Ball ballancing robot

FREDRIK SANDAHL

WILLIAM MILES
Ball balancing robot

Bachelor thesis

FREDRIK SANDAHL
WILLIAM MILES

Bachelor thesis at KTH
Supervisor: Fariba Rahimi
Examiner: Nihad Subasic

MMK 2017:31 MDAB 649
Abstract

This report will research how micro stepping effects the vibrations from the motors. The report will also cover the steps that are needed for design a ball balancing robot. This type of robot can be seen as an inverted pendulum. Without any controller the system is unstable and the robot will fall. With help of an IMU that provides the necessary data which is needed to implement a controller which makes it possible for the robot to balance on the ball. Because of weak motors the robot wasn’t able to balance. Other reasons and areas of improvement is discussed in the report.
Sammanfattning

Acknowledgements

We would like to thank our supervisor, Fariba Rahimi for the feedback and support. Tomas Österås, Staffan Qvarnström for the electrical and mechanical manufacturing and the lab assistants.
Contents

1 Introduction
   1.1 Background
   1.2 Scope
   1.3 Method

2 Theory
   2.1 Omni-wheel Dynamic
   2.2 Motor control
      2.2.1 Stepper motor
      2.2.2 H-bridge
   2.3 Microcontroller
   2.4 Control theory
   2.5 Internal measurement Unit
   2.6 Filtering
   2.7 Theoretical analysis

3 Demonstration
   3.1 Hardware
      3.1.1 Microcontroller
      3.1.2 The Robot
   3.2 Data reading, \( v, \psi, \dot{\psi} \) and \( x \)
   3.3 Movement - Ball and robot
   3.4 Conducting tests

4 Result

5 Discussion and conclusions
   5.1 Discussion
   5.2 Conclusions

6 Recommendation and future work
# Nomenclature

## Abbreviation Description

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD</td>
<td>Computer-aided design</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DC</td>
<td>Direct current</td>
</tr>
<tr>
<td>IMU</td>
<td>Internal measurement unit</td>
</tr>
<tr>
<td>MATLAB</td>
<td>Matrix Laboratory</td>
</tr>
<tr>
<td>MEMS</td>
<td>Micro-electro-mechanical system</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse width modulation</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
</tbody>
</table>

## Symbols Description

- **Angular velocity for robot (degrees/s)**: \( \dot{\theta}_0 \)
- **Torque \((Nm)\)**: \( \tau_0 \)
- **Angle between robot and upright position along x-axis (degrees)**: \( \theta_x \)
- **Angle between robot and upright position along y-axis (degrees)**: \( \theta_y \)
- **Acceleration along x-axis \((m/s^2)\)**: \( a_x \)
- **Acceleration along y-axis \((m/s^2)\)**: \( a_y \)
- **Moment of inertia \((kgm^2)\)**: \( I \)
- **Moment of inertia of ball \((kgm^2)\)**: \( I_b \)
- **Moment of inertia of robot \((kgm^2)\)**: \( I_p \)
- **Gain constant**: \( K_A \)
- **Gain constant**: \( K_V \)
$K_{AV}$ Gain constant

$K_T$ Gain constant

$L$ Angular momentum ($kgm^2/s$)

$l$ Height of center of mass ($m$)

$m$ Mass of the falling person ($kg$)

$m_b$ Mass of the ball ($kg$)

$m_p$ Mass of the robot ($kg$)

$r_b$ Radius of the ball ($cm$)

$r_w$ Radius of the omni-wheels ($cm$)

$v_x$ Velocity along x-axis ($m/s$)

$v_y$ Velocity along y-axis ($m/s$)

$w$ Angular velocity for the ball ($rad/s$)

$x$ Distance traveled along x-axis ($m$)

$y$ Distance traveled along y-axis ($m$)
Chapter 1

Introduction

1.1 Background

To make robots be a part of humans everyday life they need too be flexible and move around objects in an efficient way, take little space to manage the crowded areas and be able to be knocked by individuals without falling or losing control. Two wheeled balancing robot was demonstrated in Japan 1994 [18]. The design eliminated the third wheel to make it stable and made the robot take less space but it could not maneuver side ways. The project led to new ideas, to use a ball as the wheel. There are benefits with this design, they leave very little footprint and are very agile. Many different robots have been made using different drivers like DC motors, servo motors and stepper motors. Commonly they are modeled as inverted pendulums and balanced with control theory.

1.2 Scope

The process of making a self balancing robot on a ball is very complex. The aim for this project is to use stepper motors instead of DC motors. Because stepper motors vibrate much more than DC motors it is important to minimize the noise they generate. The report will investigate the following questions. One will be able to answer by investigating and the other by research.

- How does raise/lowering the center of mass of the ball balancing robot affect the dynamics of the robot?
- How to maximize the torque from the stepper motors in relationship to the disturbance that affects the IMU readings?

