Self-Balancing Robot Control System in CODESYS for Raspberry Pi

Design and Construction of a Self-Balancing Robot using PLC-programming tools

Emil Eriksson

Emil Eriksson
Umeå University
2016
Bachelor thesis project, 15 ECTS
Bachelor of Science Program in Mechanical engineering, 180 ECTS

Emil Eriksson
Umeå University
2016
Bachelor thesis project, 15 ECTS
Bachelor of Science Program in Mechanical engineering, 180 ECTS

Foreword

This project serves as my bachelor thesis project in mechanical engineering at Umeå University. The project was carried out at, and founded by The Department of Applied Physics and Electronics at Umeå University.
I would like to thank Fredrik Holmgren at the University's workshop for assisting me in laser cutting during the manufacturing process, Henrik Jonsson and Daniel Mårtensson for reviewing this report.
Thanks also goes out to Albin Flyckt, for reflecting ideas.

Finally, I would like to thank Sven Rönnbäck who was my supervisor during this project for assisting me during the whole process.

Emil Eriksson
Umeå 2017

Abstract

The Department of Applied Physics and Electronics at Umeå University offers education and conducts research in the field of automation and robotics. To raise the competence in automation in the CODESYS development environment it's proposed to build a remote controlled self-balancing robot as a testing platform which is then programmed using CODESYS for Raspberry Pi.

The chassis of the robot consists of laser-cut plexiglass plates, stacked on top of each other and fixed using threaded rods, nuts and washers. On these plates the robots' electrical components, wheels and motors are attached.

The control system is designed as a feedback loop where the robots' angle relative to the gravity vector is the controlled variable. A PID-controller is used as the system controller and a Kalman Filter is used to filter the input signals from the IMU board using input from both the accelerometer and the gyro.

The control system is implemented in CODESYS as a Function Block Diagram (FBD) using both pre-made, standard function blocks and customized function blocks. By using the in-built web-visualization tool the robot can be remote controlled via Wi-Fi.

After tuning the Kalman Filter through plot-analysis and the PID-controller through Ziegler-Nichols method the robot can stay balanced on a flat surface.

The robots' performance is tested through a series of test scenarios of which it only completes one out of four. The project ran out of time before further testing could be done.

For future work one could improve the performance of the PID-controller through more sophisticated tuning methods. One can also add a steering-function or test different type of controllers.

Table of contents

1 Introduction

1.1 Background

Umeå University is one of Sweden's large universities with almost 31000 students (postgraduate students included) and almost 4300 employees in 2015 [1]. The university offers a wide range of education and conducts research in areas from business, medicine, humanities, and natural science.

The Department of Applied Physics and Electronics offers courses, within the Bachelor and Master programs, in building technology, energy technology, mechanical engineering, electronics, and media technology [2]. The department conducts research in two main subjects; electronics and system engineering and energy technology.

Within this area of subjects the department conducts research in automation and robotics. For this purpose, it's proposed to build a platform for experiments using the industrial development environment CODESYS. CODESYS is widely used in the industry for programming Programmable Logic Controllers, or PLCs. A PLC is a type of computer commonly used for controlling industrial processes.

It's proposed that the platform is built in the form of a two-wheel self-balanced robot, which has become a classic platform for experiments in control theory because of its ability to display abstract control concepts such as robustness, system stability, controllability etc. [3].

1.2 Aim

The purpose of this project is to raise the competence in automation in the CODESYS development environment and its application in real-time, autonomous systems.

1.3 Goal

The goal of this project is to design a two-wheel balancing robot controlled via Wi-Fi. The software should be developed using CODESYS.

The project can be used as a reference in future implementation of CODESYS in automation education and research.

1.4 Conditions

The conditions and limitations of the project are as follows.

1.4.1 Financing/Budget

The project is financed by The Department of Applied Physics and Electronics at Umeå University.
The budget is set to 3000 SEK.

1.4.2 Timeframe

400 hours.

1.4.3 Material/Equipment
Material and equipment at students own expense:

- PC
- Writing material

Remaining material and equipment at The Department of Applied Physics and Electronics expense, for example:

- Software license, CODESYS for RPi
- Raspberry Pi unit
- I/O unit, for example a Arduino UNO
- Mechanics (wheels, mounting plates, nuts and screws, etc.)
- Electronics
- Soldering equipment
- Tools

1.4.4 Facilities
Facilities at the Department of Applied Physics and Electronics:

- Workshop
- Electrical engineering lab
- Computer lab
- The students home office

1.4.5 Specification
The robot specifications are listed in order of priority and should be met in that order.

1. The robot should be equipped with a Raspberry Pi with CODESYS (high priority)
2. The robot should have TWO wheels (high priority)
3. Controller parameters should be controlled via web-interface (high priority)
4. The robot should be equipped with a IMU breakout board
5. The robot should be able to keep its balance like an inverted pendulum
6. The robot should be able to avoid collision by using a distance sensor
7. The robot should be controlled using a smartphone (web-interface)
8. The robot should have on-board power supply in the form of batteries. The size of the batteries should be based on the power usage of the electronics.
9. Testing activities should be noted and filmed
10. Web-interface should present total travelled distance (low priority)

1.4.6 Test scenarios
To analyze the regulation-process, data should be retrieved during the following test scenarios.

1. The robot balances on a flat surface and a tilted surface (30°).
2. The robot is wireless controlled.
3. The robot brakes automatically for obstacles.
4. The robot brakes automatically and drives around obstacles.

2 Frame of reference

2.1 Self-balancing robot

The self-balancing robot has become a standard platform for testing the performance of a controlled system. It originates form the principle of the inverted pendulum (Figure 1) which is a classic problem in control theory.

The inverted pendulum is often modeled as a body is attached to a massless rod which, through a hinge joint, is attached to a moving cart on wheels. By responding to the accelerating body as gravity pulls it down with an opposite acceleration of the cart, the inverted pendulum can keep its upright position.

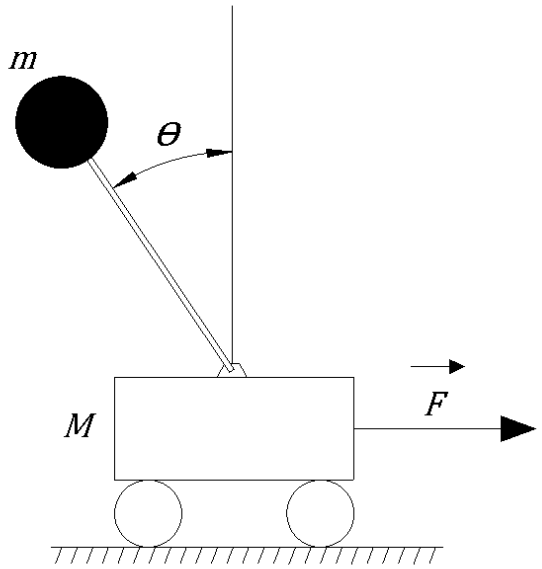


Figure 1: A schematic drawing of the inverted pendulum

For the balancing robot (Figure 2), the wheels represent the moving cart and the robots center of mass, wheels excluded, represent the body.

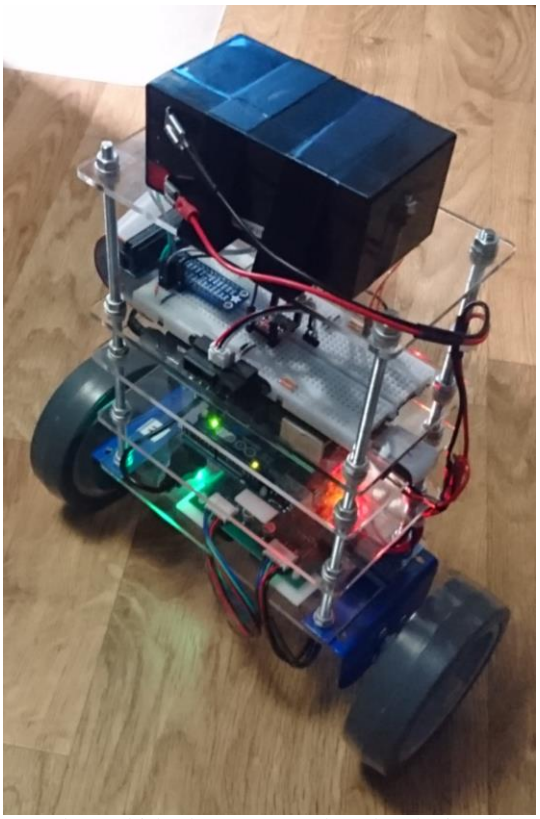


Figure 2: The self-balancing robot built for this project

The personal transporter vehicle Segway [4](Figure 3) is an example of an inverted pendulum control problem.

Figure 3: Segway [5].

2.2 The control system

By having the center of mass placed above the pivot point, the mechanical system of the self-balancing robot is inherently unstable and must have constant feedback from the control system to maintain its balance. This control process can be described by the feedback loop in Figure 4, terminology used in control theory is underlined.

1. The goal is to stay balanced meaning that the angle of the robots' position relative to the gravity vector is: 0° . This is called the target value of the controlled variable.

2. A gyro sensor attached to the robot measures the robots current angle. The actual value.

3. Included in the software is a comparator. It calculates the difference between the target value and the actual value. This is called the error and has the symbol letter *e*.

4. By using this value, one can calculate the control signal that the computer will send to the motors so that the robot will accelerate enough, but not too much, to return to its balanced position. This is done by the regulator.

5. This signal is then transformed into motion by the motors acting as the actuators in this example.

6. The process in this example is the balancing robot itself and since it's always affected by gravity, this is causing a disturbance of the robots' angle.

7. The new angle is measured by the gyro sensor, and so the loop start over.

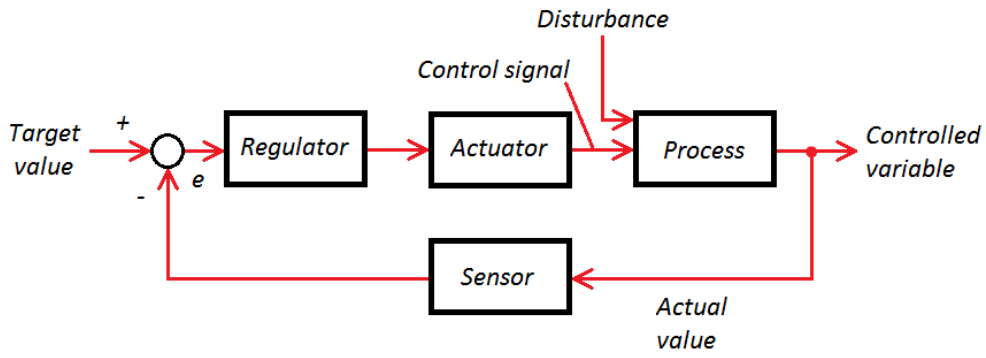


Figure 4: The basic principle of the general control loop.

2.3 Programmable Logic Controllers (PLC)

Using a PLC is the common way of controlling and monitoring an industrial process, such as the fluid level in a tank or the temperature inside a building.
For industrial applications, the PLCs are often mounted as modular racks in cabinets (Figure 5 and 6).

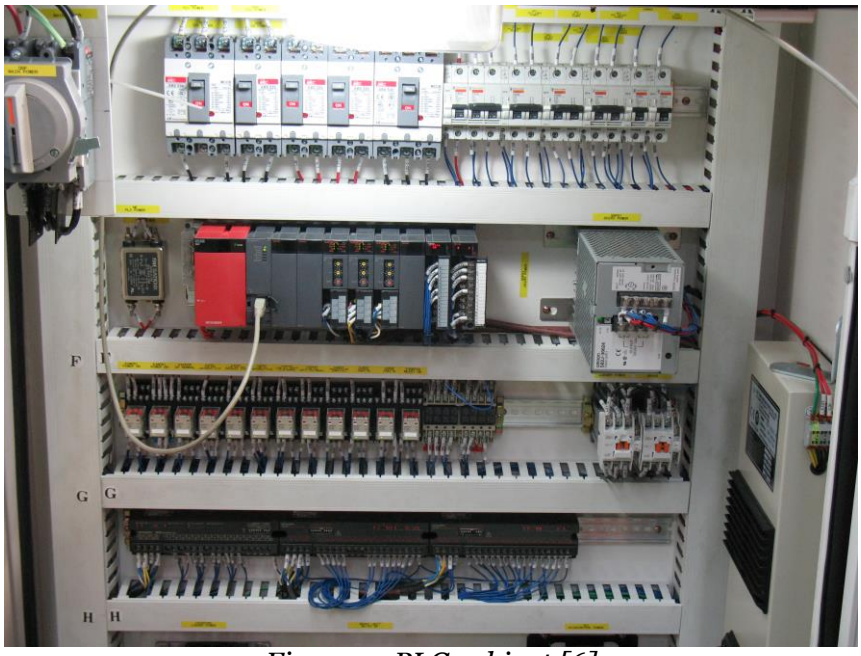


Figure 5: PLC cabinet [6]

An example of components in a PLC rack are:

- The controller (CPU, memory etc.)
- Power supply
- I/O-modules (analog input, digital input, digital output etc.)
- End modules that terminate the communication bus

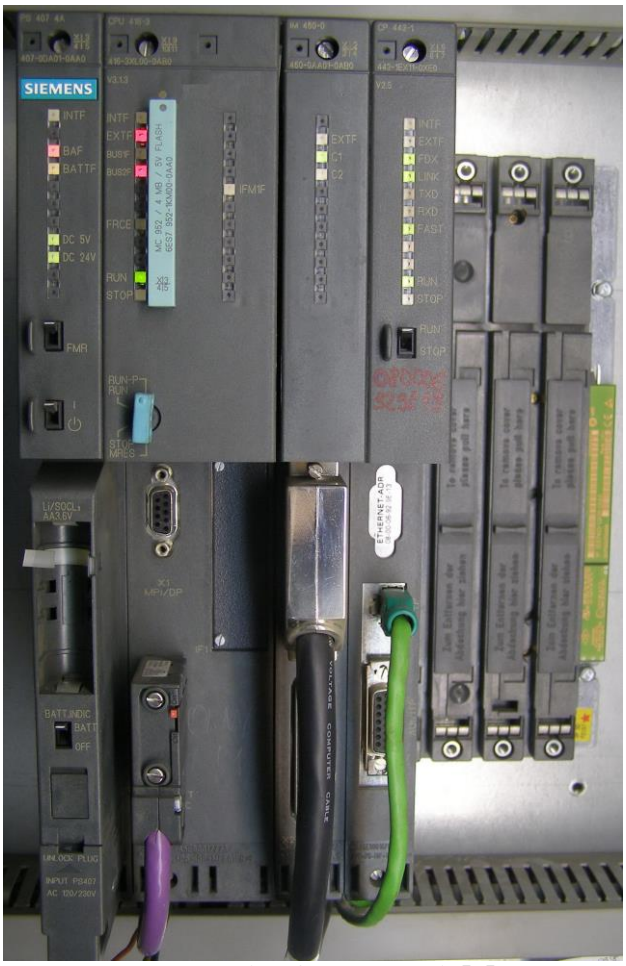


Figure 6: Example of a PLC rack [7].

A PLC works much like a computer and as such it must be programmed to be able to perform its desired task.

Since the PLC has it origin in relay-based industrial control systems, several of the programming languages used is of graphical character. For example, there is *Ladder Diagram* (LD) which originates from the relay wiring diagrams used by electricians or *Function Block Diagram* (FBD) with symbols and functions recognizable from digital technology [8].

2.4 Electrical system

The individual components of the electrical system are listed here along with a short description.

2.4.1 Raspberry Pi

To run CODESYS, the Raspberry Pi was chosen as the main controller. In this case the Raspberry Pi 1 model, as shown in Figure 7.
The Raspberry Pi from The Raspberry Pi Foundation is a credit card sized single-board computer, used primarily for computer education, home automation, robotics, and so on. The processing power is provided by the Broadcom BCM2835 SoC (system-on-a-chip), containing the 700 MHz single-core ARM1176JZF-S CPU (central processing unit) [9].

It has 26-pins including two output power pins of 3.3V and 5V and SDA and SCL for I^2C communication. 17 of these are GPIO-pins that can be used for digital input and output.

It's requires 5 V input power through the micro-USB port and can be powered through for example a USB-port or smartphone charger.

For this project the Raspberry Pi is loaded with the Debian/LINUX-based operating system Raspbian since it's recommended in the provided CODESYS documentation [10].

Figure 7: Raspberry Pi 1 model B

2.4.2 I/O unit (Arduino Uno)

Since the Raspberry Pi is limited to digital input and output and a lot of sensors use analog signals a I/O unit is used. An Arduino Uno (Figure 8) is programmed for this purpose.

The Arduino is a microcontroller board popular among hobbyists for experimental use.
It has 6 analog input pins and 14 digital I/O-pins of which 6 provide PWM output. There are also pins with a specialized function, for example for I^2C communication.
It can be powered through 12V and is on 5V logic.
The downloadable Arduino Software (IDE) is used to write the program and then load it to the Arduino by connecting the computer via USB. [11]

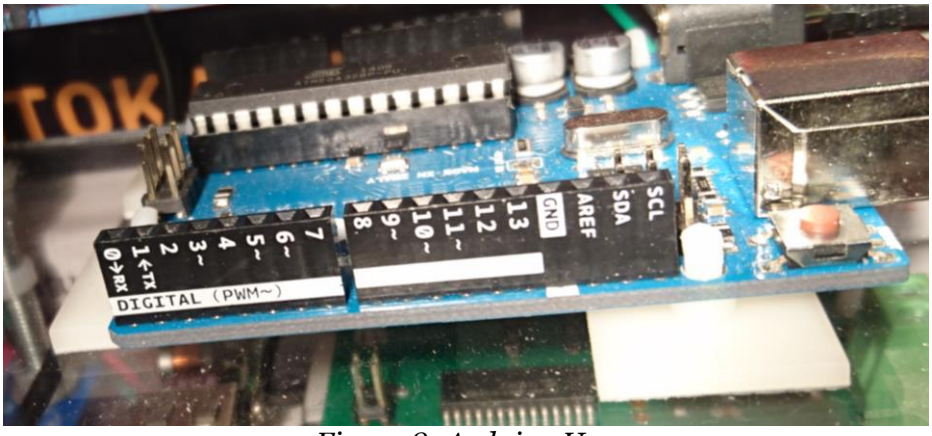


Figure 8: Arduino Uno

2.4.3 Motors

The motors chosen for this project are the EMG30 Gearmotor (Figure 9) with encoder from Robot Electronics since the motors were already in stock at The Department of Applied Physics and Electronics.

It is a 12V motor equipped with encoders and a 30:1 reduction gearbox.

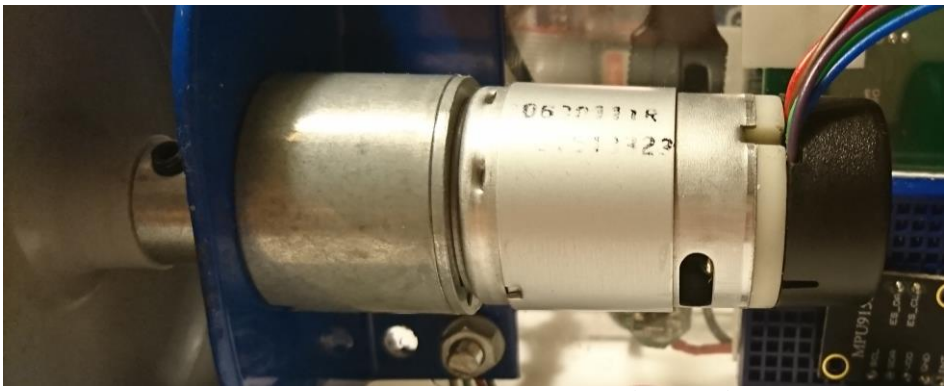


Figure 9: EMG30 Gearmotor with encoder

2.4.4 Motor Drive

As motor drive the MD25 Motor Drive from Robot Electronics (Figure 10) was chosen since it was already in stock at The Department of Applied Physics and Electronics.

