis, compensation along the x-axis.

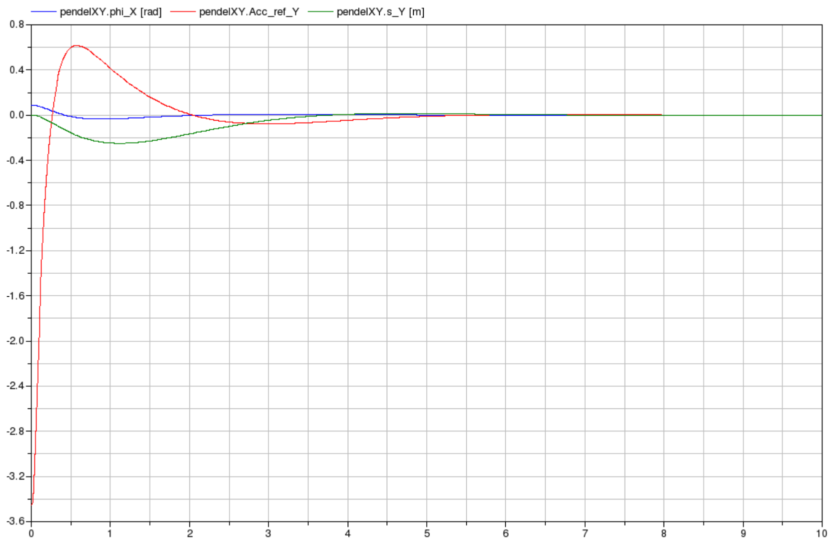

Figure 5.4: Initial disturbance rejection with 0.0873 rad (5 degrees) angle around the x-axis, compensation along the y-axis.

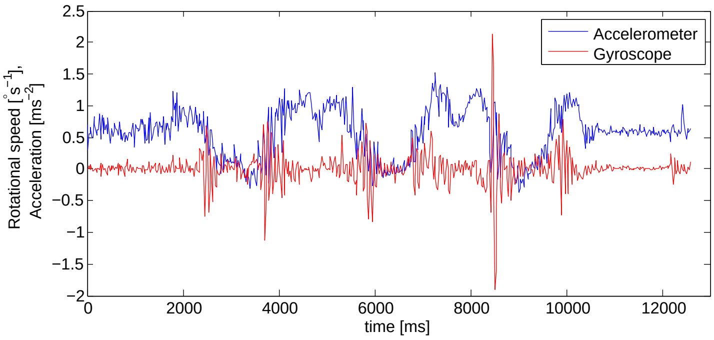

Figure 5.5: Measurements from accelerometer and gyroscope during the experiment.

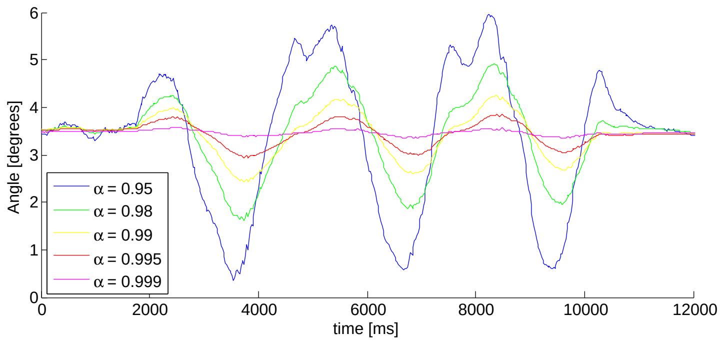

Figure 5.6: Estimated angle from the complementary filter for different α.

