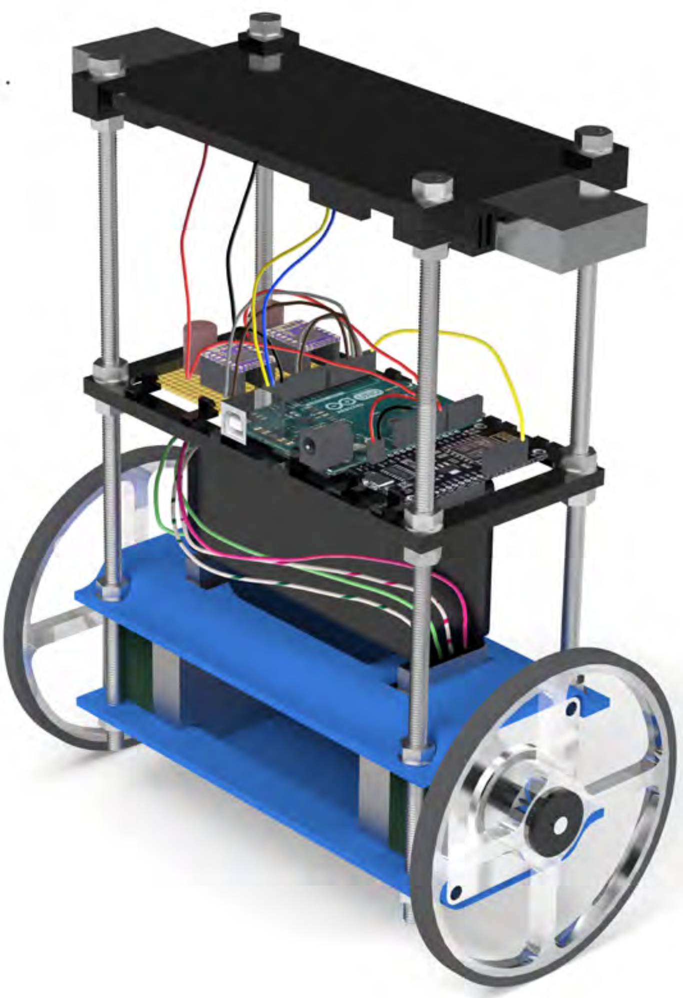
DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2020

Self-balancing robot

FREDRIK IHRFELT

WILLIAM MARIN



KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

Two wheel self-balancing robot

WiFi steerable self-balancing robot.

FREDRIK IHRFELT, WILLIAM MARIN

Bachelor's Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA ITM-EX 2020:36

Abstract

This thesis aims to creating a self-balancing robot. The movement of the two wheeled self balancing robot resembles the human movement more than an traditional four wheeled vehicle. The Principe of balance is based on the inverted pendulum. Balance is achieved by using a PID controller with inputs from a gyroscope and accelerometer. Stepper motors are used to move the robot. This thesis researches how well the robot can transport packages, as well as controlling it wireless. To test the theory in practice, a demonstrator was built.

Keywords
Mechatronics, self-balancing, robot, package delivery, remote controlled

Referat

Tvåhjulig själv-balanserande robot

Denna uppsats strävar efter att skapa en själv-balanserande robot. Rörelsen av en tvåhjulig själv-balanserande robot liknar den mänskliga rörelsen mer än en traditionell fyrhjuligt fordon. Balansprincipen är baserad på principen för en inverterad pendel. Balans uppnås genom att en PID regulaor med input från ett gyroskop och accelerometer samt stegmotorer som reagerar på vinkelförändrigar. Denna uppsats kommer undersöka hur väl roboten kan leverera paket samt hur roboten kan fjärrstyras. För att testa teorin i praktiken byggdes en demonstrator.

Nyckelord
Mekatronik, själv-balanserande, robot, paketleverans, fjärrstyrd

Acknowledgements

We would like to thank Nihad Subasic, supervisor and examiner, for giving us the right knowledge and support that made this project possible. Seshagopalan Thorapalli among other students have helped us a lot with the project. We also want to thank the ITM School at KTH for financial support. Thanks to all mechatronics students that has helped us along the way.

Contents

List of Figures

List of Tables

Acronyms

3D Third Dimensional. 12

ADC Analouge-Digtal Converter. 15

CAD Computer Aided Design. 2

IMU Inertial Measurement Unit. 1

OLED Organic Light-Emitting Diode. 15

PLA Polylactic Acid. 12

PMMA Polymethyl Methacrylate. 12

SEK Swedish Krona. 2

TCP Transmission Control Protocol. 9, 22

WiFi Wireless Fidelity. 1

Chapter 1

Introduction

1.1 Background

The principle of a two wheeled self-balancing robot is similar to that of the human body. The human body is an inverted pendulum balancing the upper body around the ankles[1]. Two wheeled self-balancing robots have advantages compared to other configurations, they are mechanically simple, have a zero turn radius and have superior stability in inclines since they lean into the incline[2]. This could make two wheeled self-balancing robots useful for for automated package deliveries. In order to maintain balance a gyroscope and accelerometer, in form as an Inertial Measurement Unit (IMU) is used. This deviates the angle of the robot. This angle is then used to increase/decrease speed of the motors with the goal of maintaining a constant angle[3]. The potential future use of this type of technique combined with the interesting similarities with the human movement was a motivation from authors to choose this topic.

1.2 Purpose

The purpose of this thesis is to evaluate the possibility of using two wheeled self-balancing robots for package deliveries. This was done by testing a self-balancing robots ability to maintain balance with an off centered weight on top and evaluating the viability of remote control over WiFi. The research questions to be asked are:

- How much weight can be added off center while still maintaining position for the robot?
- What is the response time when controlling the robot with a Wireless Fidelity (WiFi) connection?
- How do WiFi and Bluetooth steering compare?

A package with uneven weight distribution and desired robot response is shown in figure 1.1.

Figure 1.1. Figure showing uneven package and desired robot response. Made in Solid Edge[15].

1.3 Scope

The focus of the project was to build a demonstrator and implement a code that enables the demonstrator to balance and to be controlled over a wireless connection. There was also an aim to enable the robot to carry packages without loosing balance. The budget for the components needed to be bought for the project is set to 1 000 Swedish Krona (SEK) and the time constraint for the project was five months. Given the budget and time constraint the demonstrator will not be able to maneuver autonomously.

1.4 Method

First research was conducted to understand the system and the electronics. Then dynamic equations were derived. These equations were used to design a PID controller that would enable balancing of the robot. The demonstrator was designed as a Computer Aided Design (CAD) model in Solid Edge in order to determine the parameters needed for the dynamic equations. The physical demonstrator was built with the CAD model as a guideline. Experiments with the physical demonstrator was then conducted in order to determine the demonstrators ability to balance with an off center weight, as well as determine the the performance of WiFi communication.

Chapter 2

Theory

The principle of creating a two wheeled self-balancing robot is similar to the inverted pendulum principle, see figure 2.1. When a a tilt from the equilibrium occurs the motors will generate a torque that drives the wheels in the same direction as the tilt, the wheels will move the same distance as the center of gravity in order to maintain balance[5]. To enable the robot to move forward a joystick will be used to accelerate and change direction of the robot. In order to achieve forward movement the angle set point will be increased, changing the equilibrium point. This will lead to the wheels constantly moving in the same direction as the lean of the robot to maintain the angle set point. The theory that covers the design of our robot will be explained in this chapter.

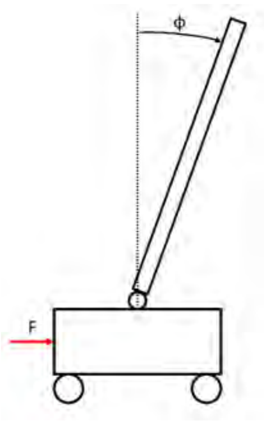


Figure 2.1. Figure showing model for inverted pendulum. Made in Illustrator[4].

2.1 Mechanical system

In order to design a control system, a mechanical model of the robot that describes the movement on the robot and the forces that act on the pendulum and the wheels was derived.

The system was modeled as cart and a pendulum, see figure 2.2. The system was simplified by only studying the forces and angles in the x- and y-axis. When deriving the mechanical system it was assumed that the wheels never loose contact with the ground, resulting in the wheels not having any movement in the y-axis. Any disturbances due to change of the center of gravity was neglected[6].

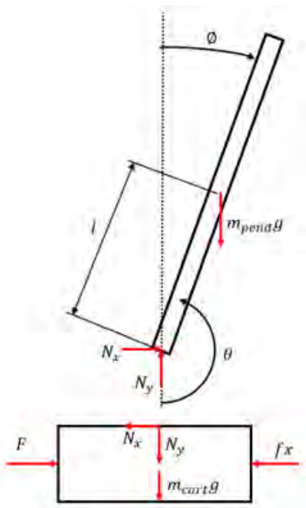


Figure 2.2. Figure showing the forces acting on the system. Made in Illustrator[4].

2.2 Control system

The systems dynamic equations 2.1 and 2.2 were derived from the forces acting on the system, seen in figure 2.2. See Appendix A for details.

$$F_{input} = (m_{cart} + m_{pend})\ddot{x} + f\dot{x} + m_{pend}l\ddot{\theta}\cos\theta - m_{pend}l\dot{\theta}^2\sin\theta \tag{2.1}$$

$$(I_{pend} + m_{pend}l^2)\ddot{\theta} + m_{pend}gl\sin\theta = -m_{pend}l\ddot{x}\cos\theta \tag{2.2}$$

Where $\ddot{x}$ is the systems acceleration along the horizontal plane and $\ddot{\theta}$ is the pendulums angular acceleration. The system could now be written on the state-space model.

In order to apply linear control theory an assumption was made that the robot will deviate little from the equilibrium, see Appendix B for details. The systems transfer functions, equation 2.3 and equation 2.5, could then be derived. See Appendix C for details.

$$G_\phi(s) = \frac{\frac{m_{pend}ls}{q}}{s^3 + \frac{f(I_{pend}+m_{pend}l^2)}{q}s^2 - \frac{(m_{cart}+m_{pend})m_{pend}gl}{q}s - \frac{fm_{pend}gl}{q}s} \tag{2.3}$$

where

$$q = (m_{cart} + m_{pend})(I_{pend} + m_{pend}l^2) - (m_{pend}l)^2 \tag{2.4}$$

$$G_X(s) = \frac{\frac{(I_{pend}+m_{pend}l^2)s^2 - gm_{pend}l}{q}}{s^4 + \frac{f(I_{pend}+m_{pend}l^2)}{q}s^3 - \frac{(m_{cart}+m_{pend})m_{pend}gl}{q}s^2 - \frac{fm_{pend}gl}{q}s} \tag{2.5}$$

2.3 PID-controller

The stability of the system is determined by the placement of the poles. They can not be placed too near the origin or the system will be too slow. Nor can they be placed too far from the origin since this will demand a too high input current. The imaginary part of the poles can not be too large since it may lead to oscillation[7].