This motor drive was designed to work with two EMG30 motors [12]. It can be controlled via I^2C as well as STTL serial on 5V logic.
The main features as described by the manufacturer are [12]:

- Reads and counts motors encoder pulses
- Drives two motors independently
- Reads motor current
- 12V power input
- Onboard 5V regulator
- Steering feature, a robot can be turned by sent value.
- Variable acceleration and power regulation.

One can choose whether to send speed or turn input as signed values ranging from -128 to 127 where 0 is stop, or as unsigned values ranging from 0 to 255 where 128 is stop.

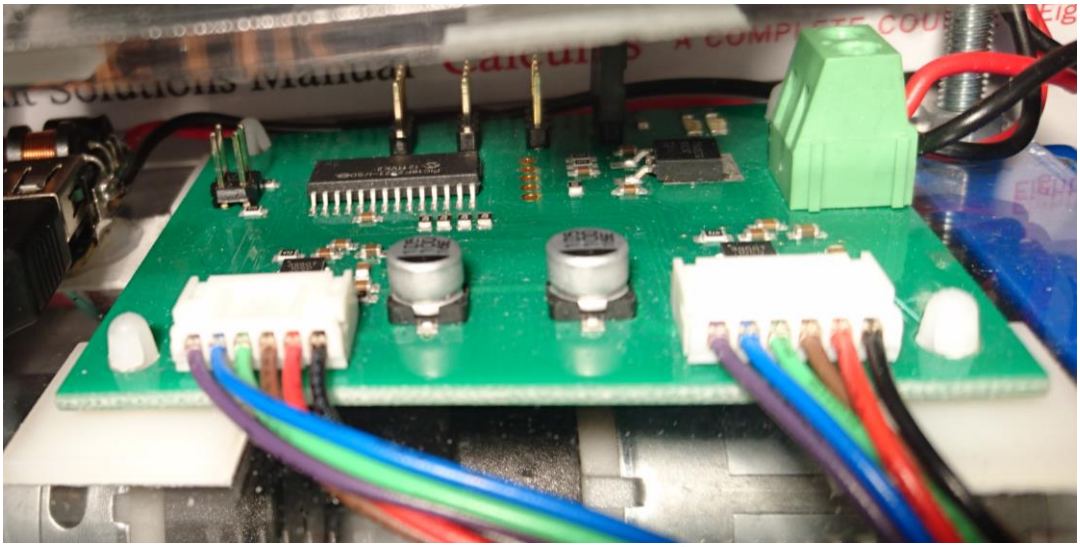


Figure 10: MD25 Motor drive

2.4.5 IMU breakout board

To measure the robots tilt and acceleration an IMU or *Internal Measurement Unit* is needed. The MPU9150 breakout board from Drotek (shown in Figure 11), with the MPU9150-chip [13] from InvenSense, is chosen since there is already a library and device description file included in the Raspberry Pi CODESYS-package.
It's a 9-DOF (degrees of freedom) IMU meaning it has a 3-axis gyroscope measuring angular velocity, a 3-axis accelerometer measuring acceleration and a 3-axis magnetometer that functions like a compass. It also has a temperature sensor [13].
Although the magnetometer won't be used in this project, it can still be interesting to have access to since this robot is meant to be used as a platform for future experiments.

It's powered by 5V or 3.3V and communicates via the I²C -bus.

Figure 11: MPU9150 IMU from Drotek

2.4.6 Logic Level converter (LLC)

Since the system communicates via I²C with different logic levels (3.3V and 5V) a converter is used to safely connect the components.
The LLC from Sparkfun (Figure 12) is chosen since it is reasonably priced and there is a detailed tutorial on how it works and how it's used at the Sparkfun website; https://learn.sparkfun.com/tutorials/using-the-logic-level-converter.

It has two bi-directional shifters marked TXO and TXI and since I²C communication works in both ways, this is where the SLA and SCL wires are connected to the LLC [14].

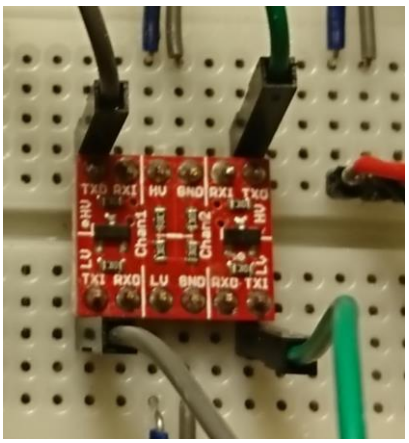


Figure 12: Logic Level Converter

2.4.7 Distance sensor

To detect obstacles and brake automatically the robot is equipped with a SHARP GP2Y0A21YK infrared distance sensor (Figure 13). It detects objects at a distance ranging from 10 to 80 cm and outputs a nonlinear analog signal between 0 to 3.3V [15].

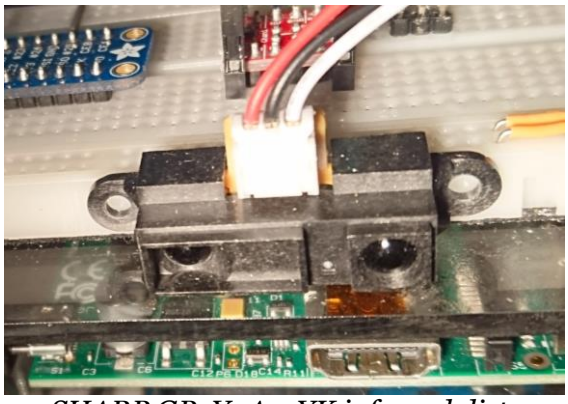


Figure 13: SHARP GP2Y0A21YK infrared distance sensor

2.4.8 Battery

As the power source a 12V, 1.3 Ah lead-acid battery is used, visible in Figure 14. The reason for this is that the motors require a 12V power source and the acid-lead battery is available at a low cost compared with more energy-dense alternatives such as NiMH-batteries. It can also be recharged with a conventional car-battery charger.

Figure 14: Battery

The battery life time is calculated according to table 1.

Table 1: The total power consumption of the electrical components

Device	**Current usage (mA)**
Raspberry Pi	1000
Arduino Uno	50
Motors (MD25)	2800 x 2 (max)
Sensors (approx.)	200
Total	**6850**

$$Runtime\ (h) = \frac{Battery\ capacity\ (Ah)}{Device\ concumption\ (A)} = \frac{1{,}3}{6{,}85} \approx 0{,}2\ h = 12\ min$$

A total run time of about 12 minutes is enough for most experiments and is comparable with many RC-devices, such as toy-drones. Probably the lifetime will be significantly longer since the motors ideally never will run on full speed when the robot is balancing.

2.4.9 12V to 5V converter

The Raspberry Pi requires a 5V input power supply. To achieve this from a 12V battery, a converter is needed. A USB-charging device for cars, shown in Figure 15, is disassembled and modified for this purpose.

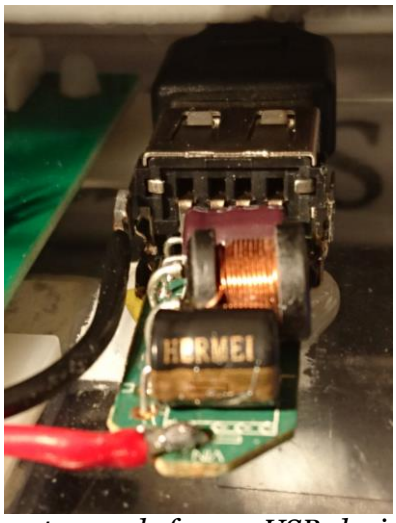


Figure 15: 12V to 5V converter made from a USB-device charger for use in cars

2.5 Software

2.5.1 CODESYS

The robot is programmed using the CODESYS [16] programming tool, version 3.5 by Smart Software Solutions GmbH. It's a free-of-charge, hardware-independent programming system for PLCs widely used in automation industry.

CODESYS follows the international standard IEC 61131-3 (International Electrotechnical Commission, 2013), which incorporates the programming languages used for PLC-programming. These are; *Ladder Diagram* (LD), *Instruction List* (IL), *Sequential Function Chart* (SFC), *Function Block Diagram* (FBD) and *Structured Text* (ST). ST is a high-level language associated with Pascal and C.

PLC-manufacturers have recently started to follow the standard IEC 61131-3 to a growing extent [8].

Included in CODESYS is also the IEC-specified functions and function blocks (FBs) as well as a series of libraries with ready-made functions for different applications.

CODESYS also comes with a graphic visualization tool and a simulator for testing programs without hardware [17].

2.5.2 CODESYS Control application for Raspberry Pi

To use a Raspberry Pi as a PLC, a CODESYS package is installed on the Raspberry Pi.
The package contains a CODESYS plugin to install and update the package on the Raspberry Pi as well as device description files for numerous Raspberry Pi compatible devices such as the Raspberry Pi Camera, Adafruit PWM and several types of gyros, including MPU9150. It also makes it possible to control these devices via I^2C, SPI or 1-wire.

There are also instructions how to create additional device description files [10].

2.6 Filter

2.6.1 The Discrete Kalman Filter

The Kalman-filter is an algorithm which by using estimations based on the previous states of a system, taking measured values, predicted error, process- and measurement noise into account, makes an estimated prediction about the future states of the system which tend to be more accurate than single measurements [18].

The filter is built as a feedback cycle with two groups of equations. One "Prediction"-group and one "Correction"-group as shown in Figure 16, and described with equations 1 to 8. To improve the robustness of the filter, a validation gate based on observations of the innovation is used. If the observations are inside of the Mahalanobis distance of the innovation, they are accepted. If they are not, they are thrown away.

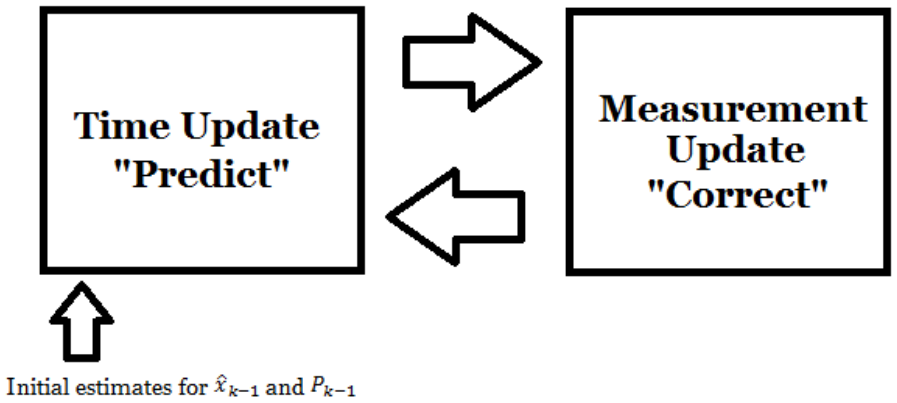


Figure 16: Kalman Filter feedback loop

In this application, the Kalman Filter estimates two states; the pitch-angle and the gyro bias based on the measurements from the MPU9150. This is a common method for filtering the signal from a IMU and was for example used by Christian Sundin and Filip Thorstensson in their master thesis on an autonomous balancing robot at Chalmers in 2012 [19]. It's also described on the blog TKJ Electronics [20].

After the initial estimates for $\hat{x}_{k-1}$ and P_{k-1} are set, the feedback cycle of the Kalman Filter performs as follows:

Time update ("Predict")

1) Project the state ahead

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{1}$$

2) Project the error covariance ahead

$$P_k^- = AP_{k-1}A^T + Q \tag{2}$$

Measurement update ("Correct")

1) Compute the Kalman gain, K_k

$$K_k = P_k^- H^T S_k^{-1} \tag{3}$$

where the prediction covariance S_k is calculated:

$$S_k = H P_k^- H^T + R \tag{4}$$

2) Update estimate with measurement z_k

$$\hat{x}_k = \hat{x}_k^- + K_k y_k \tag{5}$$

where the innovation y is calculated:

$$y_k = z_k - H\hat{x}_k^- \tag{6}$$

3) Calculate the Mahalanobis distance d_M based on the innovation y.
This is then used as a validation gate for the innovation. Observations outside d_M are thrown away and $K_k = 0$ if observation is not valid

$$d_M = \sqrt{y^T S^{-1} y} \tag{7}$$

4) Update the error covariance

$$P_k = (I - K_k H) P_k^- \tag{8}$$

2.7 Communication

2.7.1 I²C

The I²C-protocol is chosen as communication between the main CPU (Raspberry Pi), the gyro board, motor driver, and the I/O-unit (Arduino) as seen in Figure 17.

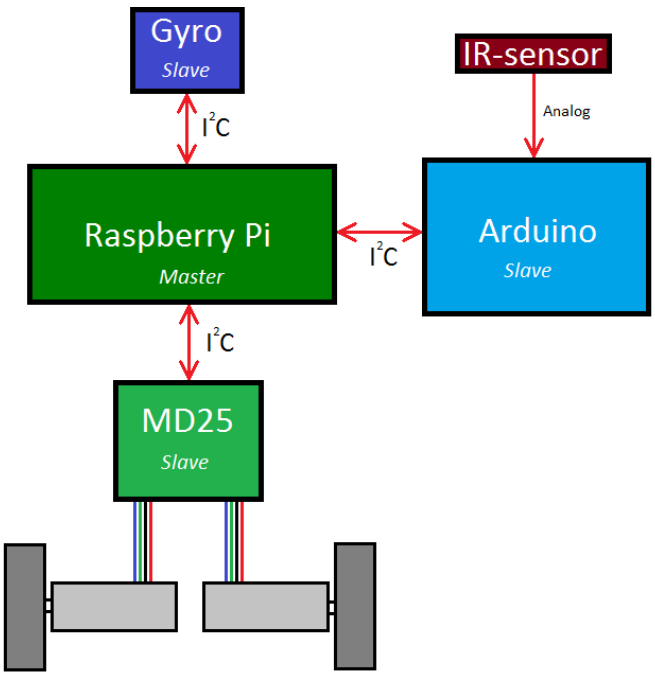


Figure 17: The communication network of the robot.

The communication network can consist of a (or even multiple) master device(s) and up to 1008 slave devices. It requires two-wires which represents two signals; SCL (the clock signal) and SDA (the data signal). The master device generates the clock signal.

I²C bus drivers can pull the signal low but not drive it high. A pull up resistor is placed on each signal line to restore the signal to high after being asserted to low.

A message is sent in two separate frames; first the address frame (7-bits) where the master calls for the slave to which the message is directed and then, one or more data frames (8-bits), containing the data [21]. Figure 18 shows the structure of these frames.

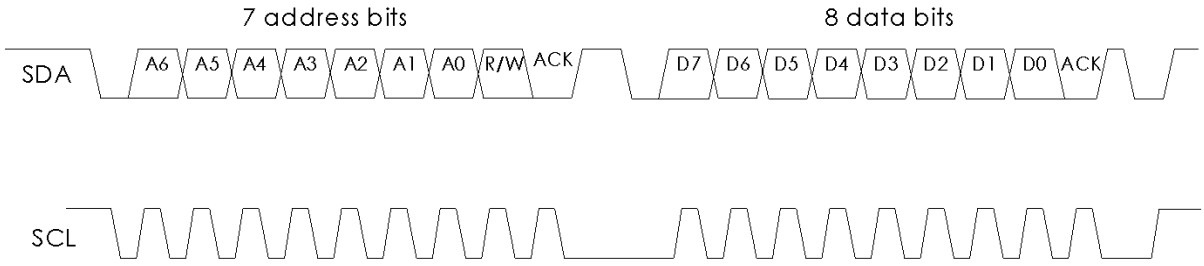


Figure 18: The I²C-message with the address- and data-frame

3 The process

A practical method is applied to the design of the robot. By using off-the-shelf parts and open-source hardware and software, information is often easily accessible online. One can find useful tips and tricks in the official online forums. One should bear in mind that the information provided is not necessarily accurate since these forums are open to anyone to contribute. CODESYS [22] and Raspberry Pi [23], as well as Arduino [24] all have well moderated forums.

3.1 Robot design

The following components are recommended use by The Department of Applied Physics and Electronics since they are already in stock at the faculty:

- MD25 - Dual 12Volt 2.8Amp H-Bridge Motor Drive
- 2x EMG30 – Gear Motor with encoder
- 2x EMG30 mounting bracket
- 2x Wheel 100mm

Using these parts as the framework for the design, a 3D model is created using the 3D-CAD software SolidWorks (Figure 19).

The components; gyro board, motor driver board, Raspberry Pi, Arduino etc. is attached to laser cut 3mm clear PMMA-plates, also known as Plexiglas. The plates are then placed as levels on a frame of threaded rods (M5), one in each corner and fixed by nuts, creating the chassis of the robot. The hole-pattern of the motor brackets is transferred to the PMMA-plates so that the motors could be attached directly to the main frame.
The 3D-model is used to determine the necessary space between the two motors and thereby the length of the PMMA-plate (the width of the robot).

The open solution makes the design flexible since one easily can access the individual components and use breadboard jumper wires for coupling. This is an asset for an experimental platform where you would like to be able to attach more sensors to conduct different kinds of experiments in the future. One of the levels consists of a stick-on breadboard, cut to fit the PMMA-plate.

It's an open design with all the parts accessible. It's also flexible with adjustable and replaceable levels. As an experimental platform both these characteristics are preferable since you can easily adjust the robot to the conditions of the experiment you want to perform.

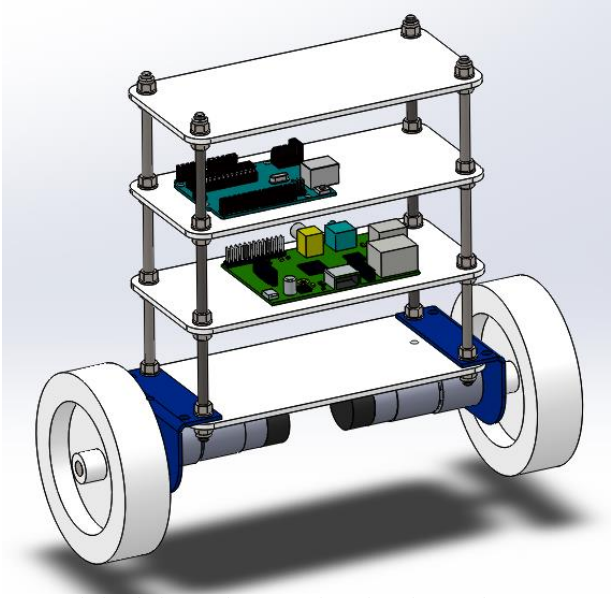


Figure 19: The first 3D-model of the robot with wheels and motors attached complete with an Arduino- and Raspberry Pi-model for scale reference.

3.2 Manufacture and assemble

The only parts that needs manufacturing is the PMMA-plates. PMMA is suitable for laser cutting which can be achieved with sufficient precision and surface finish.
Due to the simplicity of the design, instead of converting the SolidWorks-file, a sketch of the plate is redrawn in Adobe Illustrator in the computer already connected to the laser cutter and then loaded to the cutter. A 3mm PMMA-plate is placed in the laser cutter and the speed and depth settings are set and the plates are cut out as seen in Figure 20.

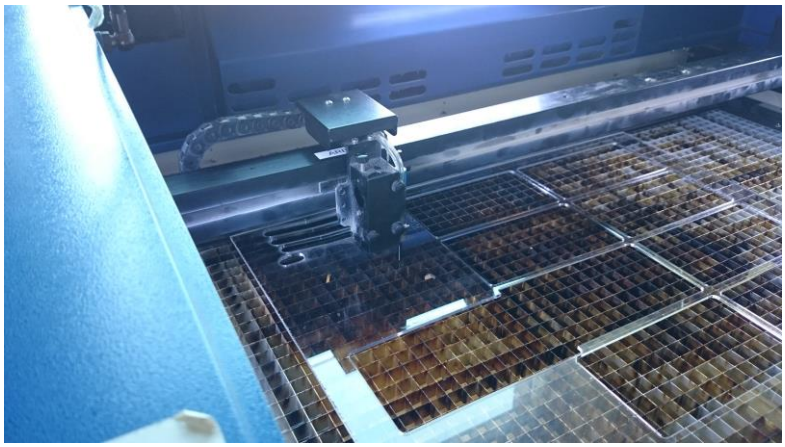


Figure 20 : Laser cutting 3mm PMMA-plates

The boards (Arduino, RPi, and MD25) do all have mounting holes and is attached to the PMMA-plates using stick-on spacers, some of which had to be modified (using a crafting knife) due to different hole sizes.
The PMMA plates and motors are then assembled using threaded rods, nuts and washers according to Figure 21, and Figure 22 below. The electronics is connected according to the circuit diagram in Figure 23.