1.3 Method

The goal of this project is to make a ball balancing robot. To achieve this objective the following procedure is done.
• Design the test rig as a CAD model and create the test rig using a 3D-printer.
• Derive the equation for controlling the ball with the omni wheels.
• Extract values from the IMU and filter out the noise.
• Extract the data from the sensors and filter it too make it accurate.
• Find the optimal micro-stepping that will cause high torque without causing too much noise.

The robot will be powered by three stepper motors to ensure precision from the motors. Each motor is turning one omni wheel. The omni wheels are designed to be in an angel that will prevent the robot from sliding of the ball. A accelerometer and gyroscope will give the feedback signal to the controller to be analyzed. The calculations are done by an Arduino Uno. The robot are designed to fit a size five football.
Chapter 2

Theory

2.1 Omni-wheel Dynamic

Figure 2.1: Omni wheels degrees of freedom. [1]
Omni-wheels has the ability to move in the direction perpendicular to its own rotation. But there is only one direction that can be controlled. The Omni-wheels that have been used for this project have cylindrical bearings around the wheel, which can be seen in the figure 2.1. The bearings make the wheels have two degrees of freedom. By combining three omni wheels with an angel of $120^\circ$ form each other as seen in 2.2.
2.2 Motor control

2.2.1 Stepper motor

For this project a stepper motor has been chosen because of there high precision without using an encoder. Stepper motor is a brushless DC-motor that have the benefit of being able to split every rotation too many small steps of equal sizes. This makes the stepper motor very precise. It can be commanded to hold a specific position without needing any feedback controller. Stepper motor is controlled by sending electric pulses to the motor and every pulse makes the motor turn one step. By having two electromagnets and teeth that are not aligned with the magnets, giving current to the different electromagnets will make the axis step. The stepper motor that has been used in this project is a two phase stepper motor[9].

2.2.2 H-bridge

![H-bridge diagram](image)

Figure 2.3: Representation of an H-bridge [3]

H-bridge is used to control motors. This is done by turning different transistors on and off. So that the direction of the motor can be controlled by directing the current through transistors that works as switches. In the H-bridge the switches which are parallel to each other works together. When one par is open the other one is closed, see figure 3.4. In this way the direction of the current changes which will make the motor rotate in different direction. In this project dual h-bridges has been used, they work in the same way that a single H-bridge works but because a stepper motor has two coils a dual H-bridge is needed[12].

2.3 Microcontroller

A microcontroller is a small computer with integrated circuit, it has memory, one CPU or more, small amount of RAM and out/in pins. The most common uses for microcontroller is embedded systems. The microcontroller is able to read data from sensors, compute and give commands to the motors. [8].
2.4 Control theory

The control equations are represented by:

\[ a_x = K_A \theta_x + K_{AV} \dot{\theta}_x + K_T (x - x_0) + K_V v_x \]  \tag{2.1}

\[ a_y = K_A \theta_y + K_{AV} \dot{\theta}_y + K_T (y - y_0) + K_V v_y \]  \tag{2.2}

\( \theta \) is the angle and \( \dot{\theta} \) is the angular velocity, \( x/y \) is the position in \( x/y \) directions and \( v \) is the velocity of the ball. \( K_A, K_{AV}, K_T \) and \( K_V \) are different gains. The gains can be determined by experimenting. This is most commonly done by setting all the gains to zero and start testing to find a \( K_A \) where the system starts to become stable. Then fine tuning the robot by changing the rest of the gains. \[4\]

2.5 Internal measurement Unit

To be able to measure the position of the robot an Internal measurement Unit (IMU) is used to measure acceleration and angular rate. It is built up with one or more linear accelerometers and one or more gyroscopes or angular accelerometers. Accelerometers are a type of transducers that gives away electrical signals that are proportional to the exposed accelerations. The accelerometer is fastened to the frame of the structure that it is going to measure. A small mass inside the accelerometer is free to move and because of gravity inertia the mass will move and it will be converted into electrical signals. A new method has been developed where the measurement unit is made of silicon structure. It has small movable silicon rods that are very robust. They are often used in cellular devices. The accelerometers are very sensitive and detect a lot of noise from disturbance. \[5\]

Complementary to the accelerometer the IMU uses gyroscopes too be able too get a more accurate measurement. They measure angular rate or orientation about a given directional vector. It’s principal is having a spinning disc which is supported in light rings. Because of the frictionless joints in the rings the spinning disc will always point at the same direction. But too fit a gyroscope in a small circuit board they use vibrating gyroscopes called MEMS. It has a mass vibrating using piezoelectric current. When the gyro is spun the mass wants to keep vibrating in the same direction which causes them too put pressure on piezoelectric components which send an electrical signal to the computer \[11\].