In order to stabilize the system a PID-controller, equation 2.6, was implemented, where the input $u(t)$ is given by the deviation between the wanted and the actual output $e(t) = r(t) - y(t)$ according to [7]

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d + K_D \frac{d}{dt}e(t) \tag{2.6}$$

Where K_P is the proportional coefficient of the regulator, K_I is the integral coefficient and K_D is the derivative coefficient.

By increasing the proportional term of the regulator the control signal for the same level of error is increased proportionally, this leads to a quicker reaction but with more overshoot. An increase of the proportional term can also reduce the steady-state error. If the integral term is increased the static error can be reduced, but the stability margin of the system will decrease leading to the system becoming more oscillatory. By increasing the derivative term of the regulator the stability margin will increase, but the affect of measurement errors will increase[8].

The regulator was tuned by setting K_I and K_D to zero and increasing K_P until steady oscillations are achieved. Then K_P was halved and K_I and K_D was tuned empirically[9].

2.4 Weight placed off center

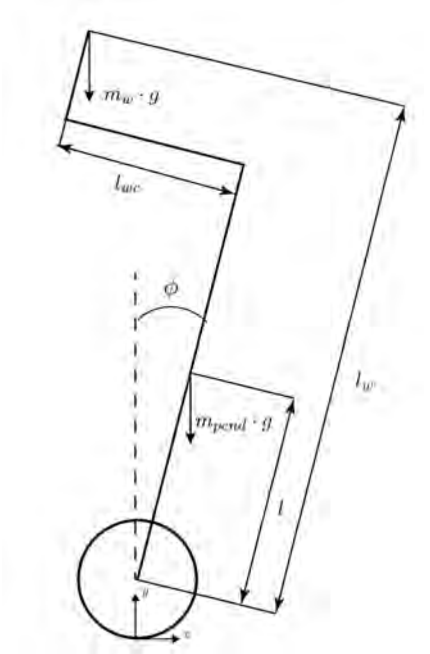


Figure 2.3. Figure showing the forces with weight placed off center. Made in Illustrator [4].

In order to maintain balance when a weight is placed off center relative the vertical axis of the robot, the angle set point of the PID controller will be increased or decreased when there is no steering input from the hand controller but the robot is moving either forward or backward. The angle set point will be increased or decreased by a constant, ϕ_c , 250 times per second, the frequency of the code, until the robot is at standstill. This will lead to the robot finding its balancing point where it is at standstill even if a weight was to be placed off center on the robot.

By increasing the value of ϕ_c the angle set point will be changed in larger steps until the robot is at standstill, this will reduce the time it takes for the robot to find its balance point but might induce oscillations and overshoot.

In order to calculate a theoretical settling time a mechanical model of the robot that describes the angle of the robot and the torque acting on the pendulum was derived, see figure 2.3. The system was modeled as a pendulum and a weight connected to the pendulum. To simplify the system the mass of the beam connecting the weight to the robot was neglected and only the angles and torque in the x- and y-axis was calculated.

Equation 2.7 describing the relationship between the angle for the balance point and the length the from the pendulums center axis to the weight was then derived. See Appendix E

$$\phi = \arctan\left(\frac{m_w l_{wc}}{m_{pend} l + m_w l_w}\right) \tag{2.7}$$

Where ϕ is the lean angle, m_w is the mass of the weight, m_{pend} is the mass of the pendulum, l is the length from the center of the wheel to the center of mass for the pendulum, l_w is the length from the center of the wheel to center of the weight and l_{wc} is the length from the top of the pendulum to the center of the weight.

Equation 2.8 could then be derived in order to calculate the theoretical settling time from the balance point angle, code frequency and the constant ϕ_c.

$$T_{s_t} = \frac{\phi}{\phi_c} f \tag{2.8}$$

Where T_{s_t} is the theoretical settling time, ϕ is the lean angle and f is the frequency of the code.

2.5 Bipolar stepper motors

Two bipolar stepper motors were used to balance the robot, the core of the motor is a permanent magnet surrounded by two coils divided into a number of small coils that act as electromagnets when current is passed trough them. By controlling the current passed through the coils the motor can be rotated in discrete steps[10].

One coil is energized and the magnetized core is attracted to the coil making the core lock into place, see figure 2.4. Then the electricity is removed from that coil and applied to the next coil, resulting in the core being attracted to the next coil and rotating one step, see figure 2.5. By pulsating the current applied to the coils the stepper motor rotates[10].

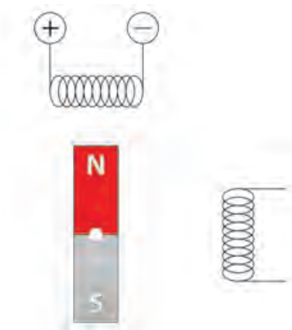


Figure 2.4. Figure showing a model of the stepper motor. Made in Illustrator[4].

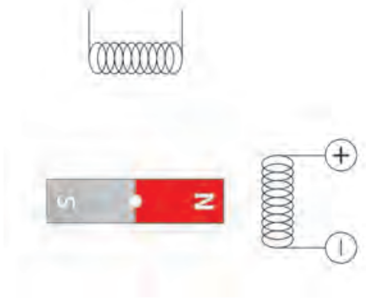


Figure 2.5. Figure showing a model of the stepper motor. Made in Illustrator[4].

The motor used rotates 2° per step, in order to reduce the angle traveled per step microstepping was used. By applying the same current to both coils at the same time the rotor locks into place between the two coils resulting in a half step, see figure 2.6. This method was used in order to achieve quarter steps, resulting in a smoother motion[10].

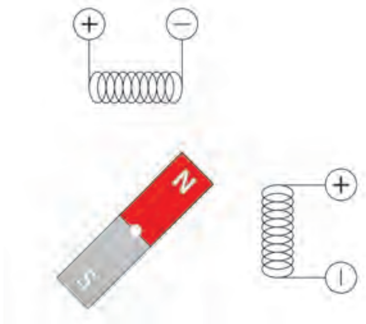


Figure 2.6. Figure showing microstepping of the stepper motor. Made in Illustrator[4].

2.6 Stepper motor driver

In order to control the stepper motors the DRV8825 driver were used. The driver mainly consists of a microstepping indexer and two H-bridges, one for each coil group of the stepper motor. The H-bridges can change the current flow through the coils and therefore change the polarity of the coils. This determines the direction the stepper motor rotates. A figure displaying a H-bridge is shown in figure 2.7. By increasing or decreasing the pulse frequency the motor speed will increase or decrease respective[11].

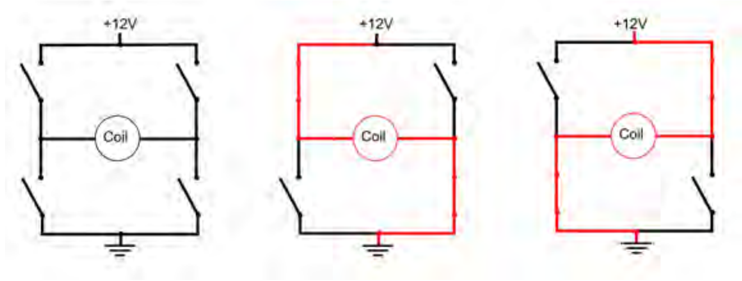


Figure 2.7. Figure showing the function of an H-bridge. Made in Illustrator[4].

2.7 WiFi communication

In order to steer the robot remotely WiFi communication will be used, a radio based wireless network. To obtain a WiFi connection between two devices the receiver acts as a access point that the other device can connect to. The other device acts as a station that connects to the access point. There are several different WiFi protocols, a set of rules for how the WiFi communication works. In this thesis the Transfer Control Protocol (TCP) will be used, it is not as fast as other protocols but it makes sure that all the data sent and uncorrupted. If sent data was to be missing or corrupted it will ask for the data to be re-sent.[12]

WiFi latency is the time that passes between a user action at the station and the response at the access point, a higher latency leads to a longer delay from when the user sends a steering command to when the command is executed by the robot. One of the principle causes of higher latency is the distance between the access point and station.[13]

2.8 Steering of the robot

To enable remote control of the robot two ESP8266 WiFi modules[14] will be used, one in the hand controller and one in the robot. The WiFi module in the hand controller will act as a access point, and the WiFi module in the robot will act as a station. Steering signals will be sent remotely from the hand controller to the WiFi module in the robot. When a steering signal is received the WiFi module in the robot will send the command to the arduino using serial communication.

To enable forward and backward movement the angle set point of the PID-controller will be increased or decreased, resulting in the robot leaning forward or backward in order to maintain the angle set point. This will make the robot move in the leaning direction.

To enable turns the output signals to the motors are divided for the left motor and the right motor. By reducing the pulse frequency to one motor and increasing the frequency to the other motor it will enable the robot to turn.

Chapter 3

Demonstrator

The following chapter will list and describe the chosen pre-fabricated components as well as the components that where constructed and manufactured. The demonstrator consists of a robot and a remote. Both where constructed in CAD. A render of the CAD displayed in figure 3.1. A complete list of all components used is found in Appendix F.

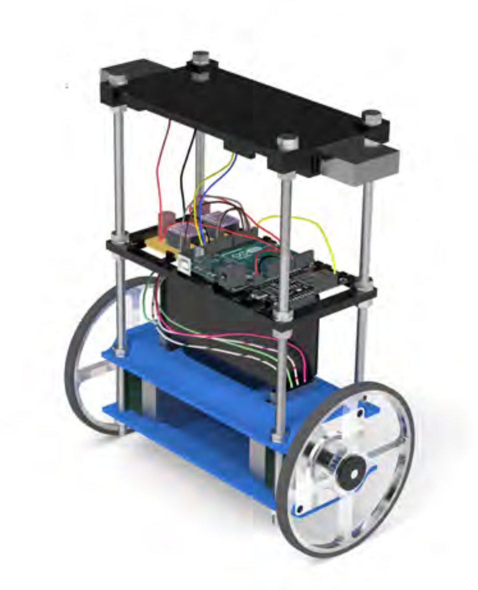


Figure 3.1. Figure showing CAD render. Made in Solid Edge[15].

3.1 Mechanical components

A handfull of different mechanical components where used to construct the demonstrator.

3.1.1 Baseplates

Four different base plates were designed to create layers for the demonstrator. Two base plates were designed to clamp the two motors in place. Another base plate was designated as a frame for mounting the Arduino, WiFi module and stepper motor drivers. This base plate also increased stability for the robot, by splitting the natural frequency in half for the threaded rod. The top base plate was designed to mount the IMU and a counter balance weight. This balance weight was used to raise the center of gravity as well as a mount enabling the mounting of the sliding weight used for the tests. All base plates where 3rd Dimensional (3D) printed in Polylactic Acid (PLA).

3.1.2 Wheels

Wheels where designed consisting of two components, rim and tire. Weight was intended to be low to reduce the moment of inertia to improve acceleration. The two rims where laser cut in Polymethyl Methacrylate (PMMA). This material was chosen for the wheels because of the high yield strengh[16]. The tires where cut from an inner tubes from an bicycle. These where then glued to the rims. This material provided high friction between the wheels and the maneuvering base.