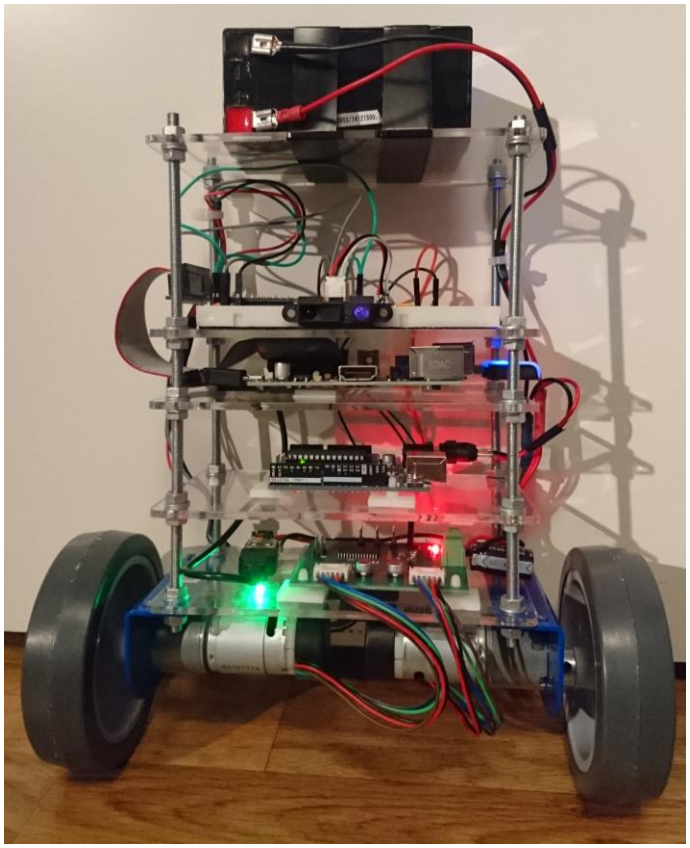


Figure 21: Front view of the robot showing from the top: battery, breadboard (level converter and distance sensor), Raspberry Pi, Arduino, MD25 and 12V-5V converter.

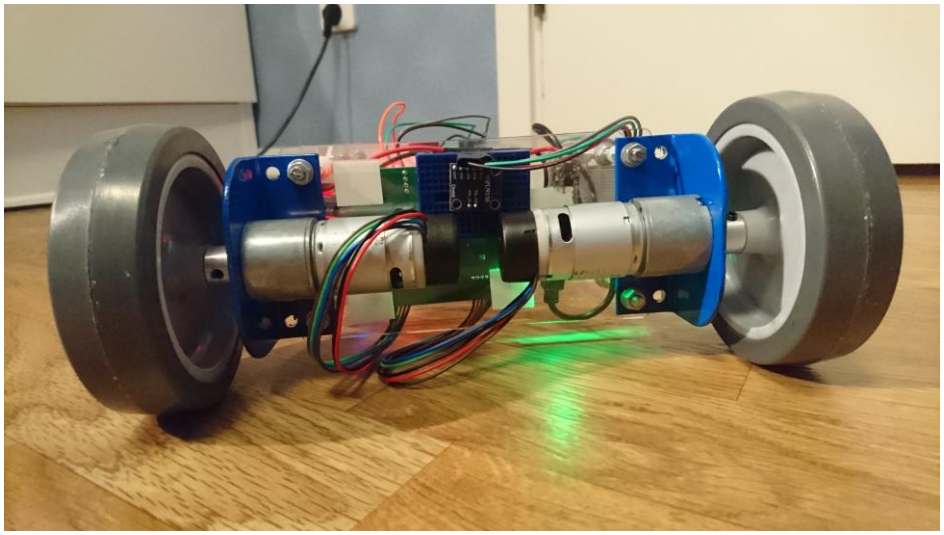


Figure 22: Under view of the robot showing the placement of the motors and IMU board.

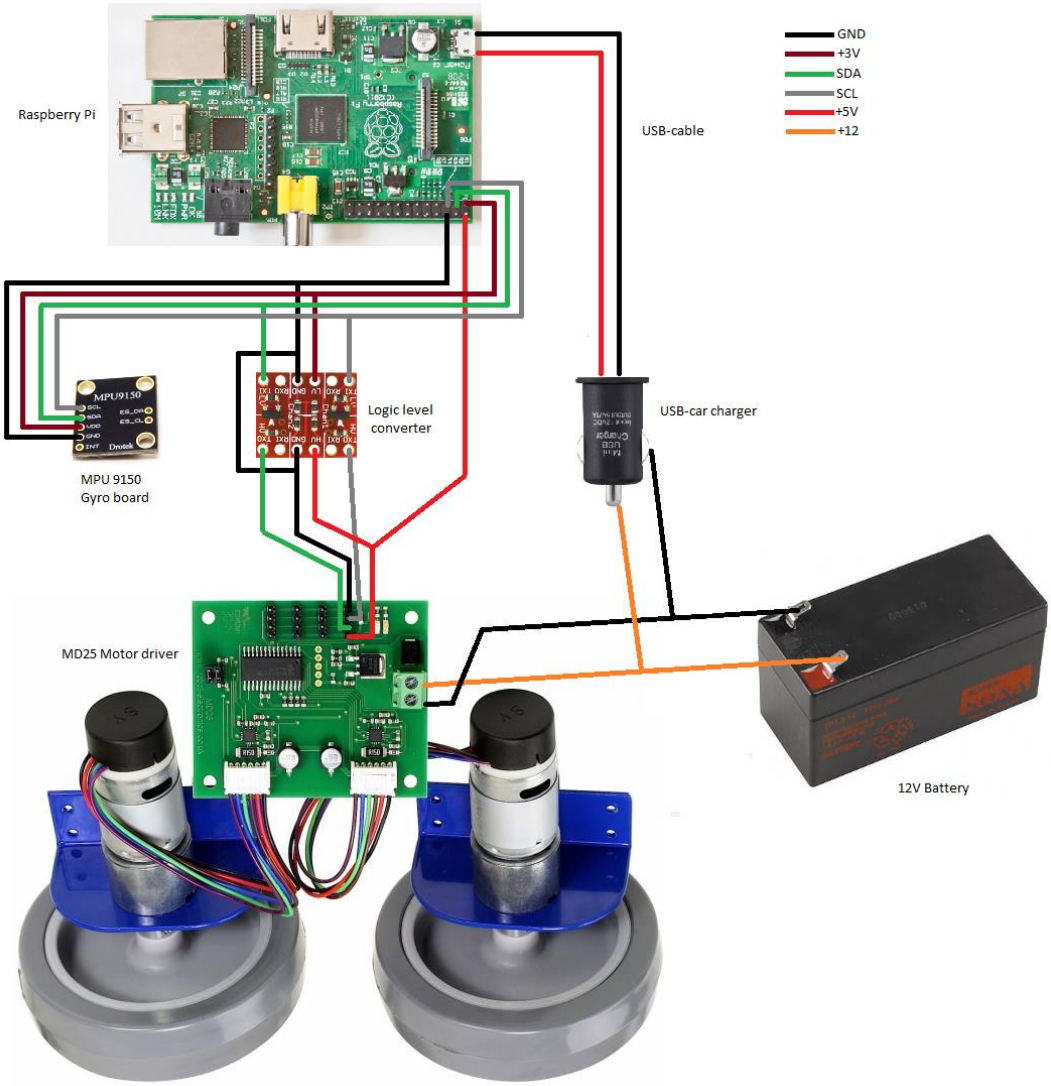


Figure 23: Circuit diagram

3.3 Startup and configuration of the Raspberry Pi

Following is the startup and configuration process for the Raspberry Pi. It is described in the documentation provided with CODESYS Control for Raspberry Pi SL [10], which is downloaded from the CODESYS Store [25].
A monitor, keyboard and mouse are plugged in to control the RPi after the OS installation.

3.3.1 Install the OS

The first thing to do with the Raspberry Pi (RPi) is to install the operating system (OS). The recommended operation system to use with CODESYS is Raspbian which can be installed either directly on the RPi or by using the installer NOOBS.
Using NOOBS, the OS is booted on the SD-card and installed to the RPi using the instructions on the Raspberry Pi Foundation website [26].

3.3.2 WLAN

The Raspberry Pi 3 model B has built-in Wi-Fi and is connected to the same Wi-Fi-network as the PC.

The IP-address of the RPi is shown by typing "hostname –I" in the terminal and press ENTER.

3.3.3 Enable I²C-communication

I²C-communication is not enabled by default on the RPi, therefore one must activate I²C manually. This is done in the configuration settings accessible by typing "sudo raspi-config" in the terminal.

3.4 Install CODESYS on the PC

To get first-hand experience of CODESYS, the program is installed on the PC according to the instructions in the provided documentation [16].
CODESYS is downloaded from the download-section on the CODESYS website [16].

3.4.1 Install CODESYS Control for Raspberry Pi package

In CODESYS, under the Tools menu; "Package manager" is selected. From here the package "CODESYS Control for Raspberry PI_3.5.8.0" is installed.

3.4.2 Install OSCAT Basic package

From the CODESYS Store, the OSCAT (Open Source Community for Automation Technology)-package is downloaded. It contains various mathematical functions from the open source community. The package is installed in CODESYS via the Package manager.

3.4.3 Install CODESYS to the RPi

The installation of CODESYS on to the RPi is done via Wi-Fi from CODESYS on the PC using the following steps [10].

a. CODESYS is opened on the PC and under the Tools menu; "Update RaspberryPi" is selected and the correct version is set by default.
b. Correct login data is entered; username: pi, password: raspberry, by default. The IP-address of the RPi is entered
c. "OK" is selected and a restart of CODESYS is performed.

3.5 Programming the control system

The control system design described in section 2.2 is used as the foundation when programming the robot. The updated control system design (Figure 24) shows how the components described above are used by the software.

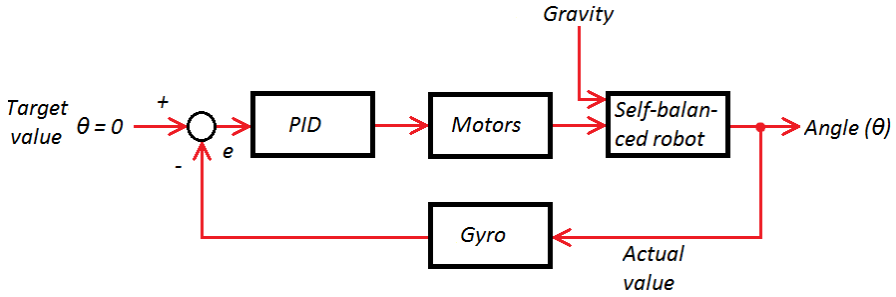


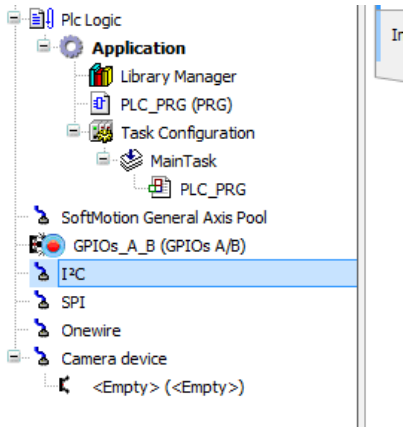
Figure 24: The updated control system design.

The main program is written in the graphical FBD-language with a function blocks representing the various devices and functions such as the gyro board, the PID controller and the motor driver board.

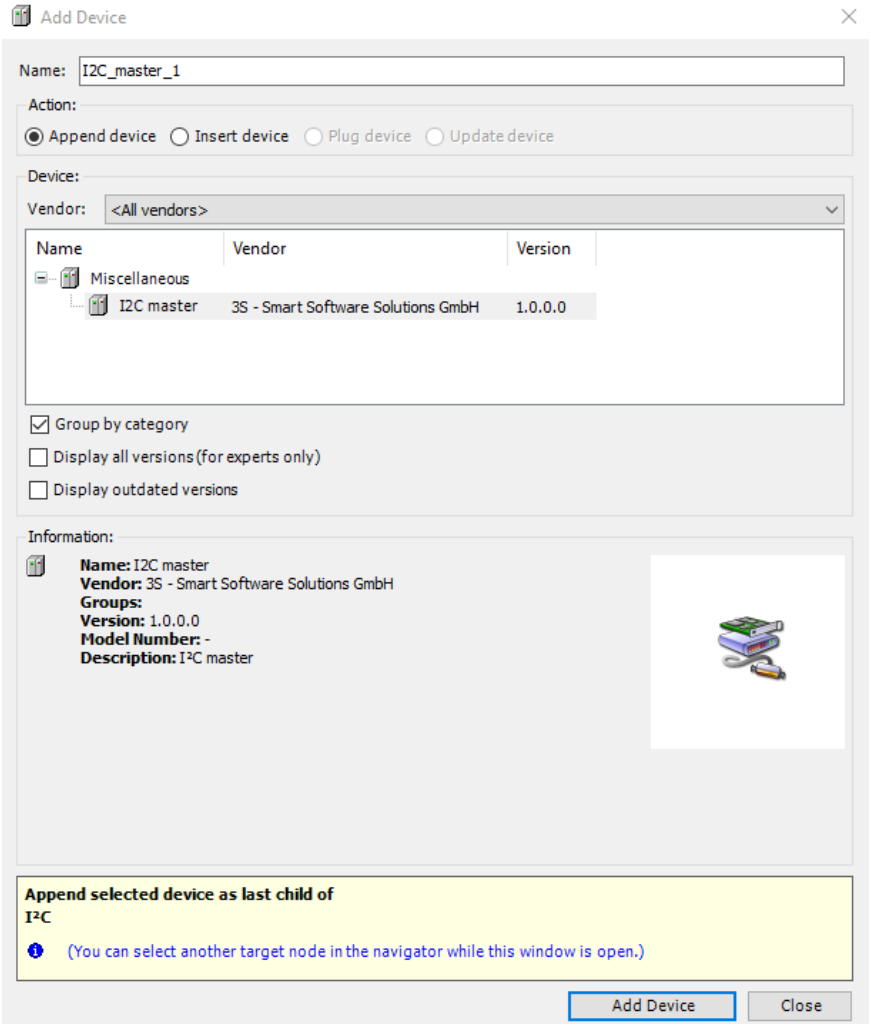
3.5.1 MPU9150 gyro board

This device communicates via I²C and is implemented as follows:

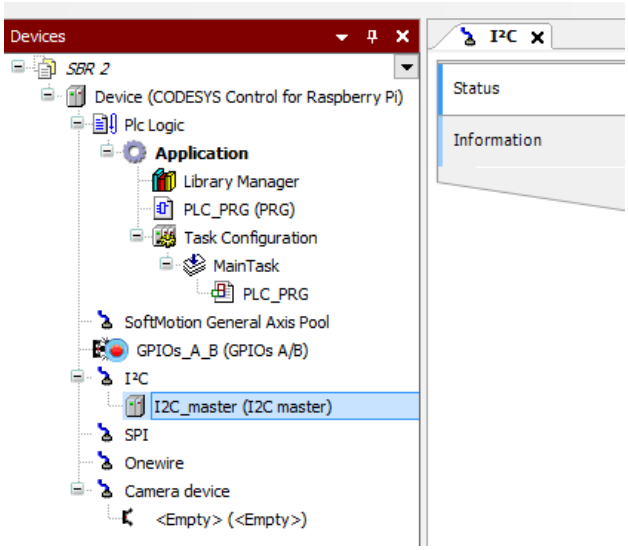
a. Right click “I²C” in the device-tree



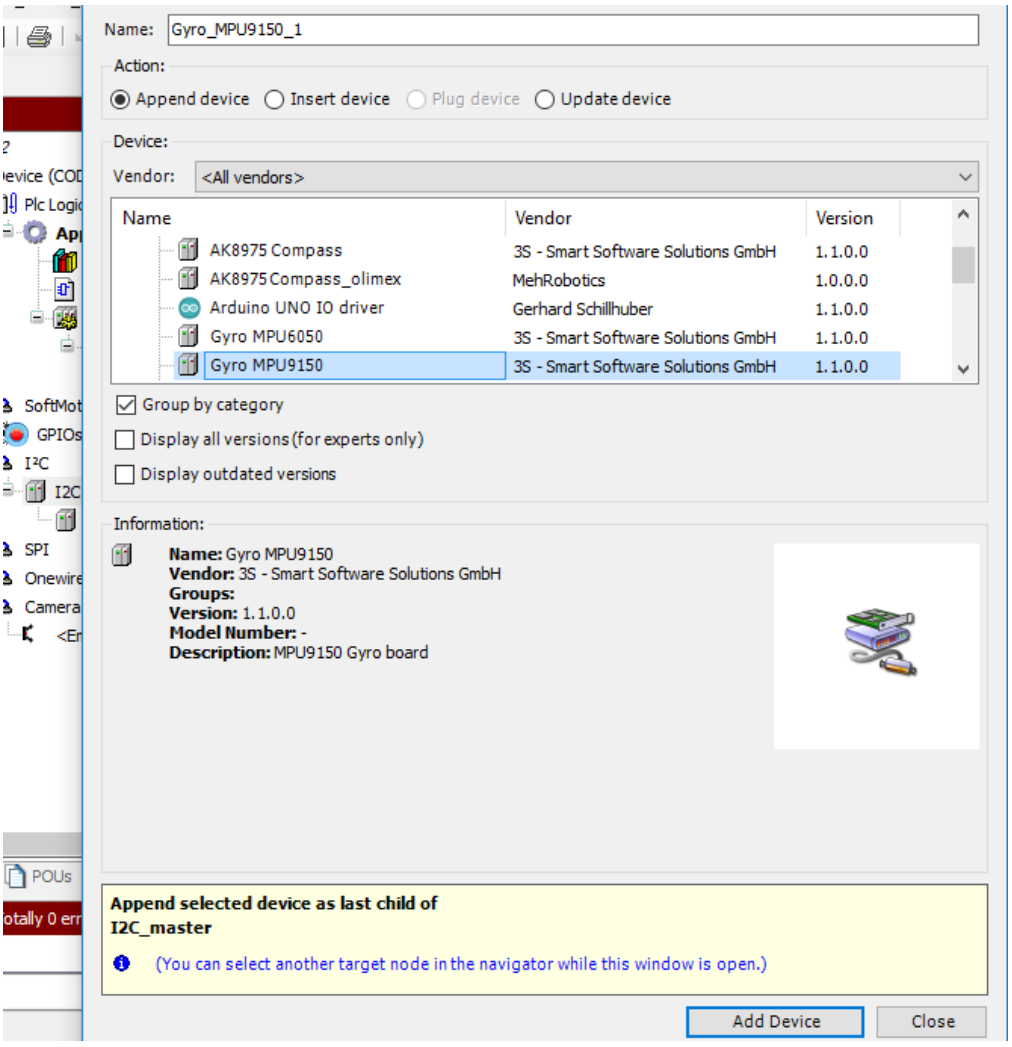
b. Click “Add Device”, select “I2C master” and press “Add Device” in the bottom right corner.



c. Now right click “I2C_master (I2C master)”

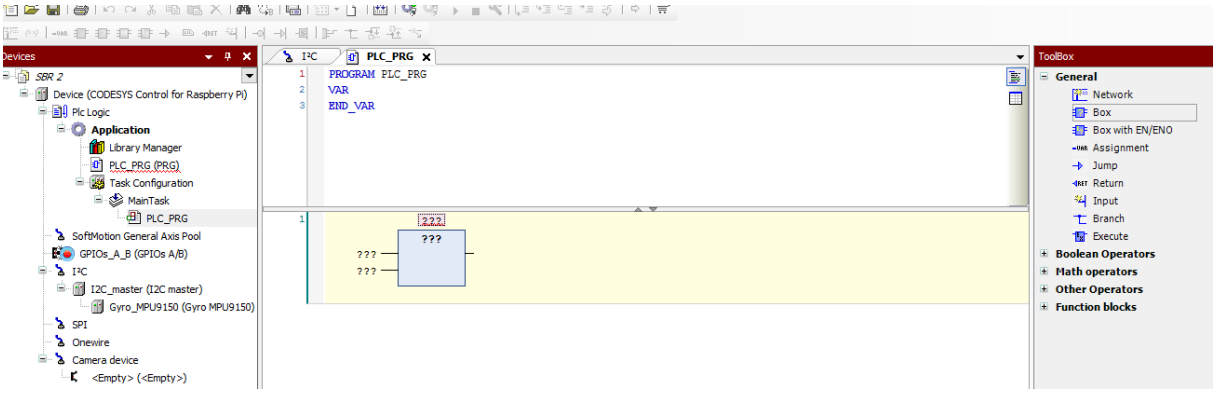


d. Click “Add Device”, select “Gyro MPU9150” and press “Add Device” in the bottom right corner.