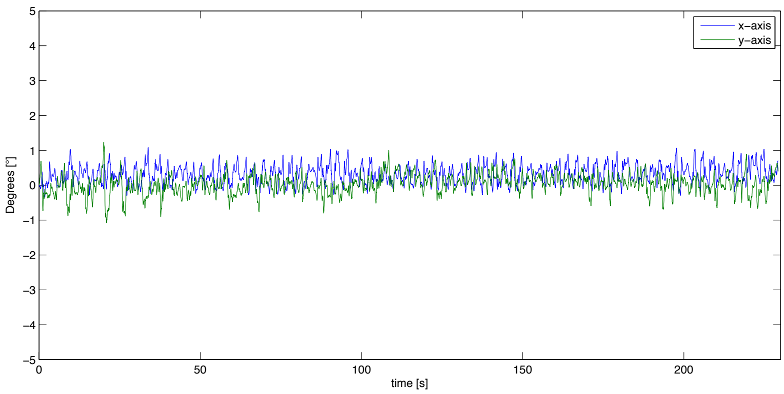

Figure 5.7: Inclination angles around the x- and y-axes when balancing.

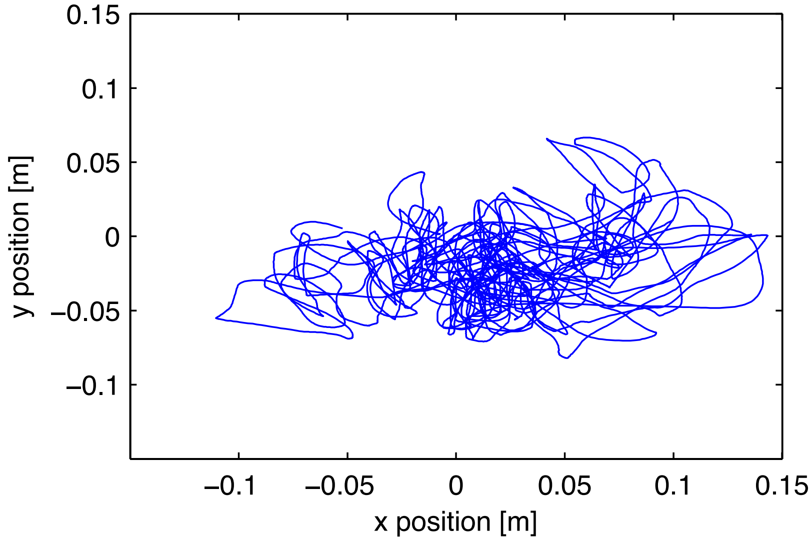

Figure 5.8: Position along the x- and y-axes of the robot when balancing.

Chapter 6

Conclusion and Future Work

A ball balancing robot was successfully developed and implemented on a microcontroller. The robot was stabilized with a LQR-controller. The dynamic model of the robot was modeled as a spherical inverted pendulum with two inputs and eight outputs. The mechanical construction consists of three actuated omni wheels. Other robots with similar design exist but they are equipped with a more advanced custom made omni wheels. The robot presented in this thesis was equipped with simpler and cheaper mass produced omni wheels. Despite the use of simpler omni wheels the performance of the robot was satisfying.

The status of the robot at the end of this master thesis has areas of improvements and more features could be added. Some suitable challenges may be:

- Improve the control design for a more stable system.
- Implement a better solution for obtaining the attitude of the robot.
- Add support for moving the robot around and add a GUI.
- Add a battery power supply to make the robot wireless.

Bibliography

[1] M. Kumaga and T. Ochiai. “Development of a robot balanced on a ball — Application of passive motion to transport —”. In: *Proc. IEEE Int. Conf. Robotics and Automation ICRA ’09*. 2009, pp. 4106–4111. DOI: `.2009.5152324`.

[2] A. Gfrerrer. “Geometry and kinematics of the Mecanum wheel”. In: *Computer Aided Geometric Design* 25.9 (2008). Classical Techniques for Applied Geometry, pp. 784 –791. ISSN: 0167-8396. DOI: `10.1016/j.cagd.2008.07.008`. URL: `http://www.sciencedirect.com/science/article/pii/S0167839608000770`.

[3] *KUKA youBot Store*. URL: `www.youbot-store.com` (visited on 2012-02-21).

[4] H. Asama et al. “Wheel for Omnidirectional Mobile Robot”. Pat. JP-patent no.3421290. 2003.