2.6 Filtering

The IMU provides real time data but because of the sensitivity of the sensors they detect lots of noise. Vibrations from the motors affect the reading from the accelerometer. Too get an accurate reading for the angle that the robot is tilting the signals from accelerometer and gyro are fused together. By applying a Kalman filter for the signal fusing, a steady and accurate reading can be obtained. Kalman filter weights the sources
of information depending on the knowledge about the signal and its characteristics. Because every sensor has flaws, weak and strong features, the signal fusing complements the different weaknesses of the different sensors, gyro and accelerometer [7].

2.7 Theoretical analysis

An important aspect when designing a ball balancing robot is to raise the center of mass, because this will lead to a longer fall time for the robot. Which will mean that the system will have more time to become stable again. A ball balancing robot is often seen as an inverted pendulum which means that the system is unstable. The human body can be seen as an inverted pendulum because input from our muscles is needed too keep the human body from falling over[15].

So what are the proofs that raising the center of mass also will raise the fall time? This has been proven [6].

![Figure 2.4: Simple model of a human falling. g is the gravity, l is the length from the ground to the center of mass of the body and theta is the angle from the upright position to degree that the body have fallen[15]](image)

If one of the foots is attached to the ground the fall can been seen as having the falling pattern of an arc. with these assumptions the fall can been seen as an inverted pendulum with l as the length from the floor to the center of mass for the body. It will also be assumed that the body only can move in \( \theta \) direction. The angular momentum principle can then be used which are described as:

\[
\tau_0 = \frac{dL_0}{dt}
\]  

(2.3)
Angular moment is a vector but in this case the vector dose not change direction if the body is rigid. The gravity can be summarized too only be pulling on the center of mass for the human body so the torque can be written as:

\[
\tau_0 = mg\frac{l}{2}\sin(\theta)
\]  

The moment of inertia can be written as:

\[
I = ml^2
\]  

If equation 2.6 is inserted in equation 2.4. If then equation 2.4 is derived it will be equal to the torque so we can replace the torque with equation 2.5. If then the second derivative of \( \theta \) is moved to one side the angular acceleration will be:

\[
\frac{d^2\theta}{dt^2} = \frac{3g\sin(\theta)}{2l}
\]  

Where equations 2.7 shows that the height to the center of mass is linear proportional to the angular acceleration. This is the equation when the mass is evenly distributed over the hole body, but in the case with a inverted pendulum the mass is assumed too be at the top. So the equation for an inverted pendulum is:

\[
\frac{d^2\theta}{dt^2} = \frac{g\sin(\theta)}{l}
\]  

So the angular acceleration will be slower with a factor of \( 3/2 \) for the inverted pendulum which is another reason too have as much of the weight at the top of the robot. But the principles are still the same, the angular acceleration will decrease with the length to the center of mass.
Chapter 3

Demonstration

3.1 Hardware

3.1.1 Microcontroller

In this project an Arduino uno (ATmega328P) is used, it has 16 in/out-pins of which six provides a PWM signal and operates at 5v.

Figure 3.1: Schematics of the hardware and their communication. Made in Lucidchart.
3.1.2 The Robot

The stepper motor that is used for this project is KH39F52-851. It is a two phase hybrid stepper motor and the drive method is bipolar. It was chosen because they could rotate in both direction and the motors are very precise. The omni wheel that are used has a diameter of 50 mm.
Figure 3.3: CAD model of the robot, missing Arduino and H-bridge circuit. Made in Solid Edge.

Figure 3.3 shows the robot which is made in CAD. From the CAD program the robots moment of inertia, mass and center of mass is found. The parameters are shown in the table below.

<table>
<thead>
<tr>
<th>$r_b$</th>
<th>Radius of the ball</th>
<th>111 cm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_w$</td>
<td>Radius of the omni wheels</td>
<td>2.5 cm</td>
</tr>
<tr>
<td>$m_p$</td>
<td>Mass of robot</td>
<td>2.113 kg</td>
</tr>
<tr>
<td>$m_b$</td>
<td>Mass of ball</td>
<td>0.6 kg</td>
</tr>
<tr>
<td>$I_p$</td>
<td>Moment of inertia of robot</td>
<td>0.111 kgm$^2$</td>
</tr>
<tr>
<td>$I_b$</td>
<td>Moment of inertia of ball</td>
<td>0.00484 kgm$^2$</td>
</tr>
<tr>
<td>$l$</td>
<td>Height of center of mass</td>
<td>0.275 m</td>
</tr>
</tbody>
</table>