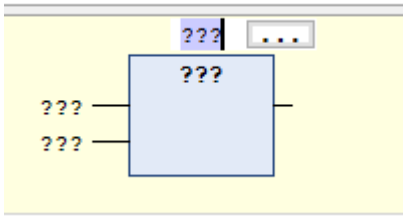


The gyro function block can now be inserted in the main program:

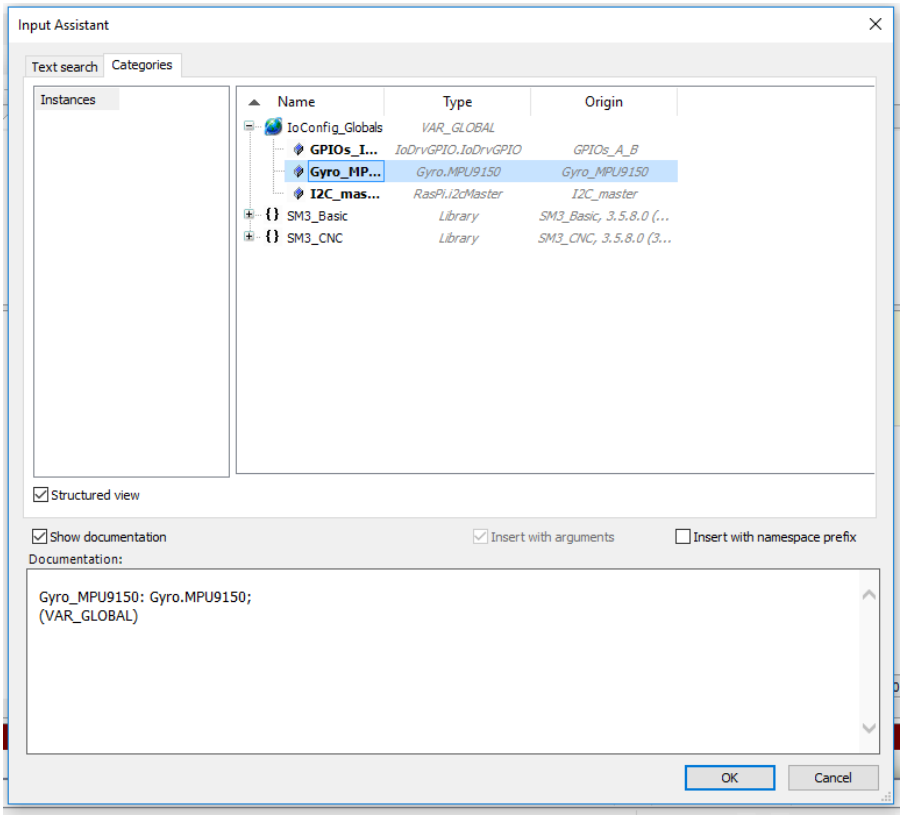
e. In the device-tree; go to the main program and insert an empty box from the “Toolbox”.



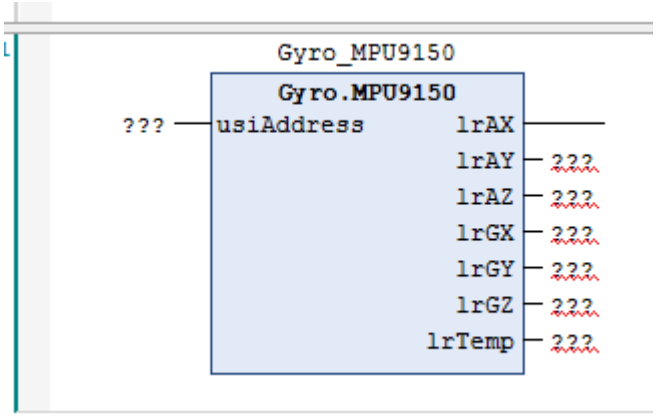
f. Open the Input assistant by pressing the three question marks <u>over</u> the box and click the three dots that appear to the right.



g. Expand “IoConfig_Globals” and select “Gyro_MPU9150”. Press “OK”



h. Press “ENTER”. The gyro can now be used in the program.



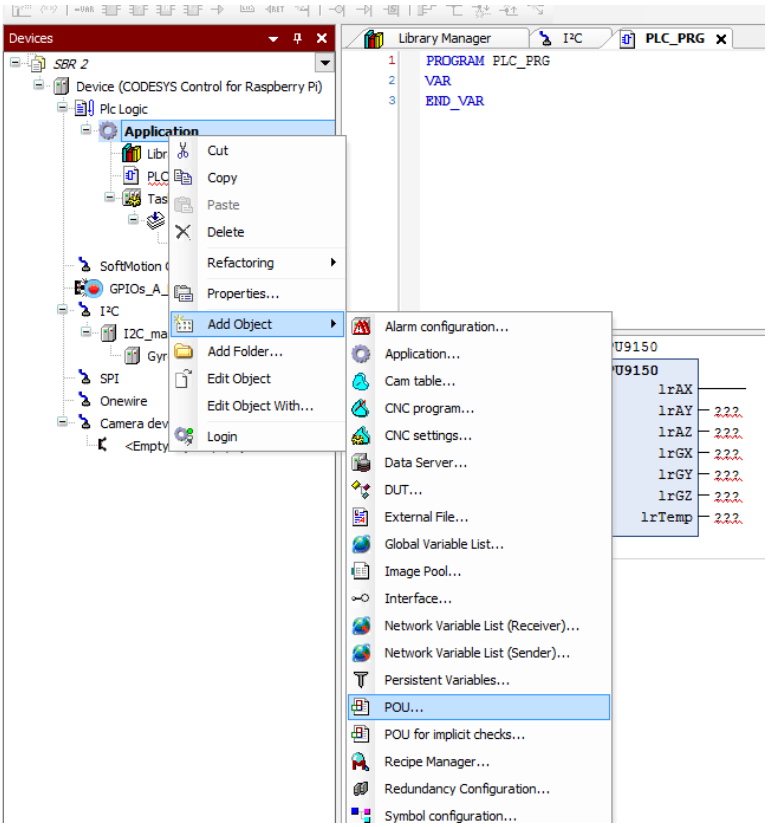
3.5.2 Calculate the angle

To calculate the angle with which the robot is tilted relative to the gravity vector, the function atan2() is used. The function is then implemented as a function block (FB) in the main program.

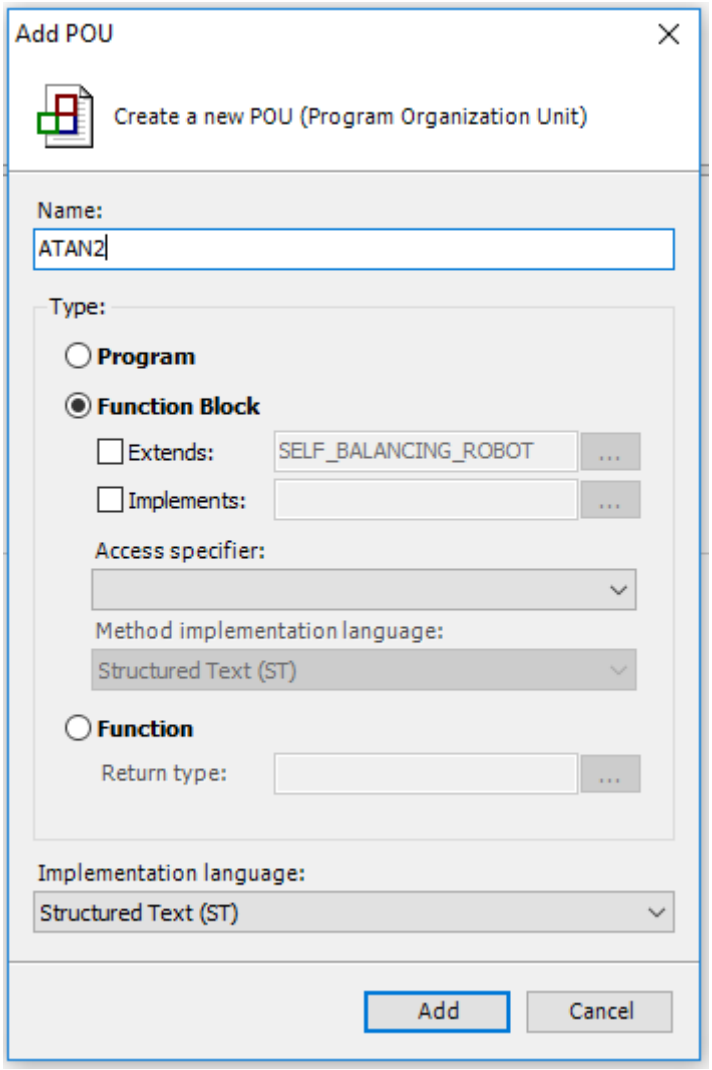
where the input is two vectors from the accelerometer on the gyro board and the output is the robots angle relative to the gravity vector.

The function block (FB) is created by following the procedure below:

a. In the device tree; right click “Application”, navigate to “Add Object” and press “POU…”.



b. The name of the FB is set to ATAN2, type is set to “Function block” and the implementation language is Structured Text (ST).

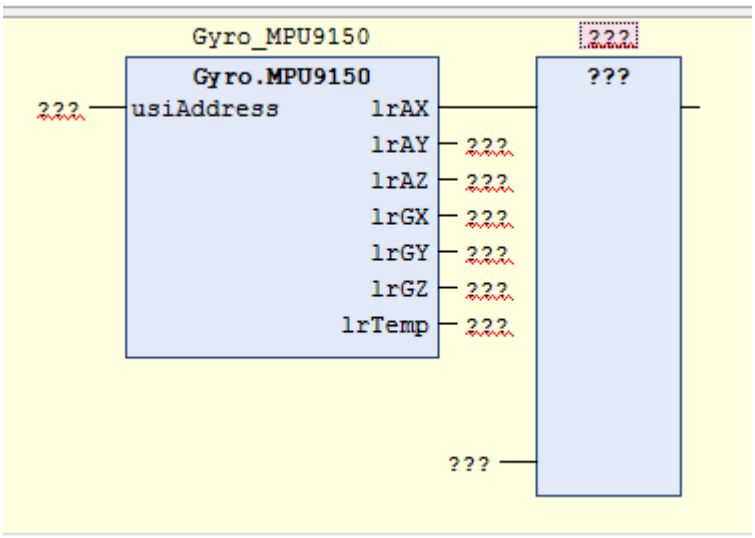


c. The input and output variables are chosen and the atan2 function is written as a set of if-statements per the atan2-defintion. To avoid NaN- (Not a Number) errors, atan2(0,0) is defined as 0. Finally, the angle is converted from radians to degrees.

```
FUNCTION_BLOCK ATAN2 //atan2 (y/x)
VAR_INPUT
    y: LREAL;
    x: LREAL;
END_VAR
VAR_OUTPUT
    angle: LREAL;
END_VAR
```

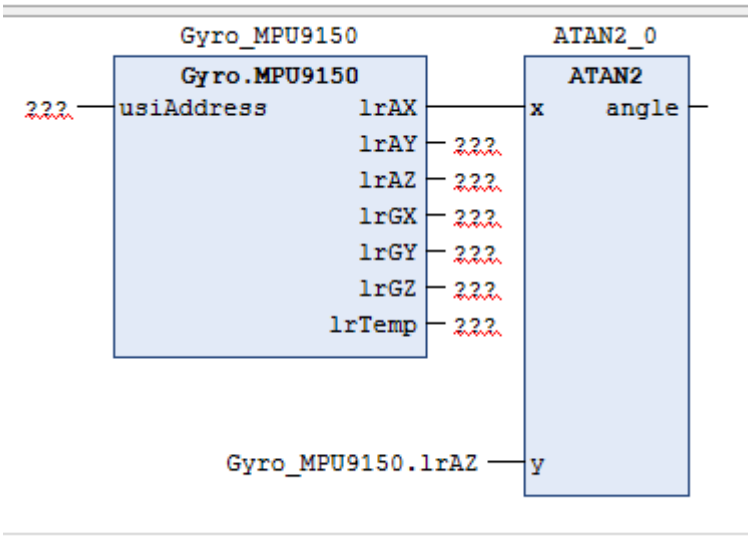
```
IF x > 0 THEN;
    angle := ATAN(y/x);
ELSIF x < 0 THEN
    IF Y >=0 THEN
        angle := ATAN(y/x)+3.14;
    ELSE
        angle := ATAN(y/x)-3.14;
    END_IF
ELSIF y > 0 THEN;
    angle := 3.14/2;
ELSIF y < 0 THEN;
    angle := -1*(3.14/2);
ELSE
    angle := 0; //if x=0 and y=0, angle = 0
END_IF

angle := angle * (180/3.14); //Convert radians to degrees
```

d. In the main program, insert an empty box from the “Toolbox” and connect it to the Ax-vector on the gyro-function block.



e. Use the “Input Assistant” to insert the ATAN2 function block.

f. Use the “Input Assistant” to choose the y-input to the ATAN2 (FB).



The function block (FB) now produces the unfiltered angle that will be put through the Kalman filter and then used as the actual value for the PID controller.

3.5.3 Kalman Filter

To implement the Kalman filter as a CODESYS function block (FB) the equations described in section 2.4 are simplified and the matrixes are broken down so they can be used in the code.

The filter is implemented as described by equations 1-8, chapter 2.4. The conditions for this application are inserted in the equations and after simplification they can be written as Structured Text (ST) in an CODESYS function block (FB) (Figure 25). This process is

described on the blog TKJ Electronics [20] where the Kalman-Filter equations are transformed into C code.

The input values are:

a) d_time – the cycle time in seconds, is calculated in the FB.
b) newRate – the measured rate from the MPU
c) newAngle – the calculated angle from the ATAN2 FB.
d) Q_angle – the variance of the measured angle from the ATAN2 FB, this is a constant that needs to be trimmed.
e) Q_bias – the variance of the rate bias, this is a constant that need to be trimmed.
f) R_measure – measurement noise, this is a constant that needs to be trimmed.

Time update ("Predict")

1) Project the state ahead

$$\begin{bmatrix} \theta \\ \dot{\theta}_{bias} \end{bmatrix}_k^- = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_{bias} \end{bmatrix}_{k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_{k-1} \tag{8}$$

ST:

```
// Time Update ("Predict")
// 1) Project the state ahead
rate := newRate - bias;
angle := angle + d_time * rate;
```

2) Project the error covariance ahead

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_k^- = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_{bias}} \end{bmatrix} \Delta t \tag{9}$$

ST:

```
// 2) Project the error covariance ahead
P_00 := P_00 + d_time * (d_time * P_11 - P_01 - P_10 + Q_angle);
P_01 := P_01 - d_time * P_11;
P_10 := P_10 - d_time * P_11;
P_11 := P_11 + Q_bias * d_time;
```

Measurement update ("Correct")

1) Compute the Kalman gain

$$\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_k^- \begin{bmatrix} 1 \\ 0 \end{bmatrix} S_k^{-1} \tag{10}$$

where the observation covariance S is calculated:

$$S_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_k^- \begin{bmatrix} 1 \\ 0 \end{bmatrix} + R \tag{11}$$

ST:

```
// Measurement Update ("Correct")
// 1) Compute the Kalman gain K where covariance prediction Sk:
Sk := P_00 + R_measure; // Estimate error

// Kalman gain
K_0 := P_00 / Sk;
K_1 := P_10 / Sk;
```

2) Update estimate with measurement innovation z_k

$$\begin{bmatrix} \theta \\ \dot{\theta}_{bias} \end{bmatrix}_k = \begin{bmatrix} \theta \\ \dot{\theta}_{bias} \end{bmatrix}_k^- + \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k y_k \tag{12}$$

where the innovation y is calculated:

$$y_k = z_k - [1 \quad 0] \begin{bmatrix} \theta \\ \dot{\theta}_{bias} \end{bmatrix}_k^- \tag{13}$$

Here is also the validation gate of the innovation implemented. ST:

```
// 2) Update estimate with measurement zk (newAngle) where the innovation y:

y := newAngle - angle;

//Reject measured values outside the innovation covariance limits (3sigma = 99.7% of mean)

RQ_lim := 3 * SQRT(Sk);
RQ_lim_negative := -1 * RQ_lim;

IF y > RQ_lim OR y < RQ_lim_negative THEN
    K_0 := 0;
    K_1 := 0;
ELSE

//Update estimate
angle := angle + K_0 * y;
bias := bias + K_1 * y;

END_IF
```

3) Update the error covariance

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_k = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k [1 \quad 0] \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_k^- \tag{14}$$

ST:

```
// 3) Update the error covariance
P00_temp := P_00;
P01_temp := P_01;
P_00 := P_00 - K_0 * P00_temp;
P_01 := P_01 - K_0 * P01_temp;
P_10 := P_10 - K_1 * P00_temp;
P_11 := P_11 - K_1 * P01_temp;
```

The output is the new estimated angle which is used as the actual value by the PID-controller.

After a datatype conversion from LREAL to REAL the newAngle-input on the Kalman filter is connected to the ATAN2 FB and the newRate-input is connected to the GY-output from the gyro board, the sign of the newRate value is adjusted by multiplying by -1.

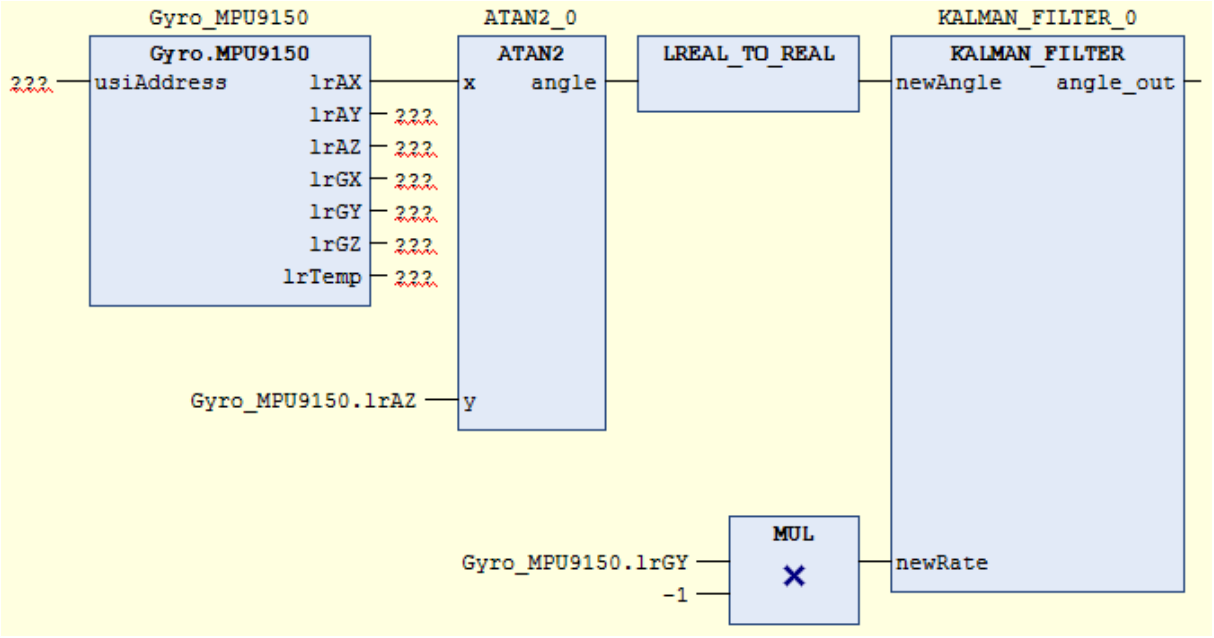


Figure 25: The implementation of the Kalman filter FB in the main program

3.5.4 The PID controller

The PID controller used in this project is included in standard CODESYS library Util, which is added through the Library Manager. The PID controller is implemented as a FB in the main program seen in Figure 26.