[5] *Lego Mindstorms*. URL: `http://mindstorms.lego.com/en-us/default.aspx` (visited on 2012-03-13).

[6] *Atmel*. URL: `www.atmel.com` (visited on 2012-02-21).

[7] *Arduino*. URL: `www.arduino.cc` (visited on 2012-02-20).

[8] *DIY Drones*. URL: `www.diydrones.com` (visited on 2012-02-21).

[9] *InvenSense, Inc.* URL: `www.invensense.com` (visited on 2012-02-21).

[10] *Honeywell*. URL: `www.honeywell.com` (visited on 2012-02-21).

[11] *Faulhaber*. URL: `www.faulhaber.com` (visited on 2012-02-21).

[12] Dr. Fritz Faulhaber GmbH & Co. KG. *Instruction Manual*. 4th. Faulhaber. 2009-09.

[13] T. Glad and L. Ljung. *Reglerteori - Flervariabla och olinjära metoder*. 2nd ed. 2003. ISBN: 978-91-44-03003-6.

[14] W. T. Higgins. “A Comparison of Complementary and Kalman Filtering”. In: 3 (1975), pp. 321–325. DOI: `10.1109/TAES.1975.308081`.

[15] S. P. Tseng et al. “Motion and attitude estimation using inertial measurements with complementary filter”. In: *Proc. 8th Asian Control Conf. (ASCC)*. 2011, pp. 863–868.

[16] R. Mahony, T. Hamel, and J.-M. Pflimlin. "Nonlinear Complementary Filters on the Special Orthogonal Group". In: 53.5 (2008), pp. 1203–1218. DOI: 10.1109/TAC.2008.923738.

[17] *Rotacaster Wheel Limited*. URL: www.rotacaster.com.au (visited on 2012-02-21).

[18] *Dymola*. URL: www.3ds.com/products/catia/portfolio/dymola (visited on 2012-02-21).

[19] *Modelica*. URL: www.modelica.org (visited on 2012-02-21).

[20] S. Colton. *The Balance Filter: A Simple Solution for Integrating Accelerometer and Gyroscope Measurements for a Balancing Platform.* 2007. URL: http://web.mit.edu/scolton/www/filter.pdf (visited on 2012-03-12).