3.1.3 Assembly of the demonstrator

Four threaded rods, 24 nuts and 24 washers where used to tighten the base plates in a vertical stack, and clamp the motors. The wheels were clamped together by a radius difference between the motor shaft and the inner hole on the rim, the tolerance of the fit was H6.

3.2 Electrical Components

3.2.1 Arduino Uno

The microprocessor used in this thesis was the Arduino Uno running the ATmega328. The Arduino has 14 digital in-/output pins and 6 analogue pins. It operates with a clock speed of 16 MHz and supports I^2C communication used with the accelerometer and gyro module[17]. Arduino Uno is shown in figure 3.2.

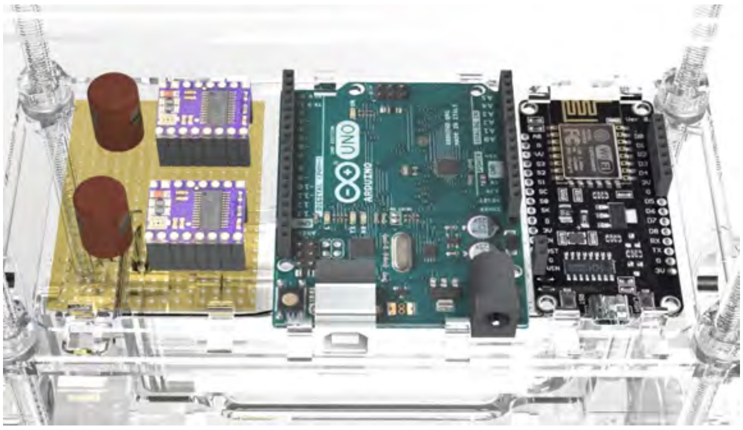


Figure 3.2. Figure showing electrical components used. Made in Solid Edge[15].

3.2.2 Sensor for angular data

In order to determine the angular deviation and angular velocity an IMU was used, the MPU6050 is a three axis accelerometer and gyro. It communicates with the Arduino via I^2C at 400 KHz, the range for the angular velocity was set to $\pm 250°/s$ and the range for the accelerometer was set to $\pm 4g$[18].

3.2.3 Stepper motors and driver

The stepper motors used was the Japan servo KH56JM2-851[19], a bipolar stepper motor with a step angle of 1,8°. The motor used is shown in figure 3.3. In order to drive the stepper motors two DRV8825 drivers were used, 1/8 microstepping was enabled on the drivers in order to reduce the step angle to 0,225°. The key specifications for the stepper motors and motor drivers are shown in table 3.1 and table 3.2 respectively.

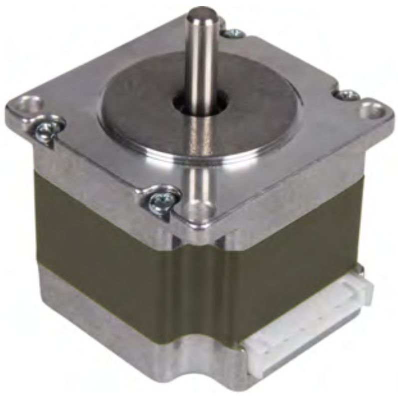


Figure 3.3. Figure showing the stepper motor[20].

Table 3.1. Table showing key specifications for Japan servo KH56JM2 851 stepper motor.

Number of phases	2
Step angle [°/step]	1,8
Voltage [V]	3,51
Current [A/phase]	1,3
Inductance [mH/phase]	5,6
Holding torque [kgf·cm]	5,0
Max speed at 12 V [rpm]	253

Table 3.2. Table showing key specifications for DRV8825 stepper motor driver.

Operating voltage [V]	8,2 - 45
Max drive current [A]	2,5
Max microstepping	1/32

3.2.4 Battery

A standard 12V 2200mAh LiPo battery was used to power our robot. LiPo batterys has the advantage of having a light weight compared to their capacity[21]. A picture of the battery is shown on figure 3.4.

Figure 3.4. Figure showing 2200mAh LiPo Battery[21].

3.2.5 Remote control and receiver

The remote consists of one ESP8266, an Analogue to Digital Converter (ADC), an analogue joystick[22] and a 5v power bank. On the receiving side an ESP8266 was used together with a resistor divider connected to the battery. The remote control also has a Organic Light-Emitting Diode (OLED) screen that displays data. A render of the remote is shown in figure 3.5 and the display is shown in 3.6.

Figure 3.5. Figure showing the remote control attached to the 5v powerbank. Made in Solid Edge[15].

Figure 3.6. Figure showing the remote display with latency in milliseconds. Made in Solid Edge[15].

3.3 Software

3.3.1 Robot Software

The flow chart in figure 3.7 explains the logic of the Arduino code. The code was based on Joop Brokkings code for a self-balancing robot[23], the code was modified to receive steering inputs from the ESP8826 via serial communication.

- System initialized by including libraries, initializing variables and configuring in- and outputs
- The IMU is calibrated by taking 500 measurements at standstill and correct the gyro and accelerometer offset values
- Data is read from the IMU and apply a complimentary filter and low pass filter to reduce noise and gyro drift before the angle is calculated
- A condition checks if the the lean angle is withing $0,5° > \phi > -0,5°$, this is to reduce the effect of measurement errors and high frequency shaking
  - If true the motor speed is set to zero and break from the loop
  - If false continue the loop
- A condition to check if there is serial data sent from the ESP8266
  - If true change the angle set point for forward/backward movement or change the speed of the two motors for rotational movement
  - If false continue the loop
- A condition to check if the angle set point is zero and the motor speed > 0 in order to find the robots balance point even with an off centered weight
  - If true increase or decrease the angle set point by ϕ_c depending on what direction the motors are turning
  - If false continue the loop
- Calculate the stepper motor pulse frequency with PID controller
- Control the stepper motors with the DRV8825 driver
- A condition to set the loop time to 40 ms, 250 Hz

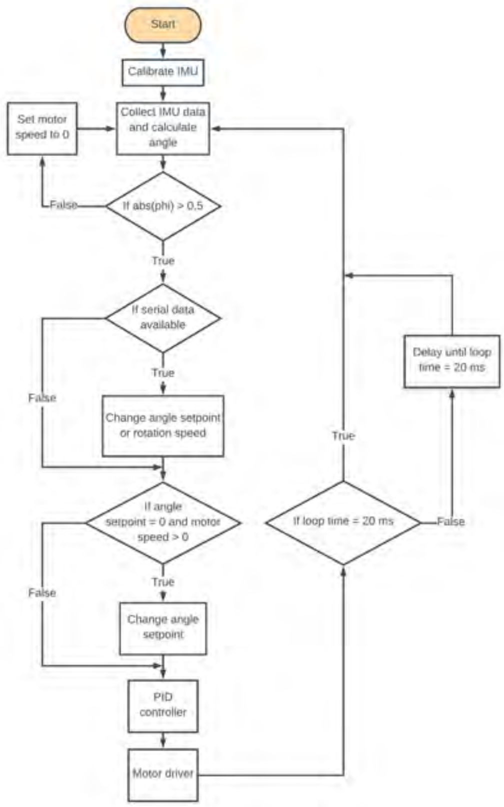


Figure 3.7. Figure showing flow chart for the arduino code. Made in Lucidchart[24].

3.3.2 Remote Software

The joystick output varies between 0-5v depending on position. This data is converted to an integer in the ADC, then scaled to a number between 0-2. This data is then transmitted as a client request from the transmitting ESP8266, and the server ESP8266 receives and responds to the request as a client. The data is then sent to the Arduino via serial communication if is it different form the previous data. The server ESP8266 is then responding with the text with a integer that changes between battery level and response time every loop. When this respond reaches the remote, it displays on screen. See Appendix H for the full code.

Chapter 4

Results

This chapter contains the results from the test performed with the demonstrator, the values of the demonstrators mass and its center of gravity is shown in table 4.1.

Table 4.1. Table showing the demonstrators mass, the length to the center of gravity and the moment of inertia.

m_{cart} [Kg]	0,917
m_{pend} [Kg]	1,501
I_{pend} [Kgm^2]	0,00002397
l [m]	0,1468
l_w [m]	0,36

4.1 Off mass center test

In order to evaluate the robots ability to balance with a weight placed off its center axis an experiment was conducted. In the experiment a one kilogram weight was placed a fixed distance from the center vertical axis of the robot. The angular data was plotted and the distance the robot travelled due to adding the weight was measured. A picture of the weight mounted on the robot is shown in figure 4.1.

Figure 4.1. Figure showing module for off mass center test. Made in Solid Edge[15].

4.1.1 Normal Weight offset

To test how the value off the constant ϕ_c would affect the settling time and oscillation a test was performed where the weight was placed one centimeter from the center axis of the robot. The graphs from two tests with different values of ϕ_c are shown in figure 4.2 and 4.3. The theoretical settling time, real settling time, oscillation and distance travelled are shown in table 4.2.

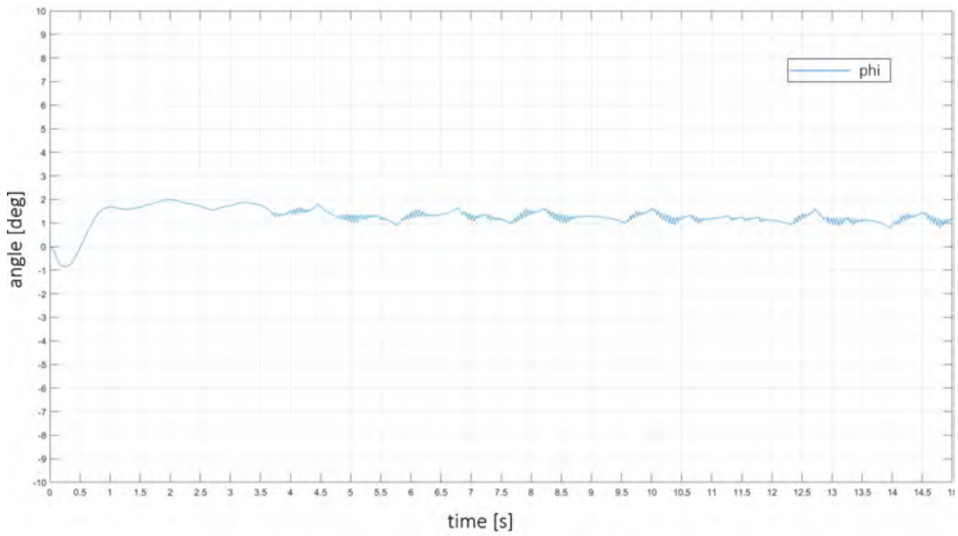


Figure 4.2. Figure showing the response of the weight placed one centimeter from center and a ϕ_c value of 0,0015. Made in Mathlab[25].