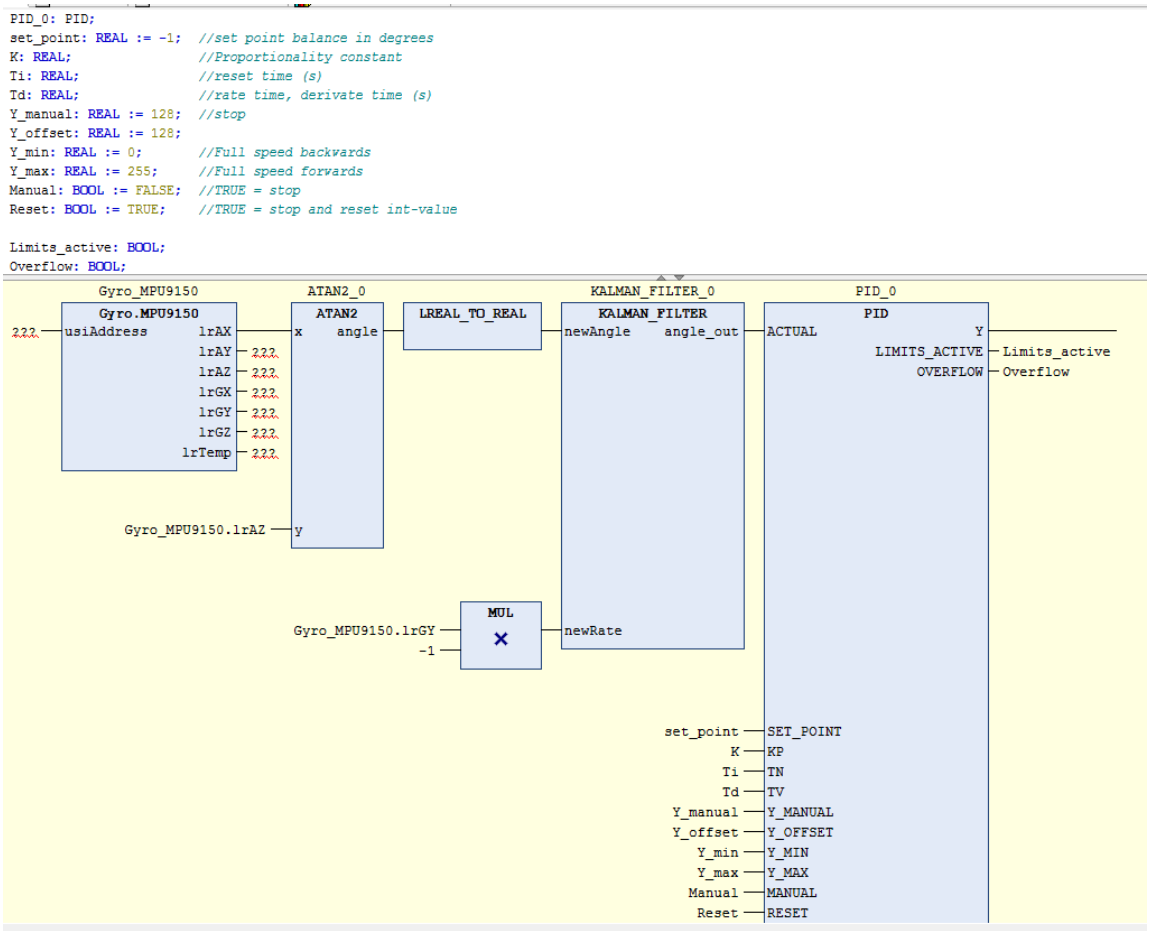


Figure 26: Implementation of the PID FB in the main program

A variable is assigned to each of the FBs inputs and outputs to make them manipulatable online, which is practical during testing and calibration. Table 2 shows the inputs and outputs of the PID-controller.

Table 2: The inputs and outputs of the PID-function block

Input	**Description**
ACTUAL	Actual angle, provided by the gyro board via the Kalman filter
SET_POINT	Desired angle
KP	Proportionality constant (K)
TN	Reset time (Ti) in seconds
TV	Rate time, derivate time (Td) in seconds
Y_MANUAL	Y is set to this value if MANUAL = TRUE (128=stop)
Y_OFFSET	Offset for manipulated value (128 = stop)
Y_MIN	Minimum value for the manipulated value = full speed backward (0)
Y_MAX	Maximum value for the manipulated value = full speed forwards (255)
MANUAL	If TRUE, Y is not influenced by controller
RESET	If TRUE, Y = Y_OFFSET and the integral part is reset
Output	
Y	Manipulated value, the speed output sent to the MD25

LIMITS_ACTIVE	Optional set value in case Y would exceed Y_MIN or Y_MAX
OVERFLOW	Indicate an overflow in the integral part (FALSE/TRUE)

3.5.5 MD25 motor driver board

The MD25 motor driver board is connected to the Raspberry Pi via I²C, and implemented in CODESYS in the same way as the MPU9150-gyro board; as an I²C -device available in the device list.

To support a device this way CODESYS uses a device description file (.devdesc.xml) and a library file (.library) containing the function block (FB) according to which the functionality of the device is programmed.

Since this device is not available as a pre-installed device in CODESYS, a custom library and device description file is created. This procedure is described in the provided documentation [10] and is implemented as follows.

First a device description file is created using an existing device description file as the template:

a. An existing I²C device description file (AdafruitPWM.devdesc) is copied and pasted, the new file is renamed MD25.
b. The file is opened in a text editor and id is changed.

```
<Device hideInCatalogue="false">
  <DeviceIdentification>
    <Type>500</Type>
    <Id>FFFF 1111</Id>
    <Version>1.0.0.0</Version>
  </DeviceIdentification>
```

c. The device information is changed

```
<DeviceInfo>
    <Name name="local:ModelName">MD25</Name>
    <Description name="local:DeviceDescription">MD25</Description>
    <Vendor name="local:VendorName">Emil Eriksson</Vendor>
    <OrderNumber>-</OrderNumber>
```

d. The vendor name and version of the related library is changed

```
<DriverInfo needsBusCycle="false">
    <RequiredLib libname="MD25" vendor="Emil Eriksson" version="1.0.0.0" identifier="deviceLib">
      <FBInstance basename="$(DeviceName)" fbname="MD25">
```

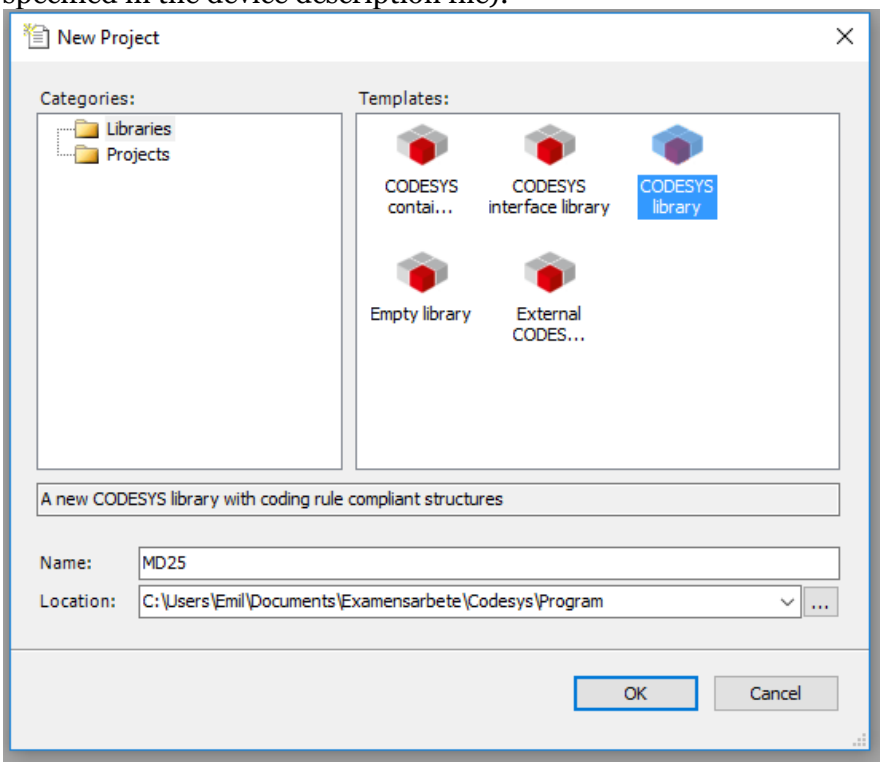
e. The name of the FB in the library is entered and the file is saved and closed

```
<RequiredLib libname="MD25" vendor="Emil Eriksson" version="1.0.0.0" identifier="deviceLib">
    <FBInstance basename="$(DeviceName)" fbname="MD25">
      <Initialize methodName="Initialize" />
      <CyclicCall methodname="BeforeWriteOutputs" task="#buscycletask" whentocall="beforeWriteOutputs" />
      <CyclicCall methodname="AfterReadInputs" task="#buscycletask" whentocall="afterReadInputs" />
    </FBInstance>
</RequiredLib>
```

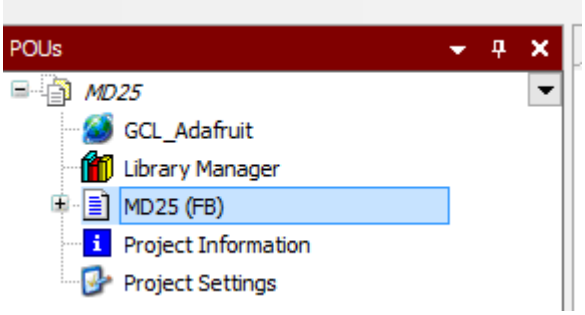
The device description file is then installed in CODESYS via Tools -> Device Repository… and can now be inserted as a I²C -device.

The library file is created per the method below, using an existing library as template:

a) In CODESYS, create a new project and select CODESYS library file named MD25 (as specified in the device description file).



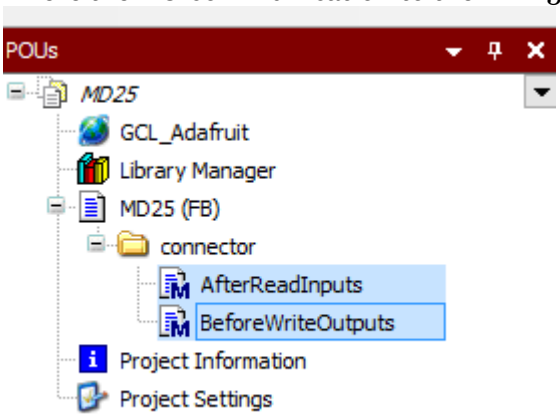
b) Create a new FB extending I²C, also named MD25 (as specified in the device description file).



c) In the FB, the I²C -adress of the MD25 is specified (0x58) and a state machine is implemented.

```
1   SUPER^();
2
3   CASE _iState OF
4   0:
5       IF usiAddress = 0 THEN
6           usiAddress := 16#58;    //i2c-adress 0x58
7       END_IF
8
9       IF init() THEN
10          _iState := 10;
11      END_IF
12  END_CASE
```

d) To read the input values from the MD25 a method is created called AfterReadInputs where the I²C-communication to the MD25s registers are specified.



The MD25 has 17 registers specified in the MD25-I²C-documentation [12], of which 16 are readable; Speed1-value, Speed2/Turn-value, the individual encoder bytes (which are used to calculate the encoder value), the remaining voltage of the battery, the electric current through the motors, the software revision number, acceleration rate and the mode of operation activated.

```
SUPER^.BeforeWriteOutputs();

IF _iState = 10 THEN
    write8(16#00, speedOne); //speed motor 1
    write8(16#01, speedTwo); //speed motor 2/ turn value
    write8(16#0E, AccRate); //Acceleration rate
    write8(16#0F, op_mode); //Mode of operation
    write8(16#10, comm);    //Command register
END_IF
```

e) To write output values to the MD25 a method called BeforeWriteOutputs is created in a similar manner. The MD25 has 5 writeable registers; Speed1-value, Speed2/Turn-value, acceleration rate, mode of operation and a command register.

```
SUPER^.AfterReadInputs();

IF _iState = 10 THEN
    speedmotorONE := read8(16#00); //Read speed of first motor
    speedmotorTWO := read8(16#01); //Read speed of second motor

    //Calculate encoder value
    r1a := read8(16#02);
    r1b := read8(16#03);
    r1c := read8(16#04);
    r1d := read8(16#05);
    r2a := read8(16#06);
    r2b := read8(16#07);
    r2c := read8(16#08);
    r2d := read8(16#09);

    positionOne := (r1a*256*256*256)+(r1b*256*256)+(r1c*256)+r1d; //Read position value encoder 1
    positionTwo := (r2a*256*256*256)+(r2b*256*256)+(r2c*256)+r2d; //Read position value encoder 2

    BatteryV := read8(16#0A); //Read battery voltage (value 118 = 11.8V)

    currentOne :=read8(16#0B); //Read current through motor 1
    currentTwo :=read8(16#0C); //Read current through motor 2
END_IF
```

The library is then saved and installed in CODESYS via the library repository.

The MD25 is implemented in the main project as an I²C-device and the FB is implemented in the main program (Figure 27).

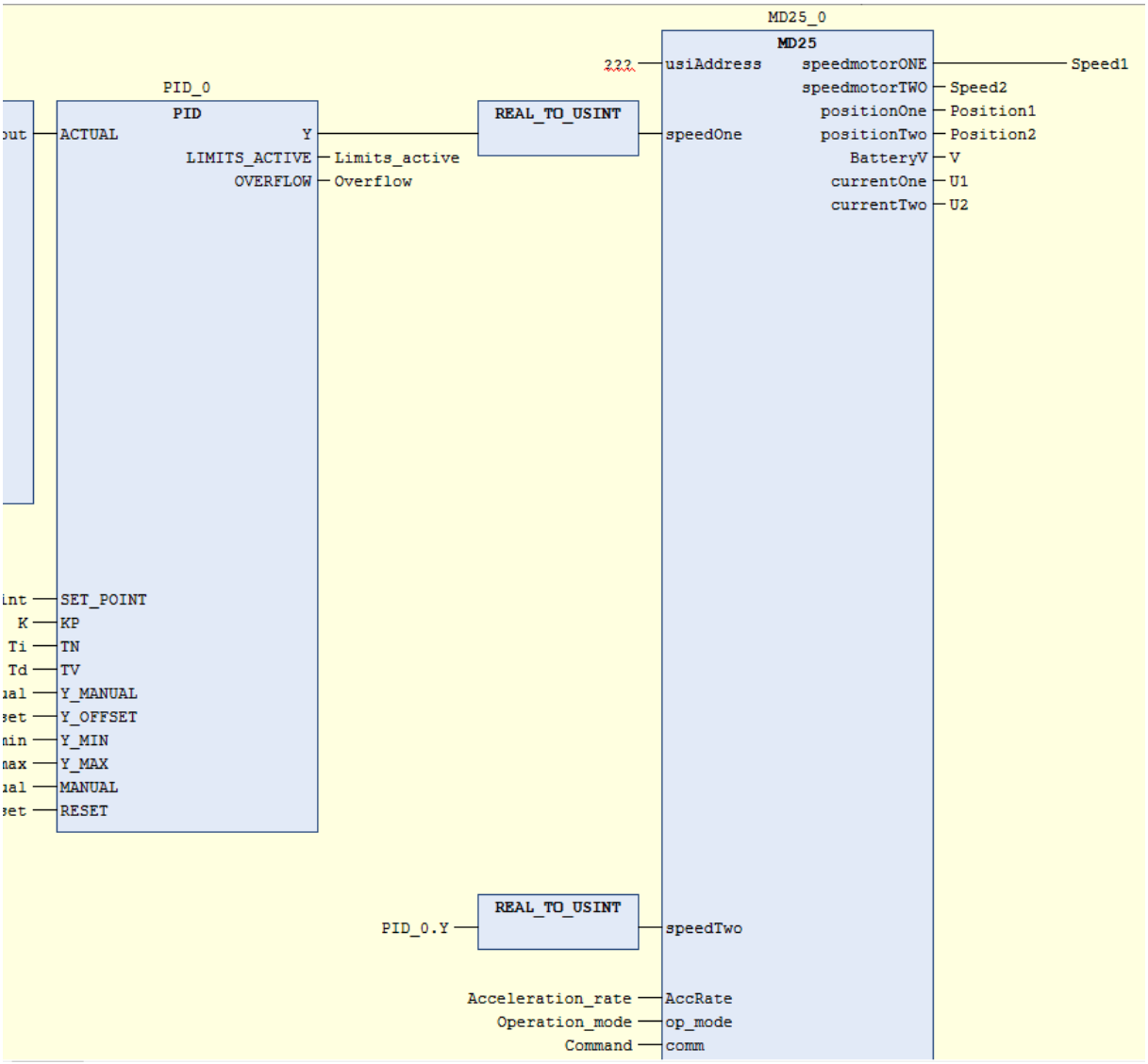


Figure 27: Implementation of the MD25 FB in the main program

3.6 Visualizations

The CODESYS visualizations-tool is used to create the GUI (Graphical User Interface) from where the robot is controlled and to create trace plot of the different processes. As an in-built function in CODESYS this visualization can be accessed through a web browser by any device connected to the same local network as the Raspberry Pi. In this way, for example a smartphone can be used as a remote control to steer the robot via Wi-Fi.

3.6.1 GUI

The GUI, shown in Figure 28 is equipped with buttons, lamps and switches to remotely control the robot. The reset function of the Kalman Filter and the PID controller is accessed through the GUI as well as adjustable PID parameters for online-tuning of the controller. Also, buttons for acceleration, braking and steering is laid out but not implemented.

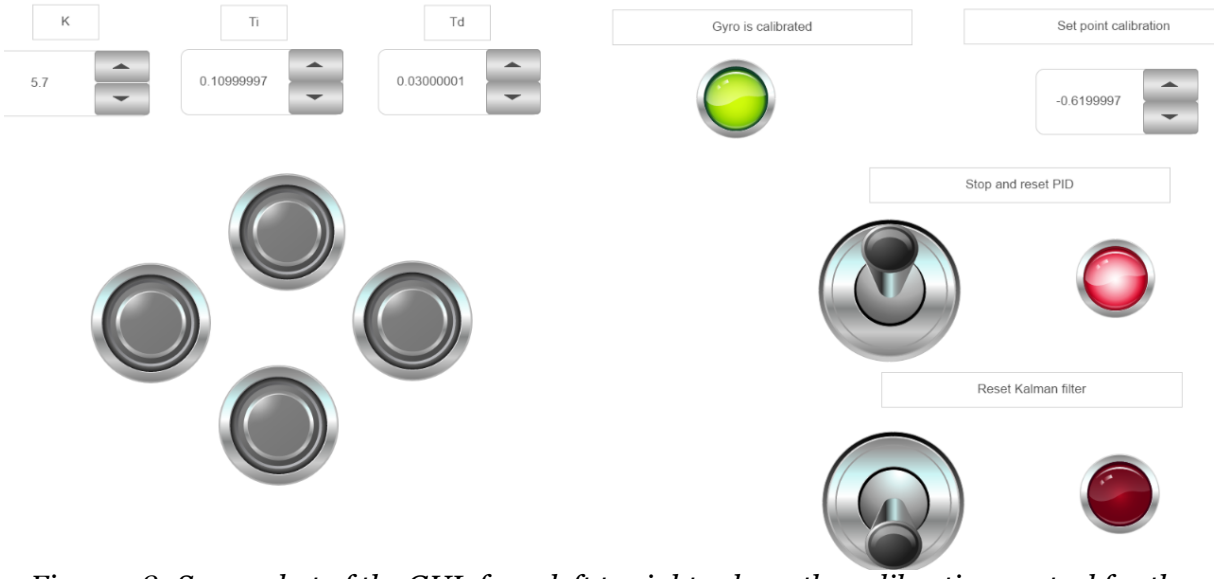


Figure 28: Screenshot of the GUI, from left to right; above the calibration control for the PID, a green lamp shows when the gyro is calibrated, the set point angle for when the robot is balanced is calibrated manually, below; the buttons for steering and acceleration (not implemented), a switch for stopping the robot and reset the PID (red lamp shows when active), and a switch for resetting the Kalman Filter in case it loses its tracking (red lamp shows when active).

