Mobile Cobot

Digital and Analog Systems Interacting with the Surroundings

Project report

Group 512

AALBORG UNIVERSITY

STUDENT REPORT

Aalborg University
Electronics and IT

Copyright © Aalborg University 2020

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Mobile Cobot

Theme:
Digital and Analog Systems Interacting with the Surroundings

Project Period:
Fall Semester 2020

Project Group:
Group 512

Participant(s):
Klaus Nørholm Frandsen Hammer
Dominic Okumu Vuga
Martin Christensen
Morten Horsted Kristensen
Hyunho Shin
Niels Christian Paredes Dyrberg

Supervisor(s):
Henrik Schiøler

Copies: 1

Page Numbers: 139

Date of Completion:
February 25, 2021

Abstract:

This paper covers the research and process for making a Mobile Cobot which should have the ability to solve the problems and difficulties faced when working with long load objects in the construction industry. Through an analysis and an interview with a master carpenter, the main problem has been determined to be the requirement of two people for lifting long objects. Thereby this has become the main focus of this report and a list of demands has been created, which if complied with would signal that the problem has been solved. The needed functionalities for solving said problem has been determined, thereby making it possible to design a Cobot that replaces the rear person in a two person lift. By implementing an algorithm that stores the previous locations of the user in front, the Cobot is able to follow the path of the user. Throughout a design phase, every design choices and software implementation has been documented and carried out, in order to comply with the set demands. Finally, all of the functionalities have been implemented using a robot platform, a microcontroller and a sensors. At last, the Cobot has been tested according to the demands. The Cobot was not able to comply with all of the listed demands, and the problem was thereby not solved in this project.

The content of this report is not freely available, and publication (with reference) may only be pursued due to agreement

with the authors.

Preface

Aalborg Universitet, February 25, 2021

Klaus Nørholm Frandsen Hammer
<knfh18@student.aau.dk>

Dominic Okumu Vuga
<dvuga18@student.aau.dk>

Martin Christensen
<mchri17@student.aau.dk>

Morten Horsted Kristensen
<mhkr18@student.aau.dk>

Hyunho Shin
<hshin20@student.aau.dk>

Niels Christian Paredes Dyrberg
<ndyrbe18@student.aau.dk>

Contents

Preface **IV**

1 Introduction **2**

2 Problem Analysis **3**
2.1 Working environment regulations . . . 3
2.2 Previous Solutions . . . 5
2.2.1 Multiple person lift . . . 5
2.2.2 Equipment with wheels . . . 6
2.2.3 Automated Cart . . . 7
2.3 Interview . . . 7
2.3.1 Economic cost . . . 8
2.4 Conclusion . . . 8

3 Technical analysis **10**
3.1 Explanation of the idea . . . 10
3.2 Biggest displacement . . . 12
3.3 Angle error margin . . . 13
3.4 Speed limit . . . 14
3.4.1 Simulation . . . 16
3.4.2 Setting the search radius . . . 18
3.4.3 Setting a maximum velocity . . . 19
3.4.4 Finding the max acceleration . . . 19
3.5 Remote controlled Cobot . . . 20
3.6 Conclusion . . . 20

4 List of demands **21**
4.1 Functional demands . . . 21
4.2 Technical demands . . . 22

5 Design of the product **23**
5.1 Algorithm for path following . . . 23
5.1.1 The ring buffer . . . 25
5.1.2 Angle limitation . . . 25
5.1.3 Search radius . . . 26
5.1.4 Propagation of the stored points . . . 26
5.1.5 Finding the steering angle . . . 27
5.1.6 Defining ring buffer size and choosing sample time . . . 29
5.1.7 Implementation of the algorithm . . . 31
5.2 Measurements needed for the algorithm . . . 35
5.2.1 Measurement of vq . . . 35
5.2.2 Measurements of θ_O and omz . . . 37

5.2.3 Measurement of L_O . . . 42
5.3 Motor . . . 42
5.3.1 Steering technique . . . 42
5.3.2 What is the max weight the Cobot can move . . . 43
5.3.3 Weight that can be moved . . . 46
5.3.4 Conclusion of force calculations . . . 46
5.3.5 How to control DC motors . . . 46
5.3.6 Implementing the controller . . . 49
5.4 Detect change in speed . . . 54
5.4.1 Solution 1: Lifting/lowering the object . . . 54
5.4.2 Solution 2: Pull/push the object . . . 54
5.4.3 Solution 3: Remote control . . . 55
5.4.4 Conclusion of speed control . . . 55
5.5 Remote Control . . . 55
5.5.1 Communication protocol . . . 55
5.5.2 Transmitter - remote . . . 57
5.5.3 Receiver - Cobot . . . 57
5.6 Construction of the Cobot . . . 59
5.6.1 The turn-table . . . 59
5.6.2 Mounting of tachometer . . . 60
5.7 Main code . . . 61
5.8 Conclusion on design . . . 62

6 Test of demands **64**
6.1 Functional demands . . . 64
6.2 Technical demands . . . 67

7 Conclusion **74**

8 Discussion **75**
8.1 The minimum and maximum velocity of the motor limits the product . . . 75
8.2 Logic power output of H-bridge gets hot & missing 5V supply to the Teensy . 75
8.3 Voltage drop across the L298 . . . 75
8.4 Incremental encoder failure . . . 75
8.5 Missing speed controller . . . 76
8.6 Bad implementation of gyro . . . 77
8.7 Searching angle limitation . . . 77
8.8 Ineffective use of omz . . . 77
8.9 Compare HW-290 acceleration measurement with motor encoder for better results . . . 77
8.10 Overfilling the algorithm buffer with values . . . 78
8.11 The demands listed for the product are not proper technical demands . . . 78
8.12 The state of the Cobot . . . 78

A Interview with carpenter **80**

B Prove for error margin equation **83**

C Prove for speed limit equations **87**

D Analysis of steering techniques **90**
D.0.1 Differential drive . . . 90
D.0.2 Tricycle drive . . . 90
D.0.3 Independent drive . . . 90

E Wireless Communication **92**
E.0.1 ISM (Industrial, Scientific and Medical) - 2.4 GHz . . . 92
E.0.2 433 MHz radio communication . . . 93
E.0.3 Conclusion on wireless transfer . . . 94

F Bluetooth Low Energy **95**
F.1 BLE Architecture . . . 95
F.1.1 Physical layer . . . 95

G Technologies for angle measurements **99**
G.1 Tachometers/rotary encoders . . . 99
G.2 Potentiometer . . . 99
G.3 Gyroscope . . . 100
G.4 Accelerometer . . . 100
G.5 Digital compass (magnetometer) . . . 100
G.6 Conclusion on angle measurements . . . 100
G.6.1 Measurement of angle θ_C . . . 101
G.6.2 Measurement of angle θ_O . . . 101

H Software **102**
H.1 Gyro . . . 102
H.1.1 UML diagram . . . 102
H.1.2 gyroSetup() . . . 103
H.1.3 readAng() . . . 103
H.1.4 readDAng() . . . 104
H.1.5 resetGyro() . . . 104
H.2 Cobot tachometer . . . 105
H.3 Motor Controller . . . 106
H.4 Main code flowcharts . . . 108
H.4.1 First drive . . . 108
H.4.2 Auto drive . . . 109
H.4.3 Manual drive . . . 110

I Wiring diagram **111**

J Measurement report - Motor variables **112**
J.1 Measurement of the motors angular velocity $\dot{\Theta}_m$ . . . 113
J.2 Measurement of the motors angular acceleration