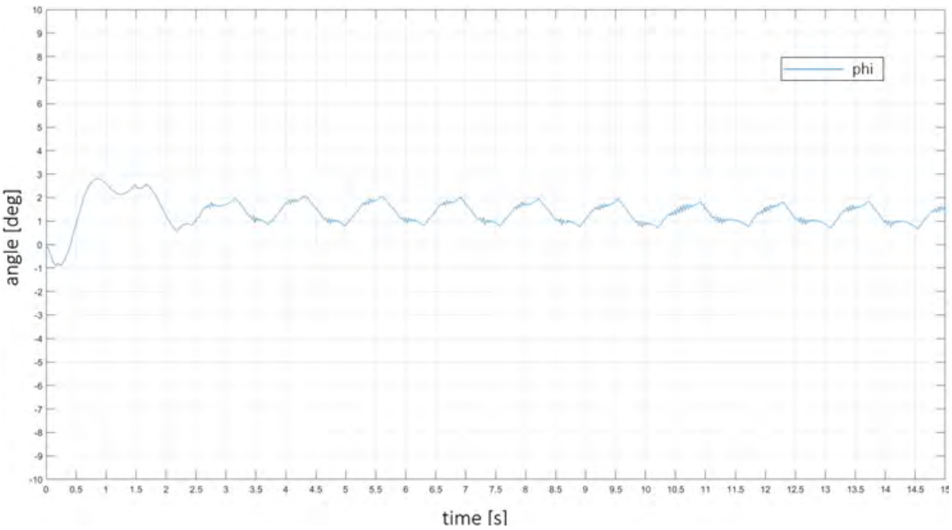


Figure 4.3. Figure showing the response of the weight placed one centimeter from center and a ϕ_c value of 0,0060. Made in Mathlab[25].

Table 4.2. Table showing angular data for one cm offset. $\Delta\phi$ is the oscillations and O is the overshoot.

l_{w_c} [cm]	ϕ_c [°]	$\Delta\phi$ [°]	O [%]	T_{s_t} [s]	T_{s_r} [s]	d [cm]
1	0,0015	0,65	42,9	2,94	3,75	46
1	0,0060	1,1	100	0,73	2,6	10

4.1.2 Maximum weight offset

The requirements for the maximum weight offset test was that the oscillations must be less that four degrees and that the travelled distance must be less than 100 cm. The graph from the test with the maximum weight offset is shown in figure 4.4 and the theoretical settling time, real settling time, oscillation and distance travelled are shown in table 4.3.

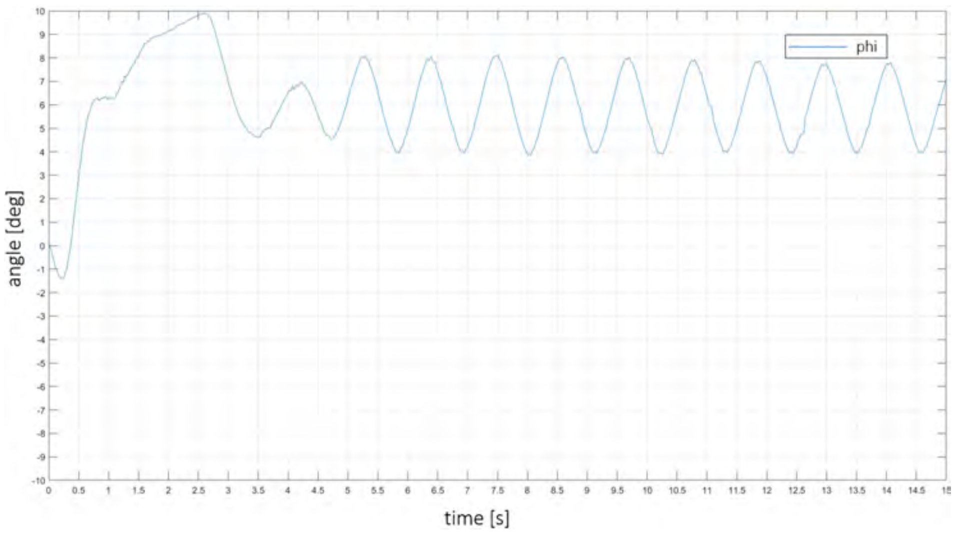


Figure 4.4. Figure showing the response of the weight placed five centimeters from center and a ϕ_c value of 0,0115. Made in Mathlab[25].

Table 4.3. Table showing angular data for five cm offset. $\Delta\phi$ is the oscillations and O is the overshoot.

l_{w_c} [cm]	ϕ_c [°]	$\Delta\phi$ [°]	O [%]	T_{s_t} [s]	T_{s_r} [s]	d [cm]
5	0,0115	4	65	1,9	3	95

4.2 Remote latency test

Latency was tested by measuring the time between the TCP request is sent from the remote until the response has been returned, see figure 4.2. The response time is then printed to the display with the 20 request average, and the maximum response time in this window. The maximum response time is 5000 ms, after that the connection is considered lost. The antennas used was the standard PCB antennas, but these can be exchanged to an external antenna for greater range.

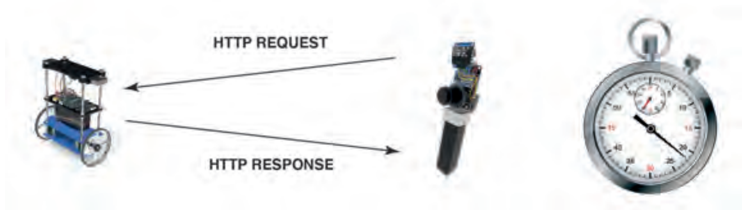


Figure 4.5. Figure showing the the remote latency test. Made in Illustrator[4].

4.2.1 WiFi Latency

The average response time remained on between 10 and 20 ms for most of the test. When the range was greater than 35 meters, response spikes begin to appear which affected the average. These appeared when the robot moved to a new position. When the robot did not move, the response time reduced to between 20-30 ms after around 10 seconds. This spike in response time was reoccurring a few times in the 35-50 meter span. When the robot moved to a distance of 50 meters, the response time became greater than 5000ms and the connection timed out.

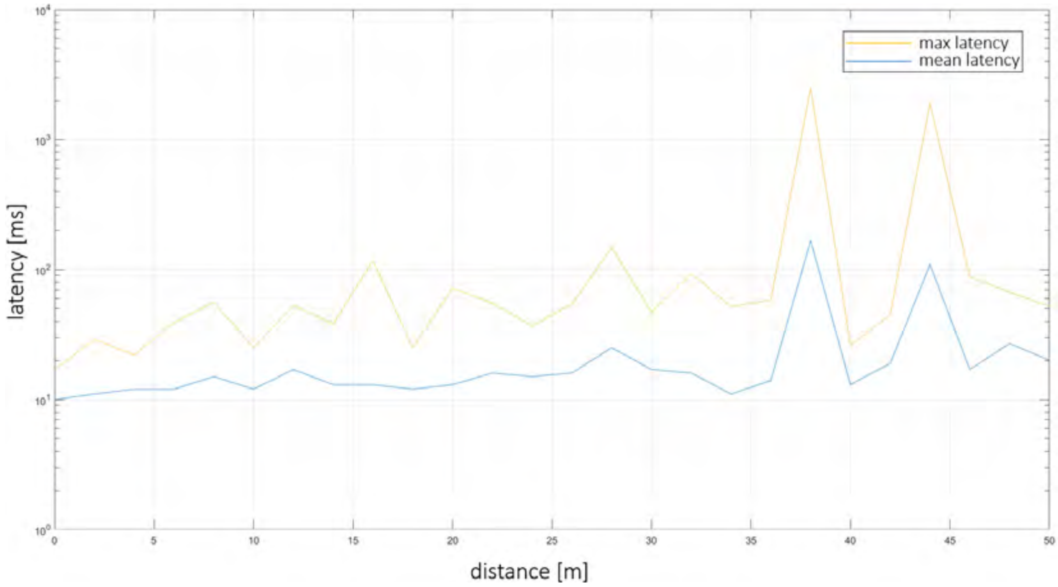


Figure 4.6. Figure showing the latency. Made in Mathlab[25].

4.2.2 WiFi Range

The WiFi connection was stable until around 35 meters from client to the robot. After that an unstable connection was able to be created until 50 meter away from the client, when the signal was disrupted and the connection to the server was lost. A illustration of signal radius and area is shown in figure 4.2.2.

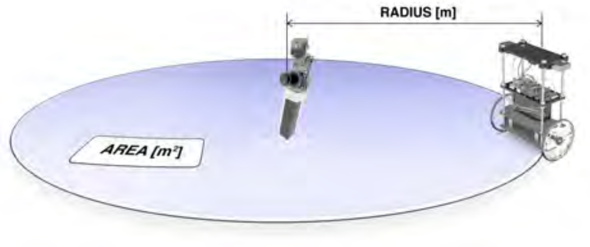


Figure 4.7. Figure showing the signal radius and area. Made in Photoshop[26].

Table 4.4. Table showing WiFi range

Connection type	Radius $[m]$	Area $[m^2]$
Stable	0-35	4000
Unstable	35-50	4000
Stable, Unstable	0-50	8000

Chapter 5

Discussion and conclusions

This chapter will discuss the results from the tests performed with the demonstrator, the chapter will also present a conclusion from the test results.

5.1 Discussion

5.1.1 The stepper motors impact

From the testing it became evident that the length of the stepper motor step had an in impact on the robots balancing performance. By using a smaller step via microstepping the robots movement became more fluid and led to the robot having a faster settling time after a disturbance and lower oscillations. The smaller step also improved the robots ability to turn. This is likely due to the larger step length leading to the robot overshooting since the positioning of the stepper motors is not accurate enough.

When using DC motors for a two wheeled self-balancing robot the two motors can have different performance, resulting in the two wheels rotating at different speeds making the robot rotate [27]. When using stepper motors this problem did not occur since the two motors always rotates in discrete steps, resulting in them always rotating at the same speed regardless of performance differences.

5.1.2 The ability to balance with off center weights

The method of changing the PID controllers angle set point in order to balance the robot with an off centered weight was successful. The increase of ϕ_c led to larger oscillations and overshoot. This is likely caused by the the larger value of ϕ_c results in the angle set point being changed too much for a small forward speed, resulting in the angle set point constantly being changed by a large amount when the robot is close to standstill. Since the robot can not be completely still this will cause oscillations to occur.

By doubling the value of ϕ_c the traveled distance from when the weight was placed on the robot was approximately halved, the increase of ϕ_c also reduced the settling time slightly. The robots ability to maintain balance after a disturbance was also affected by the increase of ϕ_c, when using a higher value the robots stability was reduced and a small disturbance led to large oscillations. From the testing it was determined that the highest usable value of ϕ_c was 0,0115.

When comparing the theoretical settling time,T_{s_t}, to the real settling time, T_{s_r}, it was evident that the theoretical settling time was shorter than the actual. The difference between the theoretical and real settling time increased as ϕ_c was increased, this is probably caused by the larger overshoot and oscillations when using a larger ϕ_c. If a PID controller was used where speed of the robot was the input and ϕ_c was the output the real settling would probably be closer to the theoretical.