3.2 Data reading, $v$, $\psi$, $\dot{\psi}$ and $x$

The IMU inbuilt gyro can sense angular velocity and the inbuilt accelerometer senses angular acceleration in three axis from the raw values of the sensor. According to the data sheet the IMU is able to be set for different sensitivity’s in different ranges [14]. The accelerometer is set to $\pm 2g$ and the gyro to $\pm 250^\circ/s$ for highest resolution. An open source kalman filter is used to signal fuse the accelerometer readings and the gyro readings and $\psi$ is calculated. From the acceleration the velocity is derived by numerical integration in real time by,

$$v_{t+1} = v_t + \dot{v}_{t+1} dt$$  \hspace{1cm} (3.1)

where $dt$ is the time difference from the last loop and $\dot{v}$ is the acceleration from the accelerometer. Data from the IMU’s gyro is the angular velocity. Which is then translated
in to proper values and used for $\dot{\psi}$. $x$ is the movement of the ball and is calculated by

$$x = w r_b$$  \hspace{1cm} (3.2)$$

where $w$ is the angular velocity of the ball around the $x$ axis and $r_b$ is the radius of the ball. The four states that are needed are now known for the $x$ axis. The same equations are used too find the states in the $y$ axis.

### 3.3 Movement - Ball and robot

![Ball and robot diagram]

Figure 3.4: The test rig, $\theta$ will be set to 30 degrees.

The translation from speed of the ball to the velocity for the motors has already been studied[13] and the following equation has been used in similar problem.[13]

First step is too split up the speed $v$ into velocity $v_f$ that is parallel to the wheels axis and $v_s$ that are perpendicular to the wheel axis. They are related to $v$ by introducing the unit vector $s$. 
\[ \mathbf{v}_s = \mathbf{v}_s \mathbf{s} + \mathbf{v}_f \mathbf{s} = \pm \| \mathbf{v}_s \| \mathbf{s} + 0 = \mathbf{v}_s \quad (3.3) \]

The \( \mathbf{s} \) matrix is:

\[
\mathbf{s} = \begin{bmatrix}
0 & -1 & 0 \\
\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\
-\frac{\sqrt{3}}{2} & \frac{1}{2} & 0
\end{bmatrix}
\quad (3.4)
\]

The velocity \( v_s \) can also be described with the angular velocity \( \mathbf{w} \) and the position vector \( \mathbf{p} \).

\[
v_{s_i} = \sum_{j=1}^{3} (\mathbf{w}_j \times \mathbf{p}_i)(\mathbf{s}_i)^T \quad (3.5)
\]

The matrix \( \mathbf{p} \) is:

\[
\mathbf{p} = r_b \begin{bmatrix}
sin(\theta) & 0 & \cos(\theta) \\
-\frac{1}{2}sin(\theta) & \frac{\sqrt{3}}{2}sin(\theta) & \cos(\theta) \\
-\frac{1}{2}sin(\theta) & -\frac{\sqrt{3}}{2}sin(\theta) & \cos(\theta)
\end{bmatrix}
\quad (3.6)
\]

And the \( \mathbf{w} \) matrix is:

\[
\mathbf{w} = \begin{bmatrix}
w_x & 0 & 0 \\
0 & w_y & 0 \\
0 & 0 & w_z
\end{bmatrix}
\quad (3.7)
\]

This will give the expression for the velocity for motor one, two and three:

\[
\begin{bmatrix}
v_{s1} \\
v_{s2} \\
v_{s3}
\end{bmatrix} = \begin{bmatrix}
-v_y\cos(\theta) - r_b\sin(\theta)w_z \\
(\frac{\sqrt{3}}{2}v_x + v_y\frac{1}{2})\cos(\theta) - r_b\sin(\theta)w_z \\
(-\frac{\sqrt{3}}{2}v_x + v_y\frac{1}{2})\cos(\theta) - r_b\sin(\theta)w_z
\end{bmatrix}
\quad (3.8)
\]

Equation (3.8) yields the translation from velocity of the ball to the velocity for the motors for the ball balancing robot.

### 3.4 Conducting tests

The precision that a stepper motor offers makes it suitable for many tasks. Full step on many stepper motor and on the KH39FM2-851 that is used for this project has a step angle of 1.8°. Too increase the precision the H-bridge is able too micro step the motors. When half stepping, the step angle is divided by two. Therefor the stepping angle becomes 0.9° per step. The H-bridge is able too make the motors move in sixteenth of a step, the stepping angle is then 1.8/16° [17]. While micro stepping the torque drastically decreases and the top speed of the motors rotation. The current is proportional to the torque, but because of magnetic saturation there is no advantage of increasing the current more than 100% of the recommended current limit. There is also a greater chance of
damaging the motors[16]. In table 3.4 the different holding torques are shown depending on what micro stepping setting is used.

<table>
<thead>
<tr>
<th>Micro steps/full step</th>
<th>% Holding Torque/Micro step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.00%</td>
</tr>
<tr>
<td>2</td>
<td>70.71%</td>
</tr>
<tr>
<td>4</td>
<td>38.27%</td>
</tr>
<tr>
<td>8</td>
<td>19.51%</td>
</tr>
<tr>
<td>16</td>
<td>9.80%</td>
</tr>
</tbody>
</table>

In figure 3.5 the holding torque is shown until 256 of a full step. If the precision becomes too high the motor may not have enough torque too turn because of its own friciton.