3.6.2 Trace Plots

A separate visualization (Figure 29) is created for plotting the Kalman Filter- and PID-performance to be used for tuning.

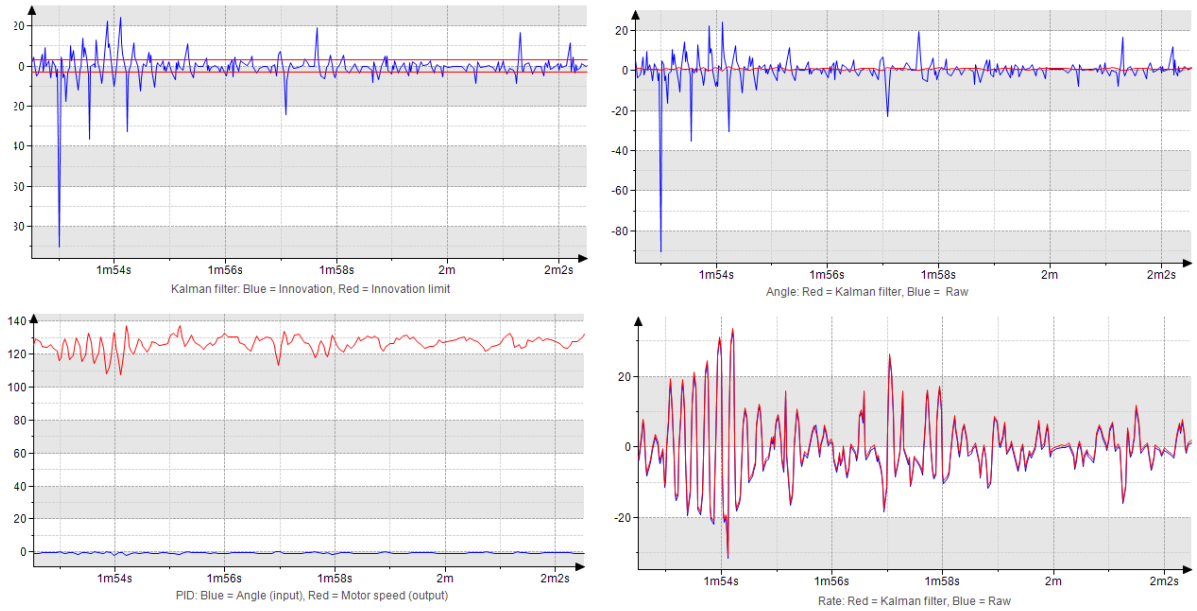


Figure 29: Trace plot visualization

4 Experiments, results and analysis

4.1 Kalman Filter tuning and performance

The Kalman Filter is tuned by measuring the angle, first in a static state, and then during a balance test. To determine the variables Q_angle, Q_bias and R_measure, three trace-plots are set up the visualization tool in CODESYS showing angle (filtered versus raw), rate (filtered versus raw) and innovation versus innovation covariance.
The variables Q_angle, and Q_bias is determined during the static test and R_measure is determined during the balance test.

4.1.1 Static measurements

The static measurements show the raw data from the accelerometer; represented as the angle as it's calculated in the ATAN2 FB, and gyro; represented as the rate, compared with the filtered values from the Kalman filter while the robot is standing still on the desk with the wheels removed.

The process noise, Q_angle, is determined by comparing the filtered and the raw angle. The filtered angle should follow the approximated mean of the raw angle as seen in Figure 30.

Q_angle is set to 0,001

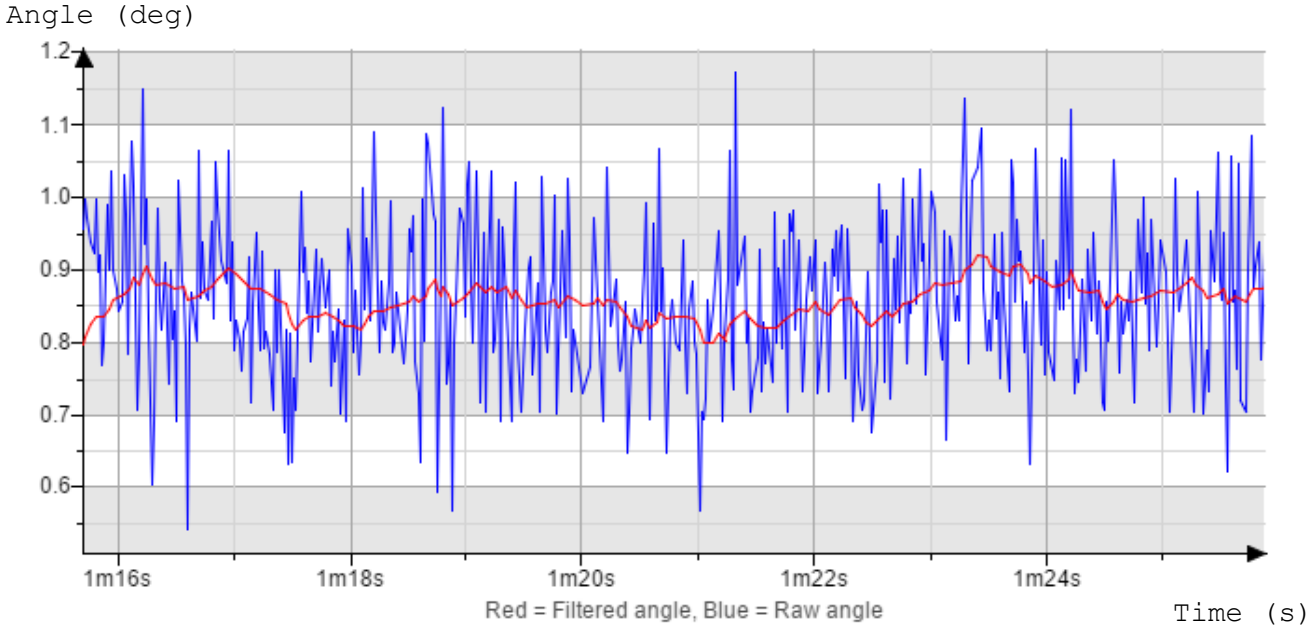


Figure 30: Observed angle and estimated angle plotted with respect to time.

Gyro drift is modelled as integrated white noise, a Brownian walk (Figure 31). The noise for the bias walk is set to Q_bias, 0,03.

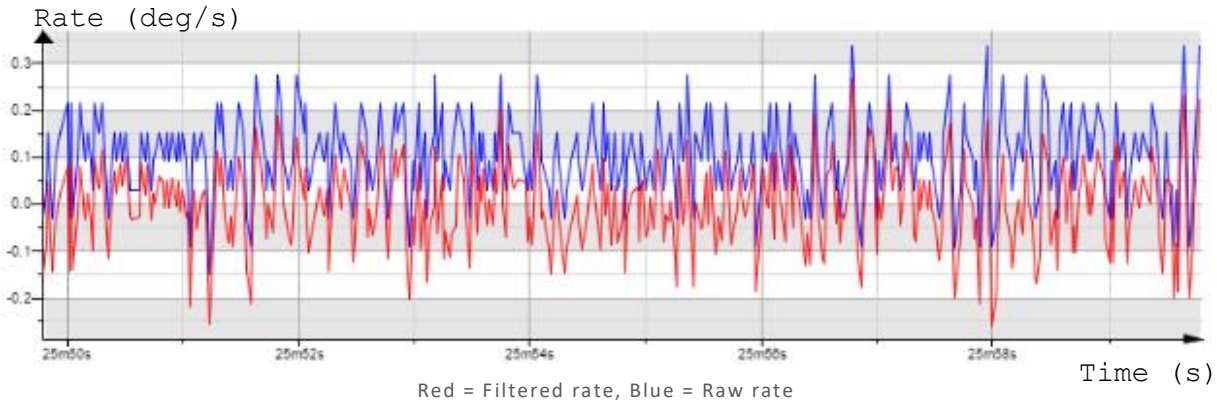


Figure 31: Gyro drift is modelled as integrated white noise.

A balance test is performed to determine the value for the variable R_measure by analyzing the innovation-plot seen in Figure 32, comparing the innovation, y, with the innovation validation gate, which is: $\pm 3{,}0 \times \sqrt{S}$. Validation gate width is ± three standard deviations (99,7%) of the innovation should be when the filter is tuned.

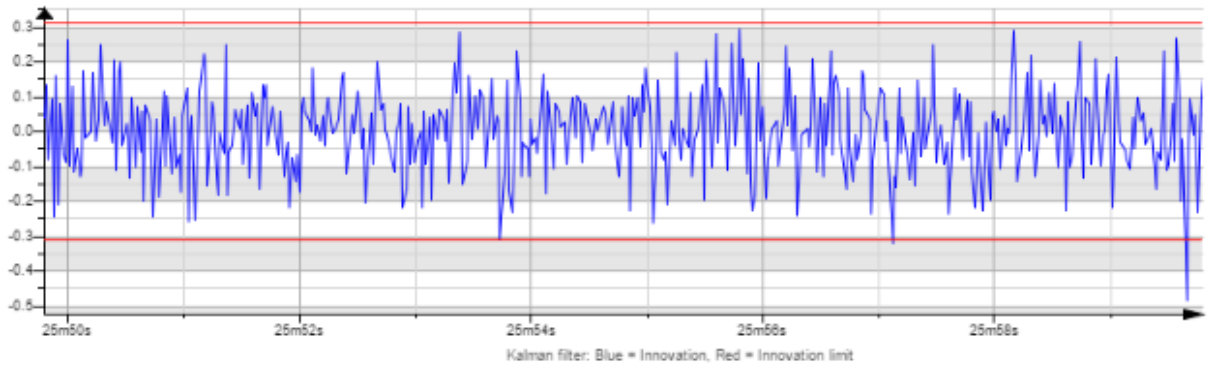


Figure 32: Validation gate is used to tune the filter so that the innovation spikes are filtered out.

Since measured values causing the innovation to go outside of this area are filtered out, R_measure is initially set to a large value (R_measure=5). Otherwise the Kalman filter will lose tracking.

The tuning is then performed parallel to the tuning of the PID controller
As the robots' ability to hold its balance increases, R_measure can be reduced.

When tuned, R_measure is set to 0,8.

4.2 PID-tuning and performance

To tune the PID controller the Ziegler-Nichols method [27] is used to get a first approximation of the K-, T_I-, and T_D-values. These values are then adjusted based on the balancing behavior of the robot.

The working order is as follows:

a) Initially, K =5, T_I = 1000000000 and T_D =0.
b) K is increased until the robot starts to oscillate about the balance position, this happened at K_o (K-zero) = 9,5.
c) The angle-plot is used to approximate the period time, T_o (T-zero) for the oscillations. Five different measurements are used to calculate To and the mean of these values are calculated in an excel-spreadsheet. The period time is approximated as = 0,43 s.
d) The values K_o = 9,5 and T_o = 0,43 are then used in Table 3 to determine the controller parameters.

Table 3: PID controller parameters according to the Ziegler-Nichols method

K_o = 9,5 T_o = 0,43	Parameters		
	K	T_I	T_D
PID controller	0,6 K_o	0,5 T_o	0,125 T_o
Tuned parameters	**5,7**	**0,22**	**0,054**

The PID parameters according to the Ziegler-Nichols method are: K =5,7, T_I = 0,22 and T_D =0,054.

This is a first approximation and a fine tuning of the PID controller by trial and error is performed which results in the following PID-parameters: K =5,7, T_I = 0,1 and T_D =0,03.

4.3 Motor and MD25 performance

An irregular twitching in the motors where noticed even though the input signal is 128, meaning the motors are set to stop.
The wheels where removed and the input is set to 128 (stop). The encoder values on both motors where plotted as shown in Figure 33 and Figure 34.

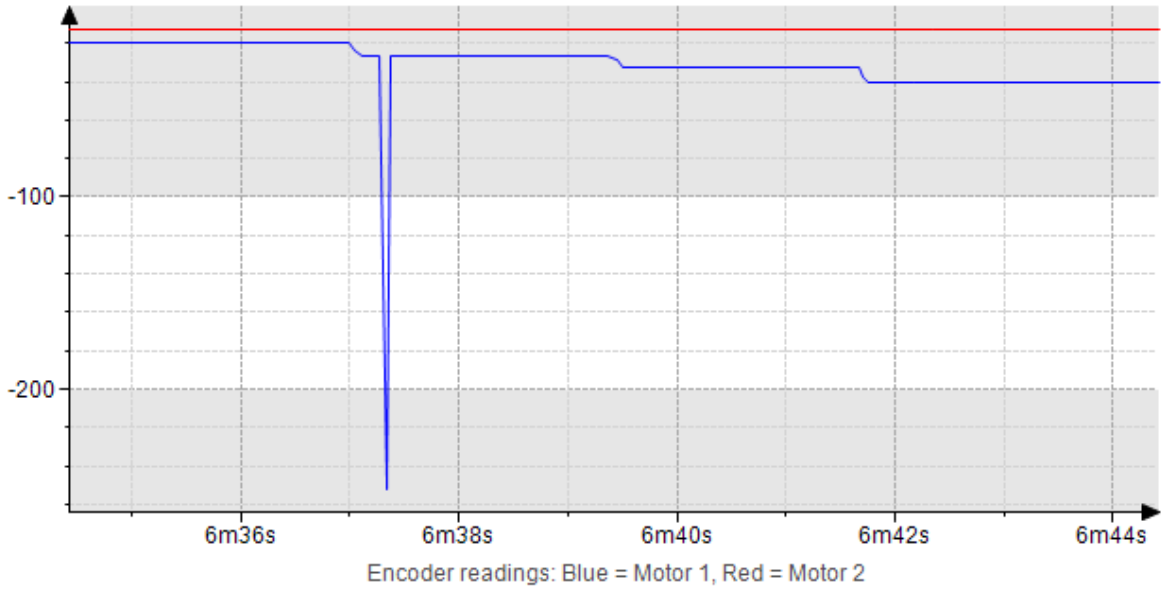


Figure 33: Encoder readings of twitching motors

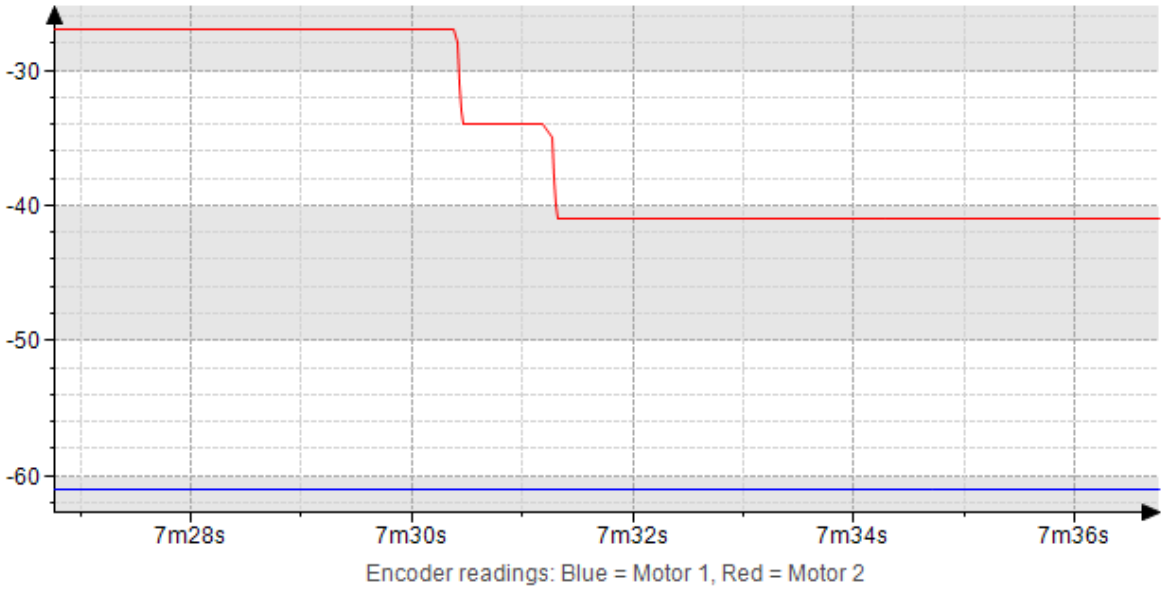


Figure 34: Encoder readings of twitching motors

4.4 Specifications and test scenario analysis

The specifications from section 1.1.1 and status (**met**/**not met**) at the end of the project:

1. The robot should be equipped with a Raspberry Pi with CODESYS (high priority) **met**
2. The robot should have TWO wheels (high priority) **met**
3. Controller parameters should be controlled via web-interface (high priority) **met**
4. The robot should be equipped with a gyro breakout board **met**
5. The robot should be able to keep its balance like an inverted pendulum **met**
6. The robot should be able to avoid collision by using a distance sensor **not met**
7. The robot should be controlled using a smartphone (web-interface) **met**
8. The robot should have on-board power supply in the form of batteries. The size of the batteries should be based on the power usage of the electronics. **met**
9. Testing activities should be noted and filmed **met**
10. Web-interface should present total travelled distance (low priority) **not met**

Result: 8/10 specifications are met.

The test scenarios from section 1.1.1 and status (**success**/**fail**/**not completed**) at the end of the project:

1. The robot is balancing on a flat surface and a tilted surface (30°). **partially completed**
2. The robot is wireless controlled. **success**
3. The robot brakes automatically for obstacles. **not completed**
4. The robot brakes automatically and drives around obstacles. **not completed**

Result: 1/4 test scenarios are completed.

4.5 Test scenario 1

The robot is balancing on a flat surface and a tilted surface (30°).

Only the first part of this test was performed where the robot stayed balanced on a flat surface, shown in Figure 35.

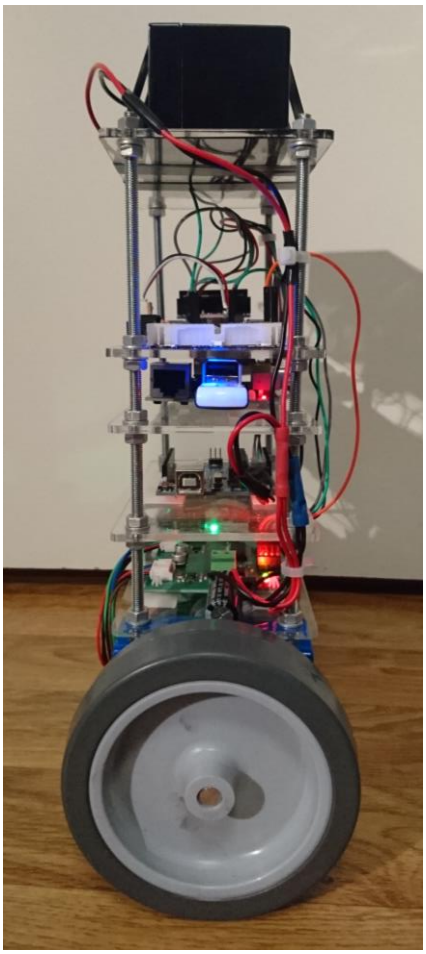


Figure 35: Robot balancing on a flat surface

4.5.1 Plots

Plots showing the performance of the Kalman Filter and PID controller during test scenario 1. Figure 36 shows the innovation plot of the Kalman Filter. Observations outside the area between the red lines are not accepted. Figure 37 shows the performance of the Kalman Filter and Figure 38 shows the performance of the PID controller.