Appendix A

Source-code

```
#include <EasyTransfer.h>
// Printing options------------------------------------------------
#define  PRINT_STATUS  1  // Print status messages
#define  PRINT_XY      1  // posx posy vx vy
#define  PRINT_OMEGA   1  // omega_m1,m2,m3 in motor rpm
#define  PRINT_ANGLES  0  // pitch, roll, yaw
#define  PRINT_OMEGAXYZ 0 // Print omegaXYZ, may be filtered
#define  PRINT_ACC_IMU 0  // Print acceleration from IMU m/s^2
#define  PRINT_ACC_CONTROL 1 // Control acceleration
#define  PRINT_V       0  // Control speed
#define  PRINT_MOTOR   1  // Motor controlsignal
#define  PRINT_MSTR    0  // Actual strings read from motors
#define  PRINT_TIME    1  // Print benchtime period time and timestamp
#define  PRINT_TEST    0  // For testing and debugging
#define  DEC_PRINT     4  // How many decimals to print

// Options  ---------------------------------------------------
#define  CONTROL_ON         1  // send control signals
#define  ACC_LIMIT_MIN      1  // acceleration, avoiding zero output
                               // for small signals
#define  CF_CONSTANT        0.995  // compl. filter constant
#define  IMU_FILTER_ACC     0  // Lowpass filter IMU acc signals
#define  IMU_ FILTER_OMEGA  1  // Lowpass filter IMU omega signals
#define  ACC_X_TEST         0  // Fixed acc x
#define  ACC_Y_TEST         0  // Fixed acc y

// Option parameters -------------------------------------------
#define  ACC_MIN       0.12  // Min acceleration
#define  ACC_DEADZONE  0.03  // Deadzone overriding ACC_MIN
#define  ACC_X_FIXED 0
```

```
#define ACC_Y_FIXED 0

// Filter parameters
#define ALPHA_IMU_OMEGA 0.8 // Lowpass filter *new value

// Feedback matrix
#define L1 -2.2 // Position
#define L2 -3.6 // Velocity
#define L3 39.6 // Angle
#define L4 11.5 // Angluar speed

// Kinematics matrix
#define M11 -351.1289080813352
#define M12 0
#define M13 0
#define M21 175.5644540406676
#define M22 -304.0865544015273
#define M23 0
#define M31 175.5644540406676
#define M32 304.0865544015273
#define M33 0

// Matrix for inverted kinematiks
#define W11 -0.054391970388845
#define W12 0.027195985194422
#define W13 0.027195985194422
#define W21 0
#define W22 -0.047104828118631
#define W23 0.047104828118631

// Negative due to motors dont follow right hand rule
#define M1_SPEED -0.087266462599716 // Encres 3*1200
#define M23_SPEED -0.051132692929521 // Encres 3*2048

// Get omega_? from query GV from faulhaber rad/s
// (encres taken care of within faulhaber unit)
// Motor_rpm /(ratio*60)*2pi = wheel_rad/s
#define GN_TO_RAD 0.03490658504

#define SENSOR_TIMEOUT 40 // Timeout before lost connection, ms
#define MOTOR_TIMEOUT 5 // Timeout for motor response
#define STRING_LENGTH 16 // Length of readins buffert strings

//---------- Declearations ----------------------------------
```

```
long timeStamp; // Time at beginning of every period
long timeStampOld;
long periodTime;  // Previous actual period
long timeSyncOffset=0; // Offset when syncing sensor and arduino clock
long benchTime; // For measure the calcing time
long motorTimeout;  // Used in readSerial123
long testTimer=0;  // For debugging and testing

// Strings used for communication
char m1str[STRING_LENGTH];
char m2str[STRING_LENGTH];
char m3str[STRING_LENGTH];
char m1str_ctrl[8];
char m2str_ctrl[8];
char m3str_ctrl[8];

float acc_x_imu=0;
float acc_y_imu=0;

float phi_x=0;
float phi_y=0;
float phi_z=0;

float omega_x=0;
float omega_y=0;
float omega_z=0;

// Motor positions
long m1New=0;
long m1Old=0;
long m2New=0;
long m2Old=0;
long m3New=0;
long m3Old=0;
long m1gn=0;
long m2gn=0;
long m3gn=0;

// Motor speeds rad/s
float omega_m1=0;
float omega_m2=0;
float omega_m3=0;

// Ball speed m/s and position m
```

```
float v_bx=0;
float v_by=0;
float pos_bx=0;
float pos_by=0;

float v_bxGN=0;
float v_byGN=0;

float acc_x_control=0;
float acc_y_control=0;
float v_x_control=0;
float v_y_control=0;
float m1_control=0;
float m2_control=0;
float m3_control=0;

//Others
boolean contact=false;  // True if connected to sensor
boolean booleanTest=false;  // For testing
int intTest=0;
int i=0;


// Easy Transfer stuff ---------------
// Protocol for sensor communication
EasyTransfer ETR;
struct RECEIVE_DATA_STRUCTURE{
  //THIS MUST BE EXACTLY THE SAME ON THE OTHER ARDUINO
  long  tS;
  float acc_x;
  float acc_y;
  float omega_x;
  float omega_y;
  float mag_z;
};

//give a name to the group of data
RECEIVE_DATA_STRUCTURE mydataR;



//--------- SETUP ---------------------------------------------
void setup() {