To be able to obtain accurate readings from the IMU it is necessary to cancel as much noise as possible from the motors. The stepper motors vibrate a lot when they are driven in low speeds. Too minimize noise the stepper motor driver is capable of micro stepping. Several tests are made to evaluate the impact on the IMU when micro stepping. The motors are driven in two different speeds when micro stepping in full-, half-, eight- and sixteenth step. The data are plotted in MATLAB and an average amplitude of four test are summarized to evaluate the disturbance. The robot was placed on a flat floor and the bearings that are on the motors shaft make them turn without the robot moving.
Chapter 4

Result

The results for this project are obtained by several tests using the finalized robot. The disturbance from the motors are then compared with the loss of torque that the stepper motors suffer when micro stepping.

![Graph showing the relation between acceleration along X-axis and disturbance while microstepping.](image)

Figure 4.1: Relation between the acceleration (x-axis) disturbance and torque loss
Figure 4.2: Relation between the acceleration (y-axis) disturbance and torque loss
Figure 4.3: Relation between the angle (x-axis) disturbance and torque loss
The graphs show high disturbance while full stepping and makes it very inefficient to use even though 100% of the torque can be utilized. Inputs with too much disturbance will make the robot not be able to function properly. But if the torque loss is too great there is a risk that the motors will start to slip. If the motors are chosen wisely the optimal stepping setting is around a quarter step. The angular disturbance is less than 1 degree in every case except one and the acceleration does not peak over $1m/s^2$. There is a loss in torque with 38%.

Equation 2.7 shows that the acceleration will decrease depending on the length to the center of mass. It is impossible to determine the exact time that it takes the body to hit the ground without providing information about the deviation of angle $\theta$ from zero in the beginning. So the system would stay upright if the angle was exactly zero, but even the smallest offset will make the system fall. The fact that the angular acceleration will decrease with length $l$ to the center of mass will lead to that the speed will do the same and lower speed will lead to longer fall time.
Chapter 5

Discussion and conclusions

5.1 Discussion

The results show some deviation, but conclusions can be made. The deviation was caused by vibration in the ground from other projects or from other students walking nearby. Conclusions that can be made from the graphs is that half step is pointless to use because it outputs less torque and causes more noise than full step. Quarter step causes significant less noise than full step and can still output 40% of the torque. Quarter step is causing so little noise that it is possible to use for a ball balancing robot project. When using eight step and sixteenth step the torque will decrease but they cause negligible noise. If a motor is able to output a lot more torque than is needed, eight step is still recommended because it causes a small amount of noise and the torque is 19.5% compared to 9.8% for sixteenth step. The torque is almost half as big for sixteenth step but the noise is similar.

By raising the center of mass of an unstable system like an inverted pendulum, it will fall slower which is shown in chapter 2.7. It can be achieved by putting a weight (or the battery for the robot) on top of the robot. But with greater mass the motors must be able to output greater torque which means bigger motors. The best solution is to make a light rig with a small mass on top.

5.2 Conclusions

When creating a ball balancing robot with stepper motors, the motors have to be strong enough even when micro-stepping with at least quarter stepping. If the motors are too weak for quarter stepping so that half or full step is needed to be used the disturbance from the motors to the IMU will be too high. The main reason for the robot to fail balancing purpose was that the motors were too weak.

If the motor spins faster they will become weaker, so bigger omni-wheels should be used which lowers the speed that is needed to output.

If the motor spins faster they will become weaker, so bigger omni-wheels should be used which will lower the needed output speed from the motors.
Chapter 6

Recommendation and future work

There is still some future work that needs to be done before the robot will be balancing on a ball. Upgrading the stepper motors for higher torque is essential for the robot to function. Also installing bigger omni wheels is beneficial to achieve a faster speed while micro stepping.
Bibliography


Appendix A

Verifying Kalman filter with MPU-6050

For the robot to respond in the correct way it is crucial that the sensor fusion with the kalman filter has small errors. Too verify that the kalman filter performs good enough test where made. The IMU was fasten to a servo, which rotated the IMU for given angles and speed. Below the graphs from the test are shown.