Remote control of the robot was affected by the off centered weight, when the weight was placed over four centimeters from the center axis the robot would fail to maintain balance when a full forward command was given to the robot. This was probably related to the motors not being able to rotate fast enough to maintain balance.

5.1.3 Is it a good idea to control a robot with WiFi?

Previous projects have mostly used Bluetooth as a protocol for steering the robot. Bluetooth has a typical effective range of around 10 meters[28], or 79 square meters. This makes WiFi a much better choice to control the robot with, since it enables it to move further from the operator. 4g broadband cellular network has a response time of 40-50 milliseconds outdoors. By May 2020, 4g cellular covers around 80 percent of Sweden[29]. To control the range with 4g cellular would give the robot the ability to be controlled from a remote distance, but the hardware is expensive and requires a subscription with the cellular operator. The cellular signal is easily blocked indoors. WiFi is a good choice when a low response time is needed and the operator is at line of sight of the robot.

5.2 Conclusions

The robot was able to balance with a weight placed off its center axis but the steering ability was reduced, in order to use a two wheeled self-balancing robot for package deliveries this will need to be addressed. Steering the robot with WiFi works well when the operator is in line of sight to the robot, or the robot is moving indoors. If the robot is suppose to move outside in a city 4g would theoretically be a better choice.

Chapter 6

Recommendations and future work

6.1 Recommendations

A external antenna is recommended for the robot and the controller, since it would give the robot a longer and more stable range. By adding a motor to the counter-weight it could be used to balance and decreasing the need to change the regulator. This would enable the robot to stand vertically even when loaded. A sun-shield for the display is also recommended, since the screen was hard to read in daylight conditions.

6.2 Future work

The value of ϕ_c could be regulated via a PID controller where the input would be the speed of the motors, this would reduce the oscillations and overshoot while still being able to have a fast response time to a weight being placed on the robot.

A cascaded PID controller could be used that has the lean angle and the position of the robot as inputs, to calculate the position either encoders could be used or the steps of the stepper motors could be used. This would give the robot the ability to return to its starting position after a weight has been placed on top of the robot. It would also make autonomous control of the robot possible.

Bibliography

[1] Loram, I, 2002. ”Human balancing of an inverted pendulum: position control by small, ballistic-like, throw and catch movements”, Journal of Physiology. [Online]. DOI: 10.1113/jphysiol.2001.013077

[2] Daniel Jones and Karl Stol. ”Modelling and Stability Control of Two-Wheeled Robots in Low-Traction Environments”, Autralasian conference on robotics and automation, 2010. Date accessed: 2020-02-10. [Online]. Available: https://www.researchgate.net/publication/267941440_Modelling_and_Stability_Control_of_Two-Wheeled_Robots_in_Low-Traction_Environments

[3] Bruno Bosco, André Enllnefjärd, Victor Ellqvist Nordenmark, Alexander Hemberg, Henrik Olsson, Jonas Tegelberg ”Utveckling av en självbalanserande tvåhjulig robot”, KTH, Tech Rep., 2012. Date accessed: 2020-01-15. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:550532/FULLTEXT01.pdf

[4] Illustrator, Adobe. [Software]. Available: https://www.adobe.com/products/illustrator.html

[5] Rui Zhang, Gang Xiong, Changjian Cheng, Xiuqin Shang, Yonghong Ma, Zichen Lu. ”Control system design for two-wheel self-balanced robot based on the stepper motor”, IEEE, 2008. Date accessed: 2020-01-25. [Online]. DOI: 10.1109/SOLI.2013.6611417

[6] F. Grasser, A. D’Arrigo, S. Colombi and A.C. Rufer, ”JOE: a mobile, inverted pendulum”, IEEE, 2002. Date accessed: 2020-02-15. [Online]. DOI: 10.1109/41.982254

[7] Torkel Glad and Lennart Ljung, Reglerteknik. Poland: Dimograf, 2014, p. 187-188.

[8] Control tutorials for MATLAB and SIMULINK. Introduction: PID Controller Design. Date accessed: 2020-02-10. [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID

[9] Albin Cassirer, Viktor Eriksson, Mikael Gnospelius, Anna Kramarz and Johan Landerholm, ”Timing Grey: En självbalanserande robot med ultraljud och egenutvecklad hastighetsmätare”, KTH, Tech Rep., 2012. Date accessed: 2020-01-25. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:550518/FULLTEXT01.pdf

[10] DroneBot Workshop. Stepper Motors with Arduino - Getting Started with Stepper Motors. Date accessed: 2020-01-20. [Online]. Available: https://dronebotworkshop.com/stepper-motors-with-arduino/

[11] Texas instruments. DRV8825 stepper motor controller IC. Date accessed: 2020-01-05. [Online]. Available: https://www.electrokit.com/produkt/stegmotordrivare-drv8825/

[12] CircuitDigest. Getting Started with ESP8266 WiFi Transciever (Part 1). Date accessed: 2020-04-10. [Online]. Available: https://circuitdigest.com/microcontroller-projects/getting-started-with-esp8266-modul

[13] Cloudflare. What Is Latency? — How to Fix Latency. Date accessed: 2020-04-05. [Online]. Available: https://www.cloudflare.com/learning/performance/glossary/what-is-latency/

[14] ESP8266, Espressif. Date accesed: 2020-01-15. [Online]. Available: https://www.lawicel-shop.se/nodemcu-v3-with-esp-12e-ch340

[15] Solid Edge ST10, Siemens. [Software]. Available: https://solidedge.siemens.com/en/

[16] Material data for PMMA. Date accessed: 2020-03-20. [Online]. Available. http://www.matweb.com/search/datasheet.aspx?bassnum=O1303&ckck=1

[17] Arduino, Arduino Uno Rev3. Date accessed: 2020-01-15. [Online]. Available: https://store.arduino.cc/arduino-uno-rev3

[18] MPU-6050. Date accessed: 2020-01-15. [Online]. Available: https://www.electrokit.com/produkt/mpu-6050-accelerometer-3-axel-monterad-pa-kort/

[19] Japan servo KH56JM2-851. Date accessed: 2020-04-20. [Online]. Available: https://www.datasheetarchive.com/KH56JM2-851-datasheet.html

[20] Nema23 stepper motor. Date accessed: 2020-02-15. [Online]. Available: https://joy-it.net/en/products/NEMA23-02

[21] Turnigy 2200mAh 3S 40C Lipo Pack, Turnigy. Date accessed: 2020-02-10. [Online]. Available: https://hobbyking.com/en_us/turnigy-2200mah-3s-40c-lipo-pack.html?queryID=55d6b2436f898e0523467dbbbc686552&objectID=18304&indexName=hbk_live_magento_en_us_products

[22] Joystick till Ps2 handkontroll. Date accessed: 2020-02-20. [Online]. Available: https://pchbutik.se/kretskort/138-joystick-till-ps2-handkontroll-passar-arduino.html?search_query=joystick&results=5

[23] Brokking. Your Arduino Balancing Robot. Date accessed: 2020-01-20. [Online]. Available: http://www.brokking.net/yabr_main.html

[24] Lucidchart. [Software]. Available: https://www.lucidchart.com/

[25] Matlab, MathWorks. [Software]. Available: https://www.mathworks.com/products/matlab.html

[26] Photoshop, Adobe. [Software]. Available: https://www.adobe.com/products/photoshop.html

[27] Dan Nguyen and Kayan Phuong, ”SB-bot”, KTH, Tech Rep., 2016. Date accessed: 2020-01-25. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:957115/FULLTEXT01.pdf

[28] Bluair. Bluetooth Range: 100m, 1km or 10km? Date accessed: 2020-05-08. [Online]. Available: http://www.bluair.pl/bluetooth-range

[29] Telia. Täckningskartor. Date accessed: 2020-05-08. [Online]. Available: https://www.telia.se/privat/support/tackningskartor

[30] Fritzing, Interaction Design Lab Potsdam. [Software]. Available: https://fritzing.org/

[31] Hanna Hellman and Henrik Sunnerman, ”Two-Wheeled Self-Balancing Robot”, KTH, Tech Rep., 2015. Date accessed: 2020-01-25. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:916184/FULLTEXT01.pdf

Appendix A

The mechanical model

The system was modelled as an inverted pendulum attached to a cart in order to simplify the equations. The calculations were inspired from Hellman and Sunnerman.[31].

A.1 Wheels

The following equation was derived from the forces acting on the cart in the horizontal plane.

$$\rightarrow: \ F_{input} = m_{cart}\ddot{x} + f\dot{x} + N_x \tag{A.1}$$

Where m_{cart} is the mass of the cart, $\ddot{x}$ is the acceleration in the x-axis, f is the coefficient of friction, $\dot{x}$ is the speed in the x-axis and N_x is the reaction force in the x-axis.

A.2 Pendulum

The following equations for the pendulum were derived from the model:

The forces perpendicular to the pendulum:

$$N_y \sin\theta + N_x \cos\theta - m_{pend} g \sin\theta = m_{pend} l\ddot{\theta} + m_{pend}\ddot{x}\cos\theta \tag{A.2}$$

Where m_{pend} is the mass of the pendulum, l is the distance from origo to the center of mass, θ is the angle from the the vertical axis and the pendulum, N_x is the reaction force in the x-axis and N_y is the reaction force in the y-axis.

The forces along the x-axis:

$$\rightarrow: \ N_x = m_{pend}\ddot{x} + m_{pend} l\ddot{\theta}\cos\theta - m_{pend} l\dot{\theta}^2 \sin\theta \tag{A.3}$$

The torque acting on the center of mass of the pendulum:

$$\overset{\frown}{A}: \ I_{pend}\ddot{\theta} = -N_x l\cos\theta - N_y l\sin\theta \tag{A.4}$$

Where I_{pend} is the moment of inertia of the pendulum.

A.3 Combining the equations

Rearranging equation (A.1) to the expression of N_x and inserting it in equation (A.3) yielded:

$$F_{input} = (m_{cart} + m_{pend})\ddot{x} + f\dot{x} + m_{pend}l\ddot{\theta}\cos\theta - m_{pend}l\dot{\theta}^2 \sin theta \tag{A.5}$$

Then equation (A.2) and equation (A.3) was combined yielding:

$$(I_{pend} + m_{pend}l^2)\ddot{\theta} + m_{pend}gl\sin\theta = -m_{pend}l\ddot{x}\cos\theta \tag{A.6}$$

Appendix B

Linearization

In order to apply linear control theory equations (A.5) and (A.6) needed to be linearized. It was assumed that the angle deviation from the upright equilibrium point would be small, the non-linear factors could then be linearized as:

$$\cos\theta = \cos\pi + \phi \approx -1 \tag{B.1}$$

$$\sin\theta = \sin\pi + \phi \approx -\phi \tag{B.2}$$

$$\dot{\theta}^2 = \dot{\phi}^2 \approx 0 \tag{B.3}$$

By inserting the approximations into equation (A.5) and (A.6) two linearized equations were written:

$$(I_{pend} + m_{pend}l^2)\ddot{\phi} - m_{pend}gl\ddot{\phi} = m_{pend}\ddot{x} \tag{B.4}$$

$$u = (m_{cart} + m_{pend})\ddot{x} + f\dot{x} - m_{pend}\ddot{\phi} \tag{B.5}$$

Since the force F_{input} is unknown it was substituted with a general control effort, u.