```

```
  Serial.begin(115200);
  Serial1.begin(115200);
  Serial2.begin(115200);
  Serial3.begin(115200);

  // Rest encoder positions and Motor Power OFF.
  Serial1.print("DI\rHO\r");
  Serial2.print("DI\rHO\r");
  Serial3.print("DI\rHO\r");

  // When uploading the old program will run a short
  // while before uploading starts.
  delay(1000);

  // Easy Transfer for sensor communication
  ETR.begin(details(mydataR), &Serial);

  // Rest encoder positions
  Serial1.print("DI\rHO\r");
  Serial2.print("DI\rHO\r");
  Serial3.print("DI\rHO\r");

  flushSerial123();

  if(PRINT_STATUS){
    Serial.println();
    Serial.println("Setup done...");
    Serial.println("Waiting for sensor...");
  }
}


//--------- MAIN LOOP ------------------------------------------

void loop() {
  // CONNECTED
  if(ETR.receiveData()){
    // If new contact Motors power ON and reset POS.
    if(!contact){
      // Motor Power ON and Reset motor position
      Serial1.print("EN\rHO\r");
      Serial2.print("EN\rHO\r");
      Serial3.print("EN\rHO\r");
    }
```

```

    // Sync clocks
    timeSyncOffset=millis()-mydataR.tS;
    timeStamp=mydataR.tS+timeSyncOffset;

    // Request motor positions
    Serial1.print("POS\r");
    Serial2.print("POS\r");
    Serial3.print("POS\r");

    sampleIMU();
    sampleOmegaPOS();
    calcBallPosSpeed();
    calcControl();
#if CONTROL_ON
    sendControl();
#endif
    endStuff();
    print2comp();
  }

  // Detect lost connection
  else {
    if(millis()>(SENSOR_TIMEOUT+timeStamp) && contact){
      connectionLost();
    }
    flushSerial123();
  }

}



//----------METHODS---------------------------------------------


// Calc control signals
void calcControl(){

  // Calc acc
  acc_y_control=L1*pos_by+L3*phi_x+L4*omega_x+L2*v_by;
  acc_x_control=L1*pos_bx-L3*phi_y-L4*omega_y+L2*v_bx;

  // Check minimim acc.
```

```
  #if ACC_LIMIT_MIN
  // X
  if(abs(acc_x_control)<=ACC_MIN){
    if(acc_x_control>=ACC_DEADZONE){
      acc_x_control=ACC_MIN;
    }
    else if(acc_x_control<=-ACC_DEADZONE){
      acc_x_control=-ACC_MIN;
    }
  }
  // Y
  if(abs(acc_y_control)<=ACC_MIN){
    if(acc_y_control>=ACC_DEADZONE){
      acc_y_control=ACC_MIN;
    }
    else if(acc_y_control<=-ACC_DEADZONE){
      acc_y_control=-ACC_MIN;
    }
  }
#endif

  // Integrate
  v_y_control=-acc_y_control*0.02+v_by;
  v_x_control=-acc_x_control*0.02+v_bx;
  // Calc wheel speeds
  m1_control=M11*v_x_control;
  m2_control=M21*v_x_control+M22*v_y_control;
  m3_control=M31*v_x_control+M32*v_y_control;
}

//----------------------------------------------------------------

// Send control signals
void sendControl(){
  sprintf(m1str_ctrl,"V%d\r\0", int(-m1_control));
  sprintf(m2str_ctrl,"V%d\r\0", int(-m2_control));
  sprintf(m3str_ctrl,"V%d\r\0", int(-m3_control));
  Serial1.print(m1str_ctrl);
  Serial2.print(m2str_ctrl);
  Serial3.print(m3str_ctrl);
}

//----------------------------------------------------------------