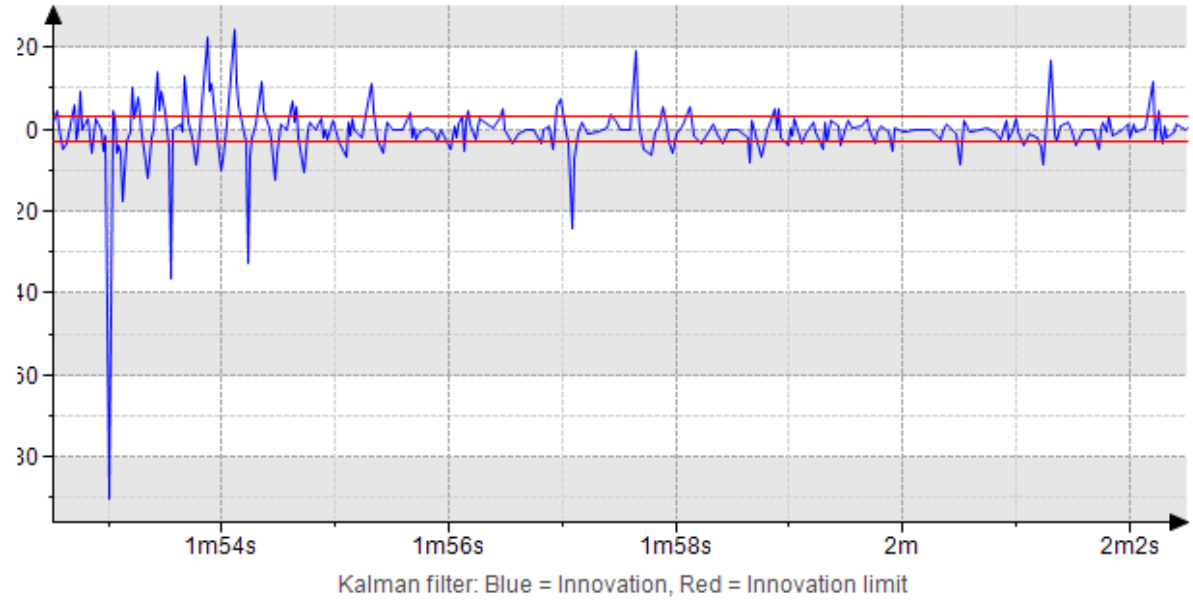


Figure 36: Kalman Filter innovation plot

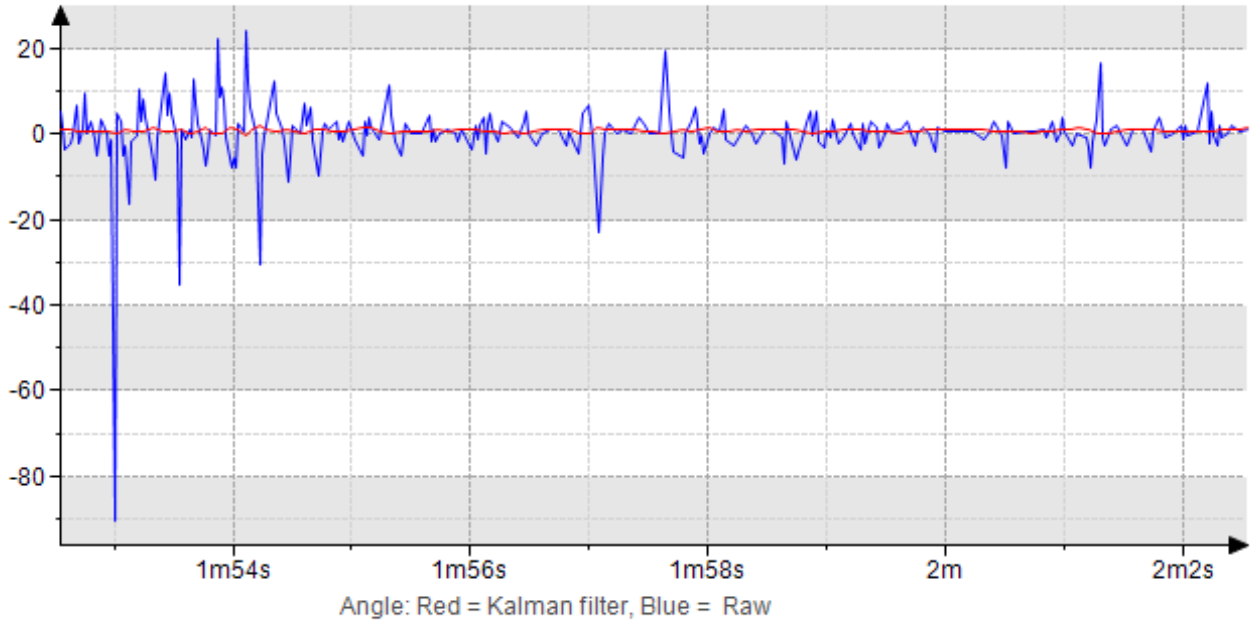


Figure 37: Kalman Filter performance; blue line represents the raw angle, calculated through atan2, and the red line represents the estimated angle produced by the Kalman Filter

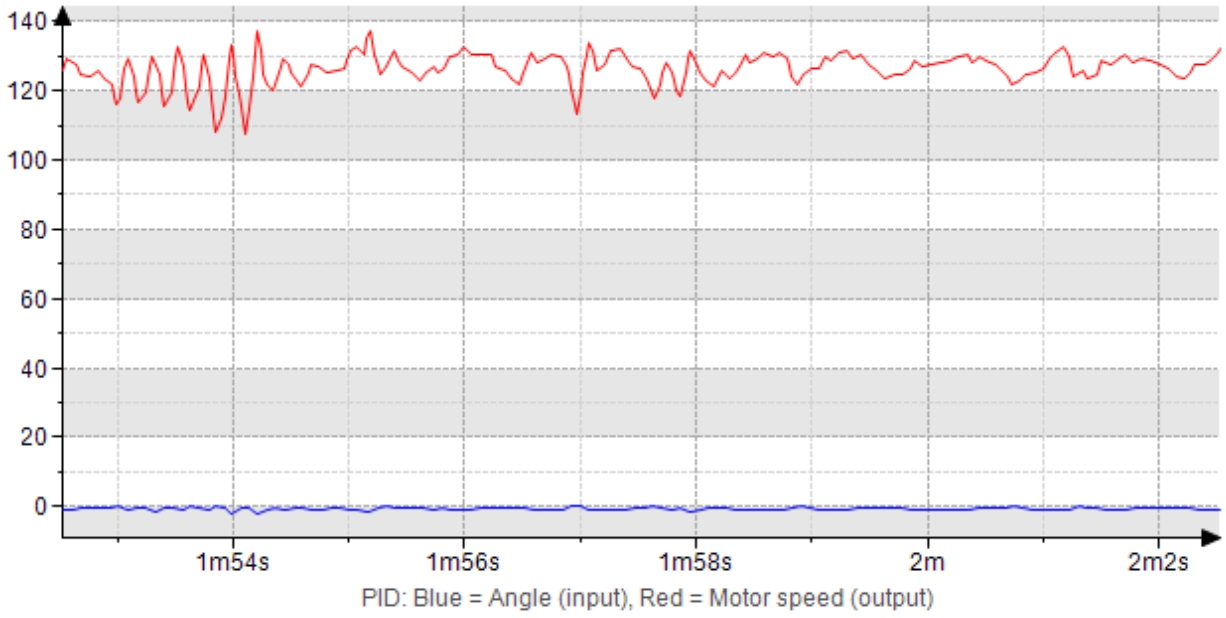


Figure 38: PID Performance; blue line represents the actual value (angle) and the red line represents the control signal (to MD25).

5 Discussions and reflections

Robotics is a multi-disciplinary field of research and requires knowledge in electronics, mechanical engineering, control theory, etc. To be able to build this robot a lot of time was spent on learning as much as possible on these subjects, from basic electronics to programming and control theory. Due to time shortage, not all of the topics could be studied in as much detail. For example; when deciding on the method of PID-tuning, a model-based method was considered but it was decided that, at least for the early tests, the Ziegler-Nichols-method should be used since it doesn't depend on one knowing the transfer function of the system.

During the testing phase the robot fell over several times which eventually damaged the 12V-5V-converter so it had to be replaced. A crash guard was successfully made from a plastic foam sitting pad to prevent this from happening again.
Also, the motors had to be repaired as the cables fell of the electronics board on the back of the motors and had to be soldered back on.

Making a balancing robot requires a robust control system and this was a lot more difficult to achieve than initially imagined. It was first as the Kalman Filter was implemented and tuned correctly as the gyro signal was clear enough to make a good input for the PID-controller.

Tuning the Kalman Filter requires a deep understanding on its functionality and it wasn't until the end of the project that it was realized that problems with the PID-tuning was due to spikes in the gyro signal and the Kalman Filter wasn't properly tuned to handle non-Gaussian distributed noise. The validation gate on the innovation was implemented to filter out the spikes but the filter lost track of the gyro signal when the angle changes where too fast. It took some additional tuning of the Kalman Filter to make it follow the track of the gyro at fast angle changes.

Learning CODESYS was a great experience and thanks to the provided CODESYS documentation, First Steps [10], it was easy to install and run the first programs in CODESYS on the Raspberry Pi to get familiar with the CODESYS environment.

In the future, we can expect robotics and automation to take a growing part in the industrial development. This development is often described as the fourth industrial revolution or Industry 4.0, where the individual sections in a factory is connected to create a "smart factory". Therefore, engineers designing the technology of the future must have a good understanding of automation and autonomous systems.
There is a potential that engineering students building a robot as described in this project, using a system like CODESYS to control it, would learn a lot about robotics and automation in an environment used in industrial applications.

6 Conclusions

During this project, a self-balancing, robot have been designed and built.
A control system design where made and implemented in CODESYS.

The Kalman Filter was successfully implemented and enabled a good reading of the angle from the gyro board. Also, the PID controller was successfully implemented.

The robot can hold its balance on a flat surface and stay balanced.

The main sources of error are the Kalman Filter-, and PID-tuning. The motors' irregularly twitching isn't an obvious source of error compared to these.

The main time of the testing phase where used to make the robot balance on a flat surface. This was very time consuming and therefore were all the testing goals not met before the project run out of time.

The Arduino I/O was tested in CODESYS and gave a good reading of the distance sensor. This never came into practice in the main program because of errors in the Arduino code resulting in failing I^2C communication. This problem wasn't investigated any further due to the low priority of the automatic braking function.

6.1 Future work

This work can be improved by adding a function for motion control, driving back and forth by manipulating the angle target value and left and right by manipulating the speed signal at each motor.
Alternatively, one could use the angle in the y-x-plane measured by the gyro board as the actual value and using a secondary controller to regulate the speed difference between the right and left motor.

To improve the understanding of PLC-systems it would be desired to get the Arduino to work as an I/O-unit, either connected via I^2C as in this project or through MODBUS [28].

Another suggestion is that one could calculate the transfer function for the system, for example by calculating the differential equations describing the dynamic of the mechanical system. This could then be used to tune the PID-controller.

One can also test another type of controller, for example Fuzzy, which has been proven to be an effective way of controlling self-balancing robots [29].

6.2 Reflection on work

A significant amount of the time spent during the testing phase was used trimming the PID controller using improperly filtered values from the gyro board without realizing the Kalman filter was not properly tuned, letting through non-Gaussian distributed spikes.

In the effort of getting rid of the spikes a median filter was constructed and tested with some improvement but not satisfying.

This was very frustrating and time consuming. To avoid this during troubleshooting, one should systematically examine the output data of every process, starting with the sensors, until one find the cause of the problem.

References

[1] Umeå University, "*Umeå University in Figures*," 25 02 2016. [Online]. Available: http://www.umu.se/english/about-umu/facts/figures. [Accessed 04 08 2016].

[2] Umeå University, "*Umeå University,*" 23 10 2013. [Online]. Available: http://www.tfe.umu.se/english/. [Accessed 04 08 2016].

[3] Z. W. Wu Junfeng, "*Research on Control Method of Two-wheeled Self-balancing Robot,*" in *2011 Fourth International Conference on Intelligent Computation Technology and Automation*, Shenzhen, Guangdong, 2011.

[4] Segway Inc., ”*Segway*,” 2017. [Online]. Available: http://se-en.segway.com/. [Använd 04 02 2017].

[5] Gawrisch, "*Segway - English Wikipedia,*" 22 February 2009. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=6015960. [Accessed 4 February 2017].

[6] Mixabest, "*Wikimedia Commons*," 7 March 2008. [Online]. Available: https://commons.wikimedia.org/wiki/File%3AMITSIBISHI_PLC_Panel.jpg. [Accessed 4 February 2017].

[7] Mixabest, "*PLC - English Wikipedia,*" 21 June 2010. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=10696883. [Accessed 4 February 2017].

[8] D. H. Hanssen, "*About PLCs,*" in *PROGRAMMABLE LOGIC CONTROLLERS : a practical approach to IEC 61131-3 using CODESYS*, Singapore, WILEY, 2015, pp. 3-13.

[9] R. P. Foundation, ”*Documentation: Hardware: Raspberry Pi*,” 2016. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md. [Använd 25 12 2016].

[10] 3S-Smart Software Solutions GmbH, *Raspberry Pi with Standard Codesys V3 : First Steps,* 2015.

[11] Arduino, ”*Arduino/Genuino UNO: Arduino & Genuino Products: Arduino,*” 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. [Avaialable 26 12 2016].

[12] R. Electronics, "*MD25 Technical Documentation: Robot Electronics,*" 2016. [Online]. Available: https://www.robot-electronics.co.uk/htm/md25tech.htm. [Accessed 26 12 2016].

[13] InvenSense Inc., *MPU-9150 Product Specification Revision 4.0,* Sunnyvale, CA, USA, 2012.

[14] Sparkfun, "*Using the logic level converter: Tutorials: Sparkfun,*" 2016. [Online]. Available: https://learn.sparkfun.com/tutorials/using-the-logic-level-converter. [Accessed 27 12 2016].

[15] Elektrokit, ”*Avståndssensor IR 10-80cm GP2Y0A21YK: Elektrokit,*” 2016. [Online]. Available: http://www.electrokit.com/avstandssensor-ir-1080cm-gp2y0a21yk.48727?gclid=Cj0KEQiAv4jDBRCC1IvzqqDnkYYBEiQA89utonrtw8csKy0F1rpkrh4VqMB0OsTwxkEiYNzRh27Kj6IaAneP8P8HAQ. [Available 27 12 2016].

[16] 3S-Smart Software Solutions, ”*Download: CODESYS,*” 2016. [Online]. Available: https://www.codesys.com/download.html. [Available 05 08 2016].

[17] D. H. Hanssen, "*CODESYS 2.3,*" in *PROGRAMMABLE LOGIC CONTROLLERS: a practical approach to IEC 61131-3 using CODESYS*, Singapore, WILEY, 2015, pp. 353-354.

[18] W. G. a. B. G, *An Introduction to the Kalman Filter,* Chapel Hill: Department of Computer Science, University of North Carolina at Chapel Hill, 2006.

[19] C. Sundin and F. Thorstensson, "*Autonomus balancing robot; Design and construction of a balancing robot,*" CHALMERS UNIVERSITY OF TECHNOLOGY, Göteborg, Sweden, 2012.

[20] K. S. Lauszus, "*A practical approach to Kalman filter and how to implement it:TKJ Electronics,*" 10 September 2012. [Online]. Available: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/comment-page-1/#comment-57783. [Accessed 18 12 2016].

[21] Sparkfun, "*I2C: Tutorials: Sparkfun,*" 2016. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c. [Accessed 03 01 2017].

[22] 3S-Smart Software Solutions GmbH, "*Forum: CODESYS,*" [Online]. Available: http://forum.codesys.com/. [Accessed 05 08 2016].

[23] Raspberry Pi Foundation, "*Forum: Raspberry Pi,*" [Online]. Available: https://www.raspberrypi.org/forums/. [Accessed 05 08 2016].

[24] Arduino, "*Forum: Arduino,*" [Online]. Available: https://forum.arduino.cc/. [Accessed 05 08 2016].

[25] 3S-Smart Software Solutions GmbH, "*CODESYS Control for Raspberry Pi SL : CODESYS Store,*" 2016. [Online]. Available: http://store.codesys.com/codesys-control-for-raspberry-pi-sl.html. [Available 05 08 2016].

[26] Raspberry Pi Foundation, "*Download: Raspberry Pi,*" [Online]. Available: https://www.raspberrypi.org/downloads/. [Available 05 08 2016].

[27] B. Thomas, "*Tumregelmetoder: Dimensionering av analoga reglersystem,*" in *Modern Reglerteknik*, Stockholm, Liber AB, 2008, pp. 190-192.

[28] FleaPLC, "*Arduino as Raspberry PI's remote IO (Codesys): fleaplc.it,*" 16 August 2014. [Online]. Available: http://www.fleaplc.it/en/tutorials/33-arduino-as-raspberry-pi-s-remote-io-codesys. [Accessed 09 01 2016].

[29] J. O. ,. a. K. Y. C. M. I. H. Nour, "*Fuzzy Logic Control vs. Conventional PID Control of an Inverted Pendulum Robot,*" in *International Conference on Intelligent and Advanced Systems 2007*, Semenyih, Malaysia, 2007.