Appendix C

Laplace transform

Equations (B.4) and (B.5) was transformed in to the Laplace domain in order to obtain the transfer functions.

$$(I_{pend} + m_{pend}l^2)\phi(s)s^2 - m_{pend}gl\phi(s) = m_{pend}lX(s)s^2 \tag{C.1}$$

$$U(s) = (m_{cart} + m_{pend})X(s)s^2 + fX(s)s - m_{pend}\phi(s)s^2 \tag{C.2}$$

Then $X(s)$ was solved from (C.1), since a transfer function is the relationship between the input and the output. Solving for $X(s)$ yielded:

$$X(s) = (\frac{I_{pend} + m_{pend}l^2}{m_{pend}l} - \frac{g}{s^2})\phi(s) \tag{C.3}$$

Equation (C.3) was then substituted into (C.2) yielding:

$$U(s) = (m_{cart}+m_{pend})(\frac{I_{pend} + m_{pend}l^2}{m_{pend}l} - \frac{g}{s^2})\phi(s)s^2 + f(\frac{I_{pend} + m_{pend}l^2}{m_{pend}l} - \frac{g}{s^2})\phi(s)s - m_{pend}\phi(s)s^2 \tag{C.4}$$

The transfer function, $G_\phi(s)$, between $U(s)$ and $\phi(s)$ was obtained by rearranging equation (C.4):

$$G_\phi(s) = \frac{\frac{m_{pend}ls}{q}}{s^3 + \frac{f(I_{pend}+m_{pend}l^2)}{q}s^2 - \frac{(m_{cart}+m_{pend})m_{pend}gl}{q}s - \frac{fm_{pend}gl}{q}s} \tag{C.5}$$

where

$$q = (m_{cart} + m_{pend})(I_{pend} + m_{pend}l^2) - (m_{pend}l)^2 \tag{C.6}$$

The transfer function, $G_X(s)$, between $U(s)$ and $X(s)$ was obtained by substituting equation (C.6) into equation (C.3) yielding:

$$G_X(s) = \frac{\frac{(I_{pend}+m_{pend}l^2)s^2 - gm_{pend}l}{q}}{s^4 + \frac{f(I_{pend}+m_{pend}l^2)}{q}s^3 - \frac{(m_{cart}+m_{pend})m_{pend}gl}{q}s^2 - \frac{fm_{pend}gl}{q}s} \tag{C.7}$$

Appendix D

Moment of Inertia and center of mass

The moment of inertia I can be calculated using formula D.1 where i is the number of bodies, r_i the radius to the movement and m_i the mass of the body.

$$I = \sum_i r_i^2 m_i \tag{D.1}$$

The center of mass y_c can be calculated using formula D.1 where i is the number of bodies, r_i is distance, m_i is mass and M is the total mass.

$$y_c = \frac{1}{M} \sum_i r_i m_i \tag{D.2}$$

The CAD program Solid Edge was used to calculate I_y and y_c.

The body: $I_{y,b} = 111mm^4$, $y_{c,b} = 0.1m$

The wheel: $I_{y,w} = 111mm^4$, $y_{c,w} = 0$

Appendix E

Weight off center model

The system was modelled as an pendulum and a weight attached to the pendulum to simplify the equations.

E.1 Pendulum

The length along the horizontal axis from the origin to the center of mass for the pendulum was derived:

$$x_{cg_{pend}} = l \sin \phi \tag{E.1}$$

Then the torque acting on the origin due to the pendulums mass was derived:

$$T_{pend} = m_{pend} g l \sin \phi \tag{E.2}$$

E.2 Weight

The length along the horizontal axis from the origin to the center of the weight was derived:

$$x_{cg_{weight}} = l_w \sin \phi - l_{w_c} \cos \phi \tag{E.3}$$

Then the torque acting on the origin due to the weights mass was derived:

$$T_w = m_w g(l_w \sin \phi - l_{w_c} \cos \phi) \tag{E.4}$$

E.3 Combining the equations

E.2 and E.4 was then added since there is equilibrium of torque at the balance point:

$$0 = m_{pend} g l \sin \phi + m_w g(l_w \sin \phi - l_{w_c} \cos \phi) \tag{E.5}$$

E.5 was then solved for the angle ϕ:

$$\phi = \arctan \left(\frac{m_w l_{w_c}}{m_{pend} l + m_w l_w}\right) \tag{E.6}$$

Appendix F

Component table

QTY	Component	Specification	Weight (g)	Price (SEK)
Hardware				
1	Arduino Uno	Rev3	40	228
2	Steppermotors	Indonesia	859	500
2	Motor drivers	DRV8825	3	158
1	Balance sensor	MPU-6050	6	49
1	ESP8266	ESP12-E	30	49
1	Battery	12V	185	200
Bought mechanical components				
4	Threaded rods	M6, length 250mm	165	27
24	Nuts	M6	49	24
24	Washers	M6	23	24
Manufactured mechanical components				
1	Counterweight	35x225x12mm	719	
1	Baseplate, bottom	81.5x170x10mm	53	
1	Baseplate, w/ cutout	81.5x170x10mm	47	
1	Component Baseplate	81.5x170x5mm	34	
1	Counterweight Baseplate	81.5x170x20mm	34	
1	Battery holder	28x112x62mm	39	
2	PMMA Wheel rims	⌀100x25mm	50	
2	Wheel holder	⌀10x20x20mm	4	
2	Wheel decks	⌀106x25mm	1	
		Total	**1 657**	**1 239**

Table F.1. List of all components, quantity, weight and price

Appendix G

Circuit diagram

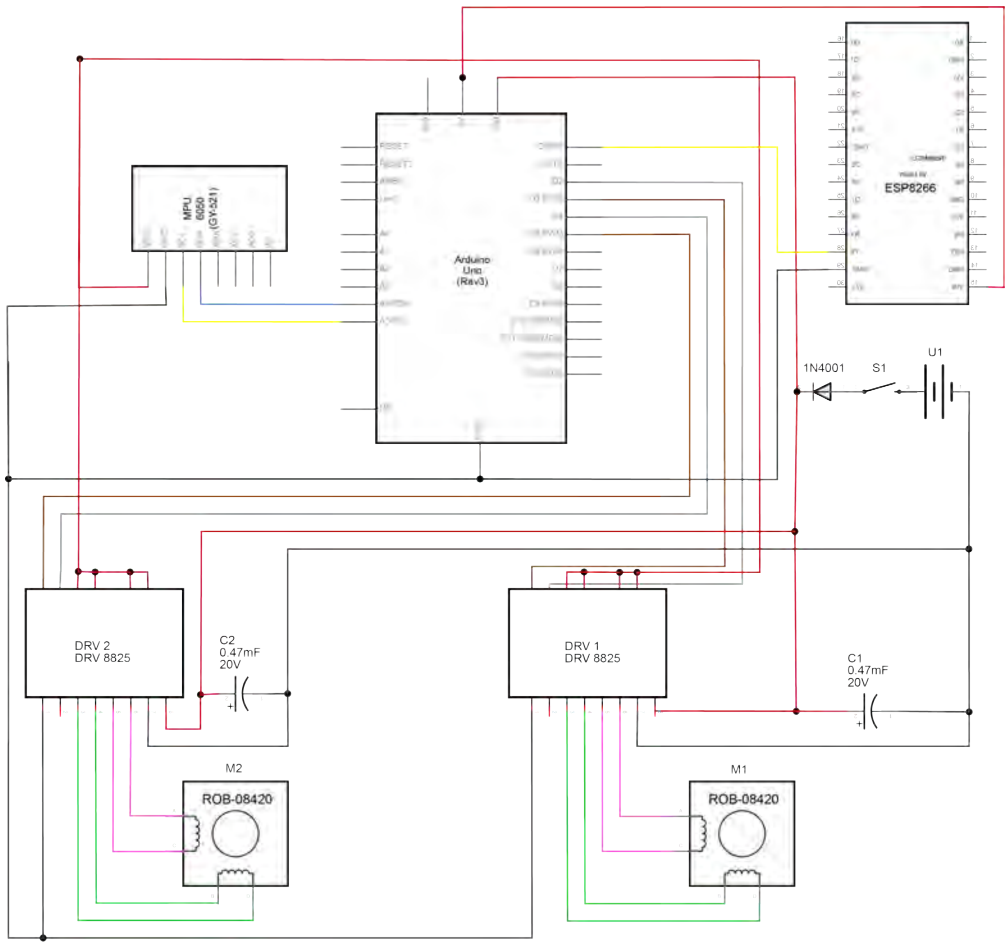


Figure G.1. Figure showing circuit diagram for the robot. Made in Fritzing[30].

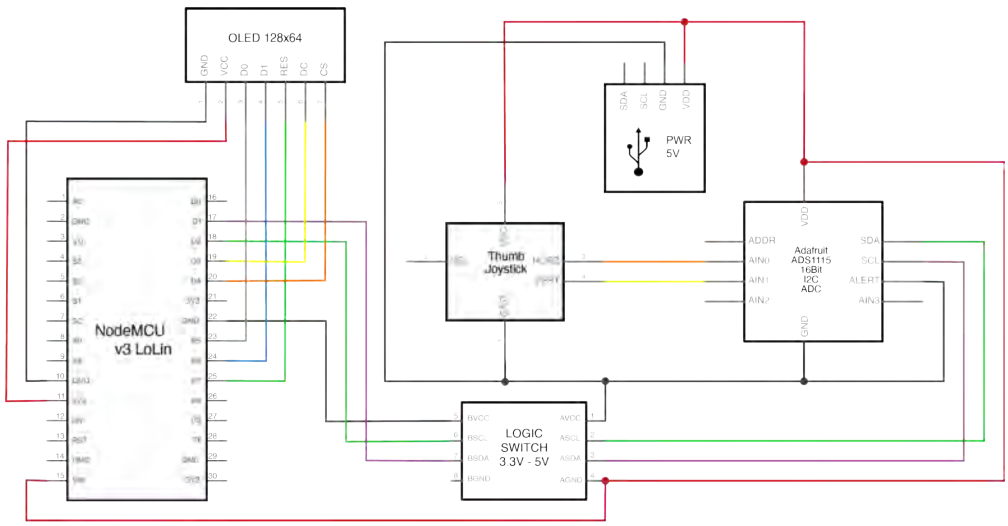


Figure G.2. Figure showing circuit diagram for the remote. Made in Fritzing.