```

```
// Filter the sensor input
void sampleIMU(){
#if IMU_FILTER_ACC
  acc_x_imu=ALPHA_IMU_ACC*mydataR.acc_x
  +(1-ALPHA_IMU_ACC)*acc_x_imu;

  acc_y_imu=ALPHA_IMU_ACC*mydataR.acc_y
  +(1-ALPHA_IMU_ACC)*acc_y_imu;

#else
  acc_x_imu=mydataR.acc_x;
  acc_y_imu=mydataR.acc_y;
#endif

#if IMU_FILTER_OMEGA
  omega_x=ALPHA_IMU_OMEGA*mydataR.omega_x
  +(1-ALPHA_IMU_OMEGA)*omega_x;

  omega_y=ALPHA_IMU_OMEGA*mydataR.omega_y
  +(1-ALPHA_IMU_OMEGA)*omega_y;

#else
  omega_x=mydataR.omega_x;
  omega_y=mydataR.omega_y;
#endif

  // Limit to 1 g, otherwise asin(acc/g) fails
  if(acc_x_imu>9.81){
    acc_x_imu=9.80;
  }
  else if(acc_x_imu<-9.81){
    acc_x_imu=-9.80;
  }
  if(acc_y_imu>9.81){
    acc_y_imu=9.80;
  }
  else if(acc_y_imu<-9.81){
    acc_y_imu=-9.80;
  }

  phi_x=CF_CONSTANT*(phi_x+mydataR.omega_x*0.02)
  +(1-CF_CONSTANT)*asin(acc_y_imu/9.81);

  phi_y=CF_CONSTANT*(phi_y+mydataR.omega_y*0.02)
```

```
  +(1-CF_CONSTANT)*asin(-acc_x_imu/9.81);
}

//-------------------------------------------------------------

// Stuff that can be done after the control signals has been sent
void endStuff(){
  benchTime=millis()-timeStamp;
  // Update
  m1Old=m1New;
  m2Old=m2New;
  m3Old=m3New;
  periodTime=timeStamp-timeStampOld;
  timeStampOld=timeStamp;
  // Clear motor buffers
  flushSerial123();
  // If new contact
  if(!contact){
    // Print status update
    if(PRINT_STATUS){
      Serial.println("Sensor OK\nMotor Power ON\nRunning...\n");
    }
  }
  contact=true;

}

//-------------------------------------------------------------
// Reads positions from motors
void sampleOmegaPOS(){
  //benchTime=micros();
  // Ask for positions
  readSerial123();
  // Parse long from strings
  m1New=atol(m1str);
  m2New=atol(m2str);
  m3New=atol(m3str);
  // Calc rotational wheel speeds. rad/s
  omega_m1=M1_SPEED*(m1New-m1Old);
  omega_m2=M23_SPEED*(m2New-m2Old);
  omega_m3=M23_SPEED*(m3New-m3Old);
}

//-------------------------------------------------------------
```

```
// Calculate boll position and speed
void calcBallPosSpeed(){

  // Calc ball speed
  v_bx=W11*omega_m1+W12*omega_m2+W13* omega_m3;
  v_by=W22*omega_m2+W23*omega_m3; // W21=0;

  // Calc ball position
  pos_bx+=v_bx*0.02; // Hard coded sampling time
  pos_by+=v_by*0.02;
}

//----------------------------------------------------------------

// Reads serial 1,2,3 will stopa at \r and replace it with \0
void readSerial123(){
  motorTimeout=millis()+MOTOR_TIMEOUT;
  // Read answers
  // Read Serial1
  i=0;
  while(1){
    m1str[i]=Serial1.read();
    if(m1str[i]=='\r'){
      m1str[i]='\0';
      break;
    }
    if(m1str[i]!=-1){
      i++;
    }
    else if(millis()>motorTimeout){
      if(PRINT_STATUS){
        Serial.println("ERROR: M1");
      }
      m1str[i]='\0'; // Keep old value FIX!!
      break;
    }
  }

  // Read Serial2
  i=0;
  while(1){
    m2str[i]=Serial2.read();
    if(m2str[i]=='\r'){
```

```
      m2str[i]='\0';
      break;
    }
    if(m2str[i]!=-1){
      i++;
    }
    else if(millis()>motorTimeout){
      if(PRINT_STATUS){
        Serial.println("ERROR: M2");
      }
      m2str[i]='\0'; // Keep old value FIX!!
      break;
    }
  }

  // Read Serial3
  i=0;
  while(1){
    m3str[i]=Serial3.read();
    if(m3str[i]=='\r'){
      m3str[i]='\0';
      break;
    }
    if(m3str[i]!=-1){
      i++;
    }
    else if(millis()>motorTimeout){
      if(PRINT_STATUS){
        Serial.println("ERROR: M3");
      }
      m3str[i]='\0'; // Keep old value FIX!!
      break;
    }
  }
}

//-------------------------------------------------------------

// Called when connection with Sensor is lost
void connectionLost(){
  // Motor Power OFF
  Serial1.print("DI\r");
  Serial2.print("DI\r");
  Serial3.print("DI\r");
```

```
  contact=false;
  if(PRINT_STATUS){
    Serial.println("ERROR  Connection lost!");
    Serial.println("Motor Power OFF");
    Serial.println("Please check sensor");
    Serial.println("Waiting for sensor...");
  }
}

//--------------------------------------------------------------

// Empty Serial 1,2,3 reading buffers.
void flushSerial123(){

  // Since new version of arduino 1.0 flush()
  // is changed and do not work in the same way
  while(Serial1.read()!=-1){
    // Do nothing
  }
  while(Serial2.read()!=-1){
    // Do nothing
  }
  while(Serial3.read()!=-1){
    // Do nothing
  }
}

//--------------------------------------------------------------

// Strings sent to computer at the end.
// If too long sampling may be delayed.
void print2comp(){
#if PRINT_XY
  // Print speed and pos
  Serial.print(pos_bx, DEC_PRINT);
  Serial.print("\t");
  Serial.print(pos_by, DEC_PRINT);
  Serial.print("\t");
  Serial.print(v_bx, DEC_PRINT);
  Serial.print("\t");
  Serial.print(v_by, DEC_PRINT);
  Serial.print("\t");
#endif