7 Appendix

7.1 PROGRAM CODE

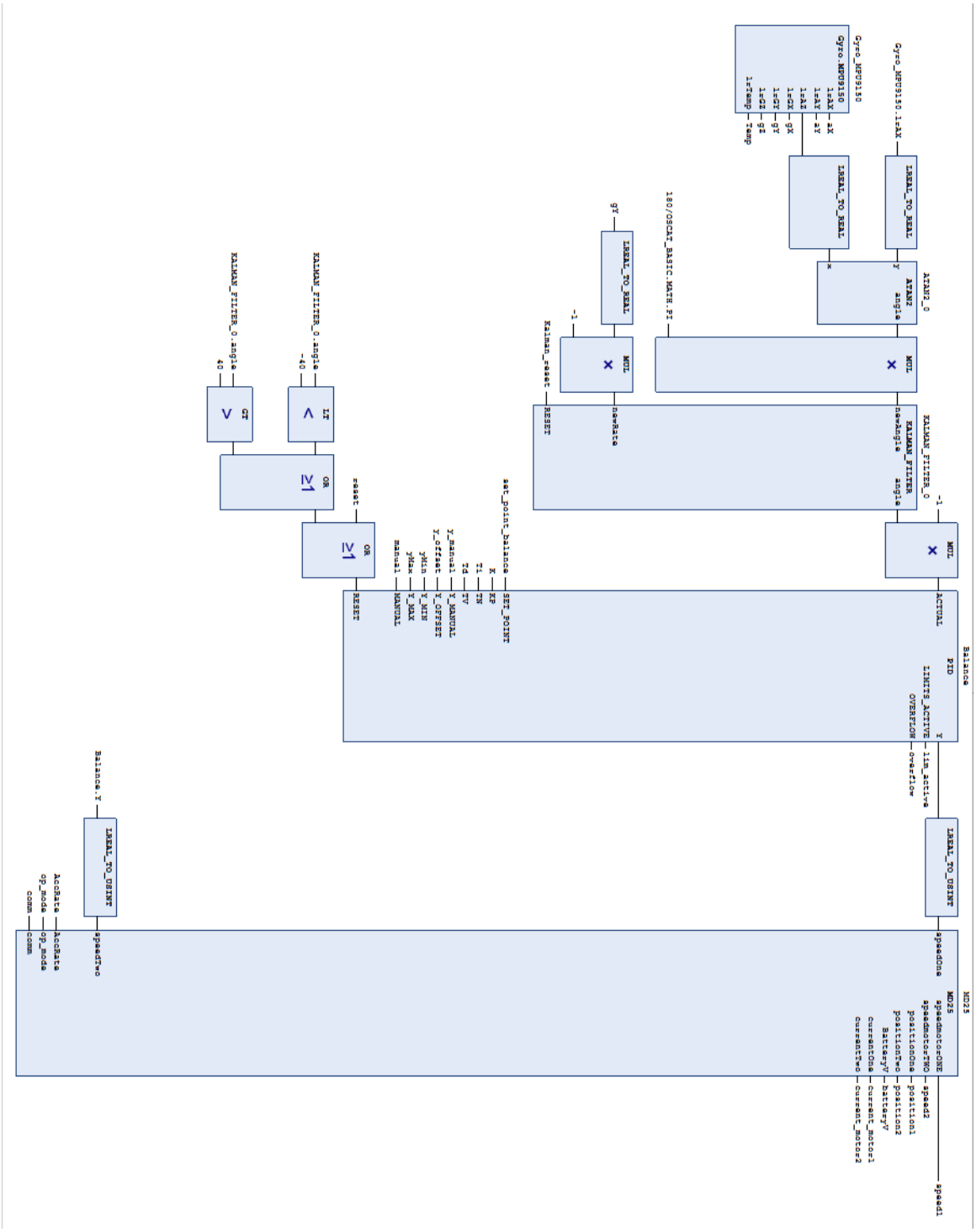
7.1.1 Main program

The IMU outputs the coordinates of the angle-vector, the atan2 function block calculates the angle relative to the gravity vector. A Kalman Filter then estimates the true angle, using the angle from the atan2-function block and the angle-velocity from the IMU. The true angle is then used as the actual value by the PID-controller which calculates the control signal, used as input to the motors through the MD25-motor driver. If the robot falls over (angle > 40 degrees or angle < -40 degrees), the motor stops and the PID-controller is reset.

```
//VARIABLES:
PROGRAM SELF_BALANCING_ROBOT_PRG
VAR
          //Gyro MPU9150
          aX: LREAL;
          aY: LREAL;
          aZ: LREAL;
          gX: LREAL;
          gY: LREAL;
          gZ: LREAL;
          Temp: LREAL;

          //ATAN2
          ATAN2_0: ATAN2;

          //Kalman filter
          KALMAN_FILTER_0: KALMAN_FILTER;

          //PID
          Balance: PID;

          K: REAL := 5.7;
          //Proportionality constant
          Ti: REAL := 0.1;
          //reset time (s)
          Td: REAL := 0.03;
          //rate time, derivate time (s)

          set_point_balance: REAL := -0.6; // calibrated angle in
degrees
          y_manual: REAL := 128; //stop
          y_offset: REAL := 128;
          yMin: REAL := 0;
          yMax: REAL := 255;
          manual: BOOL := FALSE; //TRUE = stop
          reset: BOOL := TRUE; //TRUE = stop and reset int-value

          lim_active: BOOL;
          overflow: BOOL;

          //MD25 Motor Drive
          speed1: INT;
          speed2: INT;
          position1: INT;
```

```
        position2: INT;
        batteryV: LREAL;
        current_motor1: LREAL;
        current_motor2: LREAL;
        AccRate: USINT := 10;
        op_mode: USINT := 0;
        comm: USINT := 49;


        Gyro_calibrated: BOOL;
        Kalman_reset: BOOL;
END_VAR
```

//FUNCTION:


Gyro_MPU9150
IREAL_TO_REAL
ATAN2_0
ATAN2
MUL
180/OSCAT_BASIC.MATH.PI
Kalman_reset
RESET
KALMAN_FILTER_0
KALMAN_FILTER_0.angle
-40
LT
OR
reset
Balance
ACTUAL
PID
LIMITS_ACTIVE
OVERFLOW
SET_POINT
MANUAL
IREAL_TO_DINT
MD25
speedmotorONE
speedmotorTWO
positionOne
positionTwo
BatteryV
currentOne
currentTwo
Balance.Y
speedOne
speedTwo
AccRate
op_mode
comm

7.1.2 ATAN2 Function block

The function block ATAN2 uses the vector, defined by its coordinate, x,y, as input to calculate the vectors angle from a plane.

Several IF- statements determines the sign of the coordinate and thereby the quadrant of the vector. Angle is defined as zero if both x and y is equal to zero.

```
//VARIABLES:

FUNCTION_BLOCK ATAN2 //atan2 (y/x)
VAR_INPUT
          y: LREAL;
          x: LREAL;
END_VAR
VAR_OUTPUT
          angle: LREAL;
END_VAR

//FUNCTION:

IF x > 0 THEN;
          angle := ATAN(y/x);
ELSIF x < 0 THEN
          IF Y >=0 THEN
                    angle := ATAN(y/x)+3.14;
          ELSE
                    angle := ATAN(y/x)-3.14;
          END_IF
ELSIF y > 0 THEN;
          angle := 3.14/2;
ELSIF y < 0 THEN;
          angle := -1*(3.14/2);
ELSE
          angle := 0; //if x=0 and y=0, angle = 0
END_IF

//angle := angle * (180/3.14); //Convert radians to degrees
```

7.1.3 Kalman Filter Function block

```
//VARIABLES:
FUNCTION_BLOCK KALMAN_FILTER // Discrete Kalman filter
VAR_INPUT
          newAngle: REAL;
          newRate: REAL;
          RESET: BOOL;
END_VAR
VAR_OUTPUT
          angle: REAL;

END_VAR
VAR
          uiLoop: UINT;
          Q_angle: REAL := 0.001; //initial test = 0,001
          Q_bias: REAL := 0.003; //initial test = 0,003
          R_measure: REAL := 1;  //initial test =0,03
          d_time: REAL;
          T1: ULINT;
          T2: ULINT;

          bias: REAL;

          P_00: REAL;
          P_01: REAL;
          P_10: REAL;
          P_11: REAL;

          Sk: REAL;

          K_0: REAL;
          K_1: REAL;

          rate: REAL;
          y: REAL;
          P00_temp: REAL;
          P01_temp: REAL;

          //additional variables
          RQ_lim: REAL;
          RQ_lim_negative: REAL;
          timeplot: REAL;

          sigma: REAL := 3;
          K_prev: REAL;

END_VAR

//FUNCTION:
T1 := T2;
SysTimeRtcHighResGet (pTimeStamp := T2);
          d_time := ULINT_TO_REAL (T2-T1)/1000; //Determines the
cycle time in seconds
```

```
uiLoop:=uiLoop+1;
IF (uiLoop>1) THEN //Set delay for testing

// Time Update ("Predict")
// 1) Project the state ahead
rate := newRate - bias;
angle := angle + d_time * rate;

// 2) Project the error covariance ahead
P_00 := P_00 + d_time * (d_time * P_11 - P_01 - P_10 + Q_angle);
P_01 := P_01 - d_time * P_11;
P_10 := P_10 - d_time * P_11;
P_11 := P_11 + Q_bias * d_time;

// Measurement Update ("Correct")
// 1) Compute the Kalman gain K where covariance prediction Sk:
Sk := P_00 + R_measure; // Estimate error

// Kalman gain
K_0 := P_00 / Sk;
K_1 := P_10 / Sk;

// 2) Update estimate with measurement zk (newAngle) where the
innovation y:

y := newAngle - angle;

//Reject measured values outside the innovation covariance limits
(3sigma = 99.7% of mean)

RQ_lim := sigma * SQRT(Sk);
RQ_lim_negative := -1 * RQ_lim;

IF y > RQ_lim OR y < RQ_lim_negative THEN
          K_0 := 0;
          K_1 := 0;

ELSE

//Update estimate
angle := angle + K_0 * y;
bias := bias + K_1 * y;

END_IF

// Eneable reset if the Kalman filter lose tracking
IF RESET = TRUE THEN
          sigma := 100;
ELSE
          sigma := 3;
END_IF

// 3) Update the error covariance
P00_temp := P_00;
P01_temp := P_01;
P_00 := P_00 - K_0 * P00_temp;
```

```
P_01 := P_01 - K_0 * P01_temp;
P_10 := P_10 - K_1 * P00_temp;
P_11 := P_11 - K_1 * P01_temp;

//plotting variables
timeplot := d_time*1000;

END_IF;
```

7.1.4 MD25 Function block

```
//VARIABLES:
//This is a library for the MD25 - Dual 12Volt 2.8Amp H Bridge Motor
Drive
//For further details, see: http://www.robot-
electronics.co.uk/htm/md25i2c.htm
//Speed values are interpreted as USINT (speed/turn value 0-255)
meaning only mode 0 and 2 is avalable

FUNCTION_BLOCK MD25 EXTENDS i2c
VAR_INPUT
          speedOne: USINT;    //send speed to first motor (mode 0),
send speed to both motors (mode 2)
          speedTwo: USINT;    //send speed to second motor (mode 0),
send turn value (mode 2)
          AccRate: USINT;               //Acceleration rate 1-10
          op_mode: USINT;               //Mode of operation (0 or 2)
          comm: USINT;                  //Command register to reset
encoders and speed regulation

          //comm = 32         - Reset encoder registers to zero
          //comm = 48         - Disables automatic speed regulation
          //comm = 49         - Enables automatic speed regulation
(default)
          //comm = 50                   - Disables 2 second timeout
of motors (version 2 onwards only)
          //comm = 51                   - Enables 2 second timeout of
motors when no I2C comms (default) (version 2 onwards only)
          //comm = 160        - 1st in sequence to change I2C address
          //comm = 170        - 2nd in sequence to change I2C address
          //comm = 165        - 3rd in sequence to change I2C address


END_VAR

VAR_OUTPUT
          speedmotorONE: INT;           //Read speed of first motor
          speedmotorTWO: INT;           //Read speed of second motor
          positionOne: INT;             //Read position value encoder
1
          positionTwo: INT;             //Read position value encoder
2
          BatteryV: LREAL;              //Read battery voltage (value
118 = 11.8V)
          currentOne: LREAL;            //Read current through motor
1
          currentTwo: LREAL;            //Read current through motor
2

END_VAR

//FUNCTION:

SUPER^();
```

```
CASE _iState OF
0:
          IF usiAddress = 0 THEN
                    usiAddress := 16#58;          //i2c-adress 0x58
          END_IF

          IF init() THEN
                    _iState := 10;
          END_IF
END_CASE

//AfterReadInput:

//VARIABLES:
METHOD AfterReadInputs : INT
VAR_INPUT
END_VAR
VAR
          r1a: INT;
          r1b: INT;
          r1c: INT;
          r1d: INT;
          r2a: INT;
          r2b: INT;
          r2c: INT;
          r2d: INT;
END_VAR


//FUNCTION:
SUPER^.AfterReadInputs();

IF _iState = 10 THEN
          speedmotorONE := read8(16#00); //Read speed of first motor
          speedmotorTWO := read8(16#01); //Read speed of second
motor

          //Calculate encoder value
          r1a := read8(16#02);
          r1b := read8(16#03);
          r1c := read8(16#04);
          r1d := read8(16#05);
          r2a := read8(16#06);
          r2b := read8(16#07);
          r2c := read8(16#08);
          r2d := read8(16#09);

          positionOne :=
(r1a*256*256*256)+(r1b*256*256)+(r1c*256)+r1d; //Read position value
encoder 1
          positionTwo :=
(r2a*256*256*256)+(r2b*256*256)+(r2c*256)+r2d; //Read position value
encoder 2
```

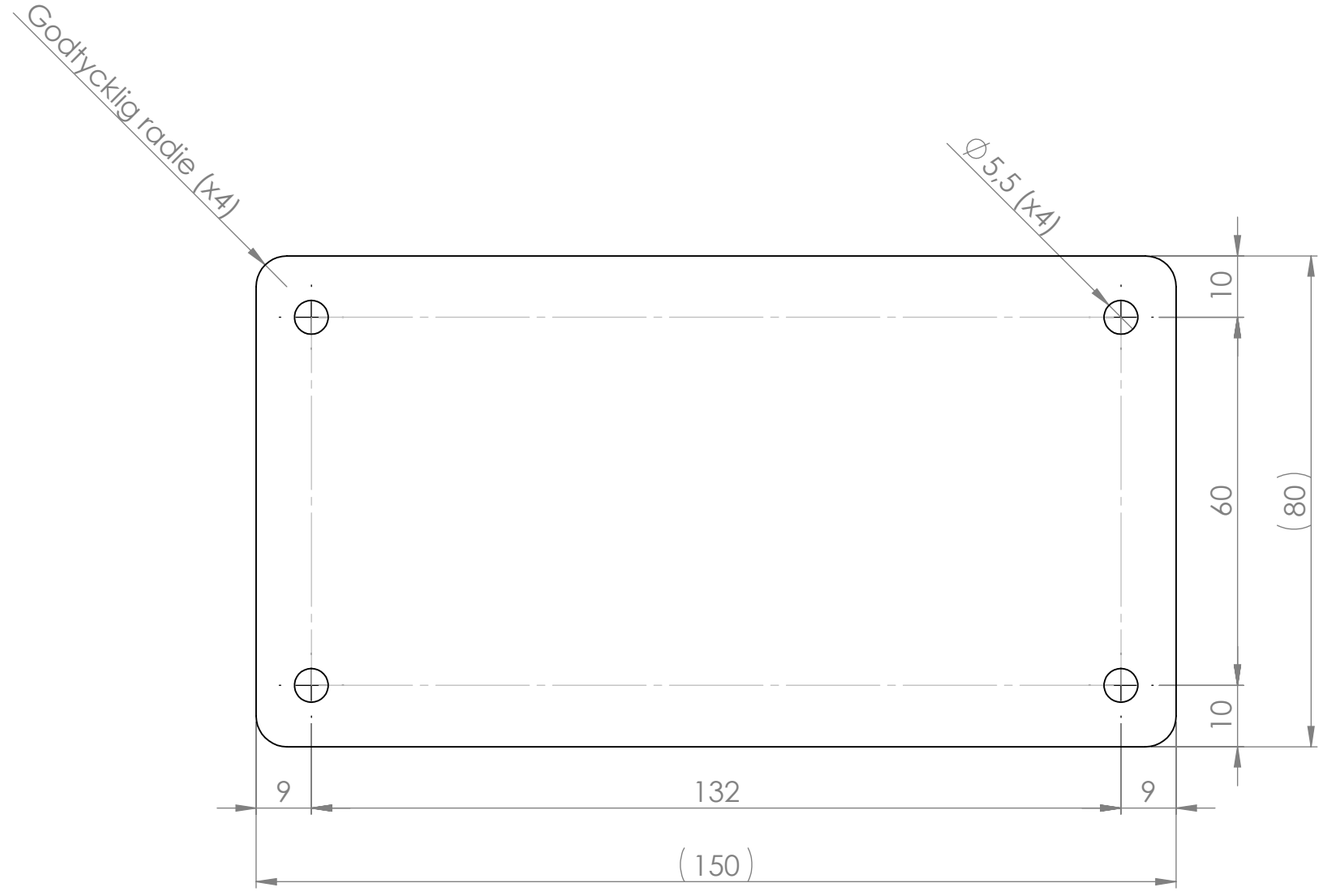
```
          BatteryV := read8(16#0A); //Read battery voltage (value
118 = 11.8V)

          currentOne :=read8(16#0B); //Read current through motor 1
          currentTwo :=read8(16#0C); //Read current through motor 2
END_IF

//BerforeWriteOutput:

//VARIABLES:
METHOD BeforeWriteOutputs : INT
VAR
END_VAR


//FUNCTION:
SUPER^.BeforeWriteOutputs();

IF _iState = 10 THEN
          write8(16#00, speedOne); //speed motor 1
          write8(16#01, speedTwo); //speed motor 2/ turn value
          write8(16#0E, AccRate);       //Acceleration rate
          write8(16#0F, op_mode);       //Mode of operation
          write8(16#10, comm);          //Command register
END_IF
```

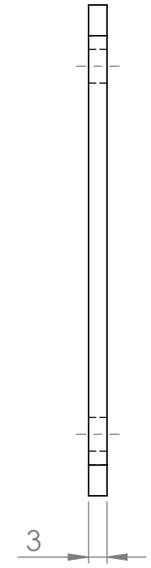
7.1.5 MD25 Device description file

```
<?xml version="1.0" encoding="utf-8"?>
<!--created with CoDeSys 3.0 (http://www.3s-software.com) by
DeviceDescriptionBuilder (3S-Smart Software Solutions GmbH)-->
<DeviceDescription xmlns="http://www.3s-
software.com/schemas/DeviceDescription-1.0.xsd"
xmlns:ts="http://www.3s-software.com/schemas/TargetSettings-0.1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Types namespace="local">
  </Types>
  <Strings namespace="local">
    <Language lang="en">
    </Language>
  </Strings>
<!--  <Files namespace="local">
    <Language lang="en">
      <File fileref="local" identifier="JoystickIcon">
        <LocalFile>Joystick.ico</LocalFile>
      </File>
    </Language>
  </Files>-->
  <Device hideInCatalogue="false">
    <DeviceIdentification>
      <Type>500</Type>
      <Id>FFFF 1111</Id>
      <Version>1.0.0.0</Version>
    </DeviceIdentification>
    <DeviceInfo>
      <Name name="local:ModelName">MD25</Name>
      <Description name="local:DeviceDescription">MD25</Description>
      <Vendor name="local:VendorName">Emil Eriksson</Vendor>
      <OrderNumber>-</OrderNumber>
<!--      <Icon name="local:JoystickIcon">Joystick.ico</Icon> -->
    </DeviceInfo>
    <Connector moduleType="500" interface="Raspberry.I2C"
role="child" explicit="false" connectorId="1" hostpath="-1">
      <InterfaceName name="local:PCI">I²C-Bus</InterfaceName>
      <Slot count="1" allowEmpty="false">
      </Slot>
      <DriverInfo needsBusCycle="false">
        <RequiredLib libname="MD25" vendor="Emil Eriksson"
version="1.0.0.0" identifier="deviceLib">
          <FBInstance basename="$(DeviceName)" fbname="MD25">
            <Initialize methodName="Initialize" />
            <CyclicCall methodname="BeforeWriteOutputs"
task="#buscycletask" whentocall="beforeWriteOutputs" />
            <CyclicCall methodname="AfterReadInputs"
task="#buscycletask" whentocall="afterReadInputs" />
          </FBInstance>
        </RequiredLib>
      </DriverInfo>
      <HostParameterSet>
```

```
        <Parameter ParameterId="1" type="std:INT">
          <Attributes channel="none" download="true"
functional="false" onlineaccess="read" />
          <Default>0</Default>
          <Name name="local:Id393218">I²C address</Name>
          <Description name="local:Id393218.Desc">Address of the
device</Description>
        </Parameter>
<!--        <Parameter ParameterId="2" type="std:UINT">
          <Attributes channel="none" download="true"
functional="false" onlineaccess="read" />
          <Default>50</Default>
          <Name name="local:Id393218">Frequency [Hz]</Name>
          <Description name="local:Id393218.Desc">Number of periods
(HIGH-LOW-cycles) per second</Description>
        </Parameter>-->
<!--        <Parameter ParameterId="3" type="std:USINT">
          <Attributes channel="none" download="true"
functional="false" onlineaccess="read" />
          <Default>4</Default>
          <Name name="local:Id393218">Number of active
outputs</Name>
          <Description name="local:Id393218.Desc">Number of outputs
that are used and need to be controlled</Description>
        </Parameter>-->
      </HostParameterSet>
    </Connector>
 <!-- <Connector connectorId="1000" explicit="false" hostpath="1"
interface="Common.SoftMotion.Servo" moduleType="1024" role="parent"
initialStatusFlag="241">
      <Var max="16"/>
      <HostParameterSet />
    </Connector>-->
  </Device>
</DeviceDescription>
```

7.2 MACHINE DRAWING OF PMMA-PLATE





1	1	Plexi, 3mm	PMMA	150x80		0.04
Det. nr	Ant	Benämning	Material	Dimension	Anm	Vikt

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:		
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL: PMMA	DWG NO. Fästplatta	A3
				WEIGHT:	SCALE:1:1	SHEET 1 OF 1