Figure A.1: Angle from the IMU threw the kalman filter oscillating between +-45 degrees
Figure A.2: Angle from the IMU threw the kalman filter oscillating between $\pm 10$ degrees

Figure A.3: Angle from the IMU threw the kalman filter oscillating between $\pm 45$ degrees
By studying at the graphs the angle deviates by a little. For the grater angles, ±45 degrees there is also a delay for the readings with 150ms and a slight overshoot. While turning the IMU ±10 degrees the delay is 90ms and the angle overshoot is about 2 degrees. And ±5 degrees gives a delay of 70ms and a angular error as the most of 0.8 degrees. The results show that the IMU performs well and the small delays that accrues are minor. For small angles the IMU is more precise than moving over a wider range of angles. Because the robot will only experience fairly small changes in angle there will not be a problem.
Appendix B

Results and graphs from disturbance testing

Below are graphs for how acceleration or the angel for x or y for a motor speed of $0.5 \text{rad/s}$ or $1 \text{rad/s}$ from full step till stepping sixteenth step differs when robot is standing still but the motors are on. The reference value is the disturbance when the motor is turned off. All diagrams in Appendix B are made in Microsoft Excell.

![Diagram of Acceleration x for motor speed 0.5 rad/s.](image)

Figure B.1: Difference in acceleration for x for motor speed of $0.5 \text{rad/s}$. 
Figure B.2: Difference in acceleration for y for motor speed of 0.5 rad/s.

Figure B.3: Difference in acceleration for x for motor speed of 1 rad/s.
Figure B.4: Difference in acceleration for $y$ for motor speed of $1 \text{rad/s}$.

Figure B.5: Difference in angle for $x$ for motor speed of $0.5 \text{rad/s}$.
Figure B.6: Difference in angle for y for motor speed of $0.5 \text{rad/s}$. 

Figure B.7: Difference in acceleration for x for motor speed of $1 \text{rad/s}$. 

32
Figure B.8: Difference in acceleration for $y$ for motor speed of 1 rad/s.
Appendix C

Arduino code

```c
#include <I2Cdev.h>

int k = 0;

/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.
   This software may be distributed and modified under the terms of the GNU
   General Public License version 2 (GPL2) as published by the Free Software
   Foundation and appearing in the file GPL2.TXT included in the packaging
   of this file. Please note that GPL2 Section 2[b] requires that all works
   based on this software must also be made publicly available under the terms of
   the GPL2 ("Copyleft").
   Contact information
   Kristian Lauszus, TKJ Electronics
   Web : http://www.tkjelectronics.com
   e-mail : kristianl@tkjelectronics.com
   */

#include <Wire.h>

#include "Kalman.h" // Source: https://github.com/TKJElectronics/KalmanFilter
#define RESTRICT_PITCH // Comment out to restrict roll to 90deg instead –
AN3461.pdf

Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;
```
// Variables for controller

float K_A = 100;
float K_AV = 0;
float K_T = 0;
float K_V = 0;
float ax;
float ay;
float X = 0;
float Y = 0;
float x_0;
float y_0;
float x_old = 0;
float y_old = 0;

// end for controller variables

// Stepping mode

#if 0
#define M1 LOW
#define M2 LOW
#define M3 LOW
#define comp 1
#endif
#if 0
#define M1 HIGH
#define M2 LOW
#define M3 LOW
#define comp 2
#endif
#if 0
#define M1 LOW
#define M2 HIGH
#define M3 LOW
#define comp 4
#endif
#if 0
#define M1 HIGH
#define M2 HIGH
#define M3 LOW
#define comp 8
#endif
#if 1
#define M1 HIGH
#define M2 HIGH
#define M3 HIGH
#define comp 16
#endif
int micro_comp = comp;  // Variables for the motor driver
int stepCount = 0;  // number of steps the motor has taken
int stp1 = 12, stp2 = 11, stp3 = 10;  // connect pin to step
int dir1 = 9, dir2 = 8, dir3 = 7;  // connect pin to dir
float rb = 0.11, rw = 0.025;  // radie boll
float v_x = 0.1, v_y = 0, w_bx, w_by, w_bz = 0;
float theta = 0.52359;
const int m = 3, p = 3, n = 1;
float T_1, T_2, T_3;  // Tids konstanter som motorn v ntar
float m1 = 0, m2 = 0, m3 = 0;  // True eller false
float w_1, w_2, w_3;  // varvtal till resp motor (rad/s)
float TIME_1 = 1, TIME_2 = 1, TIME_3 = 1;  // tids grejer
float rpm_1, rpm_2, rpm_3;  // rpm till resp motor
float rps_1, rps_2, rps_3;  // rps till resp motor
float Vy=0; Vx=0; dt=0; AngleX
float delta_t, DT;
float delta_tOld = 0;

float pi = 3.14;
int stepsPerRevolution = 360 / 1.8;
unsigned long time, IMU_loop = 0, IMU_TIME ;
int x = 0;
//End for variables for motor driver