```

```
#if PRINT_OMEGA
  Serial.print( omega_m1, DEC_PRINT);
  Serial.print("\t");
  Serial.print( omega_m2, DEC_PRINT);
  Serial.print("\t");
  Serial.print( omega_m3, DEC_PRINT);
  Serial.print("\t");
#endif

#if PRINT_ANGLES
  Serial.print(phi_x, DEC_PRINT);
  Serial.print("\t");
  Serial.print(phi_y, DEC_PRINT);
  Serial.print("\t");
  Serial.print(phi_z, DEC_PRINT);
  Serial.print("\t");
#endif

#if PRINT_OMEGAXYZ
  Serial.print(omega_x, DEC_PRINT);
  Serial.print("\t");
  Serial.print(omega_y, DEC_PRINT);
  Serial.print("\t");
  Serial.print(omega_z, DEC_PRINT);
  Serial.print("\t");
#endif

#if PRINT_ACC_IMU
  Serial.print(acc_x_imu, DEC_PRINT);
  Serial.print("\t");
  Serial.print(acc_y_imu, DEC_PRINT);
  Serial.print("\t");
#endif

#if PRINT_ACC_CONTROL
  Serial.print(acc_x_control, DEC_PRINT);
  Serial.print("\t");
  Serial.print(acc_y_control, DEC_PRINT);
  Serial.print("\t");
#endif

#if PRINT_V
  Serial.print(vx_control, DEC_PRINT);
  Serial.print("\t");
```

```
  Serial.print(vy_control, DEC_PRINT);
  Serial.print("\t");
#endif

#if PRINT_MOTOR
  Serial.print(m1_control, 0);
  Serial.print("\t");
  Serial.print(m2_control, 0);
  Serial.print("\t");
  Serial.print(m3_control, 0);
  Serial.print("\t");
#endif

#if PRINT_MSTR
  Serial.print(m1str);
  Serial.print("\t");
  Serial.print(m2str);
  Serial.print("\t");
  Serial.print(m3str);
  Serial.print("\t");
#endif

#if PRINT_TIME
  Serial.print(benchTime);
  Serial.print("\t");
  Serial.print(periodTime);
  Serial.print("\t");
  Serial.print(timeStamp);
#endif

#if PRINT_TEST
  Serial.print("\t");
  Serial.print(intTest);
#endif

  Serial.println();

}
```