Appendix H

Code

H.1 Robot code

```
/*
  Arduino Two Wheeled Self-balancing Robot

  The code:
    - Reads data from the MPU6050 and calculates the angle
    - Calculates the motor speed based on the angle error with a PID
        controller
    - Reads the serial port and changes the angular set point or
        rotational speed
    - Changes the angular set point if the robot is moving when no
        steering command is given

  The code runs at 250 Hz on a Arduino UNO R3 with a clock speed of
      16 MHz

  Created 2020 by:
  Fredrik Ihrfelt (ihrfelt@kth.se)
  William Marin (wmarin@kth.se)
 */

//--- INCLUDED LIBRARIES ---//
#include <Wire.h>

//--- DECLARING I2C ADDRESS OF MPU6050 ---//
int gyro_address = 0x68;
int acc_calibration_value = 480;

//--- SETTING PID PARAMETERS ---//
float pid_p_gain = 12;
float pid_i_gain = 0.5;
float pid_d_gain = 22;

//--- DECLARING GLOBAL VARIABLES ---//
byte start;

```

```
int left_motor, throttle_left_motor, throttle_counter_left_motor, throttle_left_motor_memory;
int right_motor, throttle_right_motor, throttle_counter_right_motor, throttle_right_motor_memory;
int gyro_pitch_data_raw, gyro_yaw_data_raw, accelerometer_data_raw;

long gyro_yaw_calibration_value, gyro_pitch_calibration_value;

unsigned long loop_timer;

//Variables for the angle and PID controller
float angle_gyro, angle_acc, angle, self_balance_pid_setpoint;
float pid_error_temp, pid_i_mem, pid_setpoint, gyro_input, pid_output, pid_last_d_error;
float pid_output_left, pid_output_right;

//Variables for steering
float spd = 1; //Stores the desired speed value
float rotation = 1;  //Stores the rotation value
float turning_speed = 30;
float max_target_speed = 200;
float desired_rotation;
float desired_speed;

//Declaring constants for serial port reading
const byte buffSize = 40;
char inputBuffer[buffSize];
const char startMarker = '<';
const char endMarker = '>';
byte bytesRecvd = 0;
boolean readInProgress = false;
boolean newDataFromEsp = false;

char messageFromEsp[buffSize] = {0};


void setup(){
  //Starting the serial port att 115200 kbps
  Serial.begin(115200);

  //Starting the I2C bus
  Wire.begin();

  //Setting the I2C clock speed to 400kHz
  TWBR = 12;


  //Creating variable pulse for stepper motor control, TIMER2_COMPA_vect
  TCCR2A = 0;
  TCCR2B = 0;
  TIMSK2 |= (1 << OCIE2A);
  TCCR2B |= (1 << CS21);
  OCR2A = 39;
```

```
  TCCR2A |= (1 << WGM21);

  //Starting the MPU6050
  Wire.beginTransmission(gyro_address);
  Wire.write(0x6B);
  Wire.write(0x00);
  Wire.endTransmission();

  //Setting the scale of the gyro to +/- 250 degrees per second
  Wire.beginTransmission(gyro_address);
  Wire.write(0x1B);
  Wire.write(0x00);
  Wire.endTransmission();

  //Setting the scale of the accelerometer to +/- 4g.
  Wire.beginTransmission(gyro_address);
  Wire.write(0x1C);
  Wire.write(0x08);
  Wire.endTransmission();

  //Setting a Low Pass Filter on MPU6050 to 43Hz
  Wire.beginTransmission(gyro_address);
  Wire.write(0x1A);
  Wire.write(0x03);
  Wire.endTransmission();

  //Defining outputs
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(13, OUTPUT);

  //Calibrating the MPU6050 by reading the gyro offset 500 times and
      calculating the mean value
  for(receive_counter = 0; receive_counter < 500; receive_counter++){
    Wire.beginTransmission(gyro_address);
    Wire.write(0x43);
    Wire.endTransmission();
    Wire.requestFrom(gyro_address, 4);
    gyro_yaw_calibration_value += Wire.read()<<8|Wire.read();
    gyro_pitch_calibration_value += Wire.read()<<8|Wire.read();
    delayMicroseconds(3700);
  }
  gyro_pitch_calibration_value /= 500;
  gyro_yaw_calibration_value /= 500;

  //Creating loop timer to achieve 250 Hz frequency
  loop_timer = micros() + 4000;

}

//--- MAIN LOOP ---//
void loop(){
```

```

  //Reading data from the serial port
  getDataFromEsp();

  //Calculate forvard speed from serial reading
  if (spd > 1) {
    desired_speed = max_target_speed*(spd - 1);
  }

  //Calculate backward speed from serial reading
  if (spd < 1) {
   desired_speed = max_target_speed*(1 - spd);
  }

  //Calculate clockwise rotation speed from serial reading
  if (rotation > 1) {
    desired_rotation = turning_speed*(rotation - 1);
  }

  //Calculate counter clockwise rotation speed from serial reading
  if (rotation < 1) {
   desired_rotation = turning_speed*(1 - rotation);
  }

  //--- READ DATA FROM ACCELEROMETER ---//
  Wire.beginTransmission(gyro_address);
  Wire.write(0x3F);
  Wire.endTransmission();
  Wire.requestFrom(gyro_address, 2);
  accelerometer_data_raw = Wire.read()<<8|Wire.read();
  accelerometer_data_raw += acc_calibration_value;
  if(accelerometer_data_raw > 8200)accelerometer_data_raw = 8200;
  if(accelerometer_data_raw < -8200)accelerometer_data_raw = -8200;

  //Calculate angle from the accelerometer data
  angle_acc = asin((float)accelerometer_data_raw/8200.0)* 57.296;

  //Set the angle of the gyro to the angle of the accelerometer if
      the robot is vertical
  if(start == 0 && angle_acc > -0.5&& angle_acc < 0.5){
    angle_gyro = angle_acc;
    start = 1;
  }

  ///--- READING DATA FROM THE GYRO ---///
  Wire.beginTransmission(gyro_address);
  Wire.write(0x43);
  Wire.endTransmission();
  Wire.requestFrom(gyro_address, 4);
  gyro_yaw_data_raw = Wire.read()<<8|Wire.read();
  gyro_pitch_data_raw = Wire.read()<<8|Wire.read();

  //Compensate the angle data with the calibration value
  gyro_pitch_data_raw -= gyro_pitch_calibration_value;
```

```
  angle_gyro += gyro_pitch_data_raw * 0.000031;

  gyro_yaw_data_raw -= gyro_yaw_calibration_value;

  //Corecting gyro drift with complementary filter
  angle_gyro = angle_gyro * 0.9996 + angle_acc * 0.0004;

  //Print the angle of the gyro for experiments
  Serial.println(angle_gyro);

  //--- PID CONTROLLER ---//
  //Calculating the angular error
  pid_error_temp = angle_gyro - self_balance_pid_setpoint - pid_setpoint;
  if(pid_output > 10 || pid_output < -10)pid_error_temp += pid_output * 0.015 ;

  //Calculating the value on the I-part and add it to i_mem
  pid_i_mem += pid_i_gain * pid_error_temp;

  //Limit the maximum I-part value
  if(pid_i_mem > 400)pid_i_mem = 400;
  else if(pid_i_mem < -400)pid_i_mem = -400;

  //Calculating PID controller output
  pid_output = pid_p_gain * pid_error_temp + pid_i_mem + pid_d_gain * (pid_error_temp - pid_last_d_error);
  if(pid_output > 400)pid_output = 400;
  else if(pid_output < -400)pid_output = -400;

  //Storing the error for the next loop
  pid_last_d_error = pid_error_temp;

  //Creating a dead band for small PID outputs
  if(pid_output < 5 && pid_output > -5)pid_output = 0;

  //--- CONTROLLER OUTPUTS ---//

  //Copying the PID output to the right and left motors
  pid_output_left = pid_output;
  pid_output_right = pid_output;

  //If a right rotation command is given, increase speed on the left motor and decrease speed on the right motor
  if(rotation < 1){
    pid_output_left += desired_rotation;
    pid_output_right -= desired_rotation;
  }

  //If a left rotation command is given, increase speed on the right motor and decrease speed on the left motor
  if(rotation > 1){
    pid_output_left -= desired_rotation;
    pid_output_right += desired_rotation;
```

```
  }

  //If a forward command is given, increase the PID angle set point
  if(spd > 1){
    if(pid_setpoint > -2.5)pid_setpoint -= 0.1;
    if(pid_output > desired_speed * -1)pid_setpoint -= 0.005;
  }

  //If a backward command is given, decrease the PID angle set point
  if(spd < 1){
    if(pid_setpoint < 2.5)pid_setpoint += 0.1;
    if(pid_output < desired_speed)pid_setpoint += 0.005;
  }

  //If no steering command is given, set the PID angle set point to
      zero
  if(spd == 1 && rotation == 1){
    if(pid_setpoint > 0.5)pid_setpoint -=0.05;
    else if(pid_setpoint < -0.5)pid_setpoint +=0.05;
    else pid_setpoint = 0;
  }

  //Change the PID angle setpoint to compensate for off centered
      weight
  if(pid_setpoint == 0){
    if(pid_output < 0)self_balance_pid_setpoint += 0.0115;
    if(pid_output > 0)self_balance_pid_setpoint -= 0.0115;
  }


  //--- CALCULATING THE STEPPER MOTOR PULSE ---//
  //Linearize the stepper motors non-linear behavior
  if(pid_output_left > 0)pid_output_left = 405 - (1/(pid_output_left
      + 9)) * 5500;
  else if(pid_output_left < 0)pid_output_left = -405 - (1/(
      pid_output_left - 9)) * 5500;

  if(pid_output_right > 0)pid_output_right = 405 - (1/(
      pid_output_right + 9)) * 5500;
  else if(pid_output_right < 0)pid_output_right = -405 - (1/(
      pid_output_right - 9)) * 5500;

  //Calculate the pulse time for the stepper motors
  if(pid_output_left > 0)left_motor = 400 - pid_output_left;
  else if(pid_output_left < 0)left_motor = -400 - pid_output_left;
  else left_motor = 0;

  if(pid_output_right > 0)right_motor = 400 - pid_output_right;
  else if(pid_output_right < 0)right_motor = -400 - pid_output_right;
  else right_motor = 0;

  //Copy the pulse time to the throttle variables so the interrupt
      subroutine can use them
  throttle_left_motor = left_motor;
```

```
  throttle_right_motor = right_motor;


  //Delay loop if the time is under 40 ms (250 Hz)
  while(loop_timer > micros());
  loop_timer += 4000;
}

//--- INTERRUPT ROUTINE FOR TIMER2_COMPA_vect ---//
ISR(TIMER2_COMPA_vect){
  //Left motor pulse calculations
  throttle_counter_left_motor ++;
  if(throttle_counter_left_motor > throttle_left_motor_memory){
    throttle_counter_left_motor = 0;
    throttle_left_motor_memory = throttle_left_motor;
    if(throttle_left_motor_memory < 0){
      PORTD &= 0b11110111;
      throttle_left_motor_memory *= -1;
    }
    else PORTD |= 0b00001000;
  }
  else if(throttle_counter_left_motor == 1)PORTD |= 0b00000100;
  else if(throttle_counter_left_motor == 2)PORTD &= 0b11111011;

  //right motor pulse calculations
  throttle_counter_right_motor ++;
  if(throttle_counter_right_motor > throttle_right_motor_memory){
    throttle_counter_right_motor = 0;
    throttle_right_motor_memory = throttle_right_motor;
    if(throttle_right_motor_memory < 0){
      PORTD |= 0b00100000;
      throttle_right_motor_memory *= -1;
    }
    else PORTD &= 0b11011111;
  }
  else if(throttle_counter_right_motor == 1)PORTD |= 0b00010000;
  else if(throttle_counter_right_motor == 2)PORTD &= 0b11101111;
}


//--- FUNTION TO READ SERIAL DATA FROM ESP8266 ---//
void getDataFromEsp() {

  //Only read serial data if available
  if(Serial.available() > 0) {

    //Store read data in variable x
    char x = Serial.read();

    //Stop reading when '>' is sent
    if (x == endMarker) {
      readInProgress = false;
      newDataFromEsp = true;
      inputBuffer[bytesRecvd] = 0;
```

```
      parseData();
    }

    //Read the string between '<' and '>'
    if(readInProgress) {
      inputBuffer[bytesRecvd] = x;
      bytesRecvd ++;
      if (bytesRecvd == buffSize) {
        bytesRecvd = buffSize - 1;
      }
    }

    //Start reading when '<' is sent
    if (x == startMarker) {
      bytesRecvd = 0;
      readInProgress = true;
    }
  }
}

//--- FUNCTION TO SPLIT THE READ SERIAL DATA ---//
void parseData()

  //Using strtoIndx as an index
  char * strtokIndx;

  //Read the first part of the string and store as the speed command
  strtokIndx = strtok(inputBuffer, ",");
  spd = atof(strtokIndx);

  //Read the second part of the string and store as the rotation command
  strtokIndx = strtok(NULL, ",");
  rotation = atof(strtokIndx);

}
```