// Variables for IMU
float lastGyroX = 0;
float AngleX_old = 0;
/* IMU Data */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int16_t tempRaw;
double Vx_0 = 0, Vy_0 = 0;
double Ax, Ay;
double lastReadingX, lastReadingY;
double gyroXangle, gyroYangle;  // Angle calculate using the gyro only
double compAngleX, compAngleY;  // Calculated angle using a complementary filter
double kalAngleX, kalAngleY;  // Calculated angle using a Kalman filter
uint32_t timer;
uint8_t i2cData[14];  // Buffer for I2C data
//End of variables for IMU
void setup() {
  Serial.begin(115200);
  int AFS_SEL = 0;
  Wire.begin();
  #if ARDUINO >= 157
    Wire.setClock(400000UL); // Set I2C frequency to 400kHz
  #else
    TWBR = ((F_CPU / 400000UL) - 16) / 2; // Set I2C frequency to 400kHz
  #endif

  i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
  i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz
                      // Gyro filtering, 8 KHz sampling
  i2cData[2] = 0x00; // Set Gyro Full Scale Range to 250deg /s
  i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to 2g
  while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers
              // at once
  while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope
  while (i2cWrite(0x75, i2cData, 1));
  if (i2cData[0] != 0x68) { // Read 'WHO_AM_I' register
    Serial.print(F("Error reading sensor"));
    while (1);
  }
  delay(100); // Wait for sensor to stabilize

  /* Set kalman and gyro starting angle */
  while (i2cRead(0x3B, i2cData, 6));
  accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
  accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
  accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);

  // eq. 25 and eq. 26
  // atan2 outputs the value of -π to (radians) – see http://en.wikipedia.org/wiki/Atan2
  // It is then converted from radians to degrees
  #ifdef RESTRICT_PITCH // Eq. 25 and 26
    double roll = atan2(accY, accZ) * RAD_TO_DEG;
    double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
  #else // Eq. 28 and 29
    double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
    double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
  #endif

  kalmanX.setAngle(roll); // Set starting angle
  kalmanY.setAngle(pitch);
  gyroXangle = roll;
  gyroYangle = pitch;
  compAngleX = roll;
  compAngleY = pitch;
// Init the outputs for the motors
pinMode(stp1, OUTPUT);
pinMode(dir1, OUTPUT);
pinMode(stp2, OUTPUT);
pinMode(dir2, OUTPUT);
pinMode(stp3, OUTPUT);
pinMode(dir3, OUTPUT);
pinMode(5, OUTPUT);
digitalWrite(3, M1);
digitalWrite(4, M2);
digitalWrite(5, M3);

// timer = micros();
time = micros();

void loop() {

// if (micros() >= IMU_loop) {
IMU_loop = micros() + 3000; // Loop every 2 mili sek

#if 1 // Set to 1 to activate

/* Update all the values */
while (i2cRead(0x3B, i2cData, 14));
accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);
tempRaw = (int16_t)((i2cData[6] << 8) | i2cData[7]);
gyroX = (int16_t)((i2cData[8] << 8) | i2cData[9]);
gyroY = (int16_t)((i2cData[10] << 8) | i2cData[11]);
gyroZ = (int16_t)((i2cData[12] << 8) | i2cData[13]);
#endif

double dt = (double)(micros() - timer) / 1000000; // Calculate delta time
timer = micros();
//time = micros();

// atan2 outputs the value of - to (radians) - see http://en.wikipedia.org/wiki/Atan2
// It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
double roll = atan2(accY, accZ) * RAD_TO_DEG;
#else

}
double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * R4; #else // Eq. 28 and 29
    double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * R4;
    double pitch = atan2(-accX, accZ) * R4;
#endif
double gyroXrate = gyroX / 131.0; // Convert to deg/s
double gyroYrate = gyroY / 131.0; // Convert to deg/s
#else // This fixes the transition problem when the accelerometer angle jumps between -180 and 180 degrees
    if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {
        kalmanX.setAngle(roll);
        compAngleX = roll;
        kalAngleX = roll;
        gyroXangle = roll;
    } else
        kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter
        if (abs(kalAngleX) > 90)
            gyroYrate = -gyroYrate; // Invert rate, so it fits the restricted accelerometer reading
            kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
#else // This fixes the transition problem when the accelerometer angle jumps between -180 and 180 degrees
    if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {
        kalmanY.setAngle(pitch);
        compAngleY = pitch;
        kalAngleY = pitch;
        gyroYangle = pitch;
    } else
        kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate the angle using a Kalman filter
        if (abs(kalAngleY) > 90)
            gyroXrate = -gyroXrate; // Invert rate, so it fits the restricted accelerometer reading
            kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter
#endif
gyroXangle += gyroXrate * dt; // Calculate gyro angle without any filter
gyroYangle += gyroYrate * dt;
//gyroXangle += kalmanX.getRate() * dt; // Calculate gyro angle using the unbiased rate
//gyroYangle += kalmanY.getRate() * dt;
compAngleX = 0.93 * (compAngleX + gyroXrate * dt) + 0.07 * roll; // Calculate the angle using a Complimentary filter
compAngleY = 0.93 * (compAngleY + gyroYrate * dt) + 0.07 * pitch;