H.2 Remote code

H.2.1 Transmitter

```
/*
  Remote Transmitter Code

  The code:
  - Reads joystick position
  - Sends data to reciever ESP8266 with TCP
  - Register reciever ESP8266 response
  - Prints data to 0.96 oled display

```

```
  Created 2020 by:
  Fredrik Ihrfelt (ihrfelt@kth.se)
  William Marin (wmarin@kth.se)
*/
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <Wire.h>
#include <Adafruit_ADS1015.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <SPI.h>

const char *ssid = "robotwifi";
const char *password = "kthkex20";

int k = 0;
int j = 0;
int i = 0;
int q = 0;
int state = 1;
int displayTime;

float q2 = 0;
float a0;
float a1;
float a2;
float refreshTime;
float refreshMean = 0;
float refreshMeanDisp;
float refreshMax = 0;
float refreshMaxDisp = 0;
float minbat = 10.5;
float maxbat = 11.25;
float batper = 0;
float spd = 0;
float rot = 0;
float spdp = 0;
float rotp = 0;
float num = 0;
float voltage = 0;
float voltage2 = 0;
float RssI = 0;

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
// Declaration for SSD1306 display connected using software SPI
#define OLED_MOSI   12
#define OLED_CLK    14
#define OLED_DC     0
#define OLED_CS     2
#define OLED_RESET  13

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, OLED_MOSI,
                         OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);
```

```
Adafruit_ADS1115 ads(0x48);

void setup() {
  Serial.begin(115200);
  if (!display.begin(SSD1306_SWITCHCAPVCC)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }

  ads.begin();  // Starts ADC

  // Explicitly set the ESP8266 to be a WiFi-client
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  // Loops until connection with server is established
  while (WiFi.status() != WL_CONNECTED) {
    int16_t i;
    display.clearDisplay();
    display.setTextSize(1);                // Normal 1:1 pixel scale
    display.setTextColor(SSD1306_WHITE);   // Draw white text
    display.setCursor(10, 10);             // Start at top-left corner
    display.print(F("Connecting"));        // Prints status to screen
    display.display();
    delay(600);
    display.print(F("."));
    display.display();
    delay(300);
    display.print(F("."));
    display.display();
    delay(300);
  }
}

void loop() {
  int16_t adc0;  // Sixteen bit integer as a result
  int16_t adc1;
  int16_t adc2;
  adc0 = ads.readADC_SingleEnded(0);   // Reads data from ADC
  adc1 = ads.readADC_SingleEnded(1);
  adc2 = ads.readADC_SingleEnded(2);
  a0 = adc0; // Converts from int to float
  a1 = adc1;
  a2 = adc2;
  spd = 200 - round(200 * a1 / a2); // Scale data from sixteen bit to
       0-200 resulotion
  rot = round(200 * a0 / a2);

  // Makes sure that value 100 is sent when joystick idle
  if (spd <= 110) {
    if (spd >= 90) {
      spd = 100;
    }
  }
```

```
  if (rot <= 110) {
    if (rot >= 90) {
      rot = 100;
    }
  }

  // Initize wifi characteristics
  WiFiClient client;
  const char * host = "192.168.4.1";
  const int httpPort = 80;

  //In case of connection disruption: Prints status and resets the ESP
  if (!client.connect(host, httpPort)) {
    display.clearDisplay();
    display.setCursor(10, 10);
    display.println(F("connection failed,"));
    display.setCursor(10, 20);
    display.println(F("resetting"));
    display.display();
    delay(4000);
    ESP.reset() ;
    return;
  }

  // Creates a URI for the request
  String url = "/data/";
  url += "?sensor_reading=";
  url += spd;
  url += "/";
  url += "&sensor_reading2=";
  url += rot;
  url += "/";
  url += "&sensor_reading3=";
  url += state;

  refreshTime = millis(); // Saves the current millis() value

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  unsigned long timeout = millis();

  // Loops until connection is established
  while (!client.available()) {
    // If there is no contact with reciever after 5000ms or more, connection is lost. Prints status to screen.
    if (millis() - timeout > 5000) {
      display.clearDisplay();
      display.print(F("Connection timeout"));
      display.display();
      client.stop();
      return;
```

```
    }
  }

  refreshTime = millis() - refreshTime; // Saves the time for contact
      with reciever
  String line = client.readStringUntil('\r'); // Reads first line of
     server response

  // removes 9 first characters of string
  while (k <= 8) {
    line[k] = 0;
    k = k + 1;
  }
  k = 0;

  // Moves every character 9 positions forward
  while (j <= 20) {
    line[j] = line[j + 9];
    j = j + 1;
  }
  j = 0;

  q = line.toInt(); // Converts string to int

  // Sets RssI status
  if ( state == 1 ) {
    RssI = q;

  }
  // Sets Vin for battery
  if (state == 2) {
    voltage = q / 10;
    state = 0;
  }

  spd = spd - 100; // Scales to  -100 to 100

  display.clearDisplay();                 // Clears Display
  display.setTextSize(1);                 // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);    // Draw white text
  display.setCursor(0, 0);                // Start at top-left corner
  display.print(F("SPD "));
  display.print(spd, 0);
  display.print(F("% "));
  display.setCursor(64, 0);
  display.print(F("ROT "));

  // Prints R if joystick x-axis position is Right
  if (rot > 100) {
    rot = rot - 100;
    display.print(F("R "));
    display.print(rot, 0);
    display.println(F("% "));
  }
```

```

  // Prints L if joystick x-axis position is Right
  else if (rot < 100) {
    rot = 100 - rot;
    display.print(F("L "));
    display.print(rot, 0);
    display.println(F("%"));
  }

  // Prints Zero if joystick x-axis is 0
  else {
    display.print("   0");
    display.println(F("%"));
  }

  refreshMean = refreshMean + refreshTime;  // Creates a 20-loop
      refresh mean
  // Saves the maximum response time
  if (refreshTime >= refreshMax) {
    refreshMax = refreshTime;
  }

  // Saves refresh times to display float
  if (i == 20) {
    refreshMeanDisp = refreshMean / i;
    refreshMean = 0;
    refreshMaxDisp = refreshMax;
    refreshMax = 0;
    i = 0;
  }
  i = i + 1;

  // Prints response time
  display.setCursor(0, 10);
  display.print(F("refreshTime "));
  display.print(refreshMeanDisp, 0);
  display.println(F("ms"));
  display.setCursor(0, 20);
  display.print(F("refreshMax "));
  display.print(refreshMaxDisp, 0);
  display.println(F("ms"));


  // Scales and print robot battery voltage
  voltage = voltage / 10000;
  display.setCursor(0, 30);
  display.print(F("Vin "));
  display.print(voltage, 2);
  display.println(F("V"));

  //Calculates and prints battery percentage for robot
  batper = (voltage2 - minbat) / (maxbat - minbat);
  display.setCursor(64, 30);
  display.print(F("Bat "));
```

```
  display.print(batper, 0);
  display.println(F("%"));
  display.display();  // prints the display
}
```

H.2.2 Reciever

```
/*
  Remote Transmitter Code

  The code:
  - Reads request from client
  - Processes request and sends to Arduino
  - Measure battery voltage
  - Sends battery voltage to remote control

  Created 2020 by:
  Fredrik Ihrfelt (ihrfelt@kth.se)
  William Marin (wmarin@kth.se)
*/
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

const char *ssid = "robotwifi";
const char *password = "kthkex20";
const int analogInPin = A0;  // ESP8266 Analog Pin ADC0 = A0

int sensorValue = 0;  // value read from the pot
int sendToEsp = 0;
float voltage = 12;
float spdA = 1;
float rotA = 1;

ESP8266WebServer server(80); //Opens the 80 port

// This handle is triggerd every 10ms
void handleSentVar () {
  if (server.hasArg ("sensor_reading")) {// this is the variable sent
      from the client

    float spd = server.arg ("sensor_reading"). toInt ();
    float rot = server.arg ("sensor_reading2"). toInt ();
    float state = server.arg ("sensor_reading3"). toInt ();

    spd = spd / 100;
    rot = rot / 100;


   // Sends data to Arduino if diffrent from previous or start value
    if (spd >= 0) {
```

```
      if (spd <= 2) {
        if (rot >= 0) {
          if (rot <= 2) {
            if (spd != spdA) {
              Serial.print("<");
              Serial.print(spd);
              Serial.print(",");
              Serial.print(rot);
              Serial.println(">");
            }

            else if (rot != rotA) {
              Serial.print("<");
              Serial.print(spd);
              Serial.print(",");
              Serial.print(rot);
              Serial.println(">");
            }

            spdA = spd;
            rotA = rot;

            //Reads battery voltage, scale and convert to int
            sensorValue = analogRead(analogInPin);
            voltage = (sensorValue * 1800000) / (1023);
            round(voltage);
            sendToEsp = voltage;

            server.send (sendToEsp, "", ""); // Sends data to transmitter
          }
        }
      }
    }
  }
}



void setup() {
  Serial.begin(115200);
  delay(50);
  WiFi.softAP(ssid, password);
  IPAddress myIP = WiFi.softAPIP();
  server.on("/data/", HTTP_GET, handleSentVar);
  server.begin();
}

void loop() {
  server.handleClient();
  delay (10);
}
```