// Reset the gyro angle when it has drifted too much
if (gyroXangle < -180 || gyroXangle > 180)
gyroXangle = kalAngleX;
if (gyroYangle < -180 || gyroYangle > 180)
gyroYangle = kalAngleY;

// Calibrate the angle
float AngleY = kalAngleY;
float AngleX = -kalAngleX;

// Calculating the acceleration with a filter
double readAccelerationX = accX;
double readAccelerationY = accY;

double alpha = 0.1;

double acc_X = alpha * readAccelerationX + (1 - alpha) * lastReadingX;
double acc_Y = alpha * readAccelerationY + (1 - alpha) * lastReadingY;

// Setting variables to latest values
lastReadingX = accX;
lastReadingY = accY;

float AngleX_new = AngleX;
float w_x = (AngleX_new - AngleX_old) / dt;

AngleX_old = AngleX;

float Ax_no_filter = accX / 16384;
Ax = acc_X / 16384;
Ay = (acc_Y / 16384);

float Ay_cal = Ay - sin(AngleX * 0.0174532925);
float Ax_cal = Ax + sin(AngleY * 0.0174532925);

// Velocity
double Vx = Vx_0 + Ax_cal * dt;
Vx_0 = Vx;
double Vy = Vy_0 + Ay_cal * dt;
Vy_0 = Vy;

X = x_old + (100 * Vy) / 11 * dt;
Y = y_old - (100 * Vx) / 11 * dt;

ax = K_A * AngleX + K_AV * gyroXrate + K_T * (X - x_0) + K_V * Vx;
ay = K_A * AngleY + K_AV * gyroYrate + K_T * (Y - y_0) + K_V * Vy;

x_old = X;
\[ y_{old} = Y; \]
\[ \delta_t = \text{micros}(); \]
\[ DT = (\delta_t - \delta_t\text{Old}) / 1000000; \]
\[ \delta_t\text{Old} = \delta_t; \]
\[ w_1 = -ay \times DT; \]
\[ w_2 = ((\sqrt{3}) / 2) \times (ax \times DT) + (0.5 \times ay \times DT)) \times \cos(30 \times 0.017452925); \]
\[ w_3 = ((-\sqrt{3}) / 2) \times (ax \times DT) + (0.5 \times ay \times DT)) \times \cos(30 \times 0.017452925); \]
\[ w_1 = w_1 \times \text{micro\_comp}; \]
\[ w_2 = w_2 \times \text{micro\_comp}; \]
\[ w_3 = w_3 \times \text{micro\_comp}; \]

```c
} 
motor_1(time, \ w_1); 
motor_2(time, \ w_2); 
motor_3(time, \ w_3); 
```
if ((micros() >= TIME_1) && (m1 == 0)) {
    digitalWrite(stp1, LOW);
    TIME_1 = TIME_1 + T_1;
    m1 = 1;
}
}

void motor_2(unsigned long time, float w_2) {
    // decides the direction for the motor2
    if (w_2 < 0) {
        digitalWrite(dir2, LOW);
    }
    if (w_2 > 0) {
        digitalWrite(dir2, HIGH);
    }
    // decides the speed for the motor2
    rpm_2 = abs(w_2) * 9.5492;
    rps_2 = rpm_2 / 60;
    T_2 = 1000000 / (400 * rps_2); // 1/(200*rps)*1000;
}

if ((micros() >= TIME_2) && (m2 == 1)) {
    // Serial.println(T_1);
    digitalWrite(stp2, HIGH);
    TIME_2 = TIME_2 + T_2;
    m2 = 0;
}
if ((micros() >= TIME_2) && (m2 == 0)) {
    digitalWrite(stp2, LOW);
    TIME_2 = TIME_2 + T_2;
    m2 = 1;
}
}

void motor_3(unsigned long time, float w_3) {
    // decides the direction for the motor3
    if (w_3 < 0) {
        digitalWrite(dir3, LOW);
    }
    if (w_3 > 0) {
        digitalWrite(dir3, HIGH);
    }
    // decides the speed for the motor3
    rpm_3 = abs(w_3) * 9.5492;
}
rps_3 = rpm_3 / 60;
T_3 = 1000000 / (400 * rps_3); //1/(200*rps)*1000;

if ((micros() >= TIME_3) && (m3 == 1)) {
    digitalWrite(stp3, HIGH);
    TIME_3 = TIME_3 + T_3;
    m3 = 0;
}

if ((micros() >= TIME_3) && (m3 == 0)) {
    digitalWrite(stp3, LOW);
    TIME_3 = TIME_3 + T_3;
    m3 = 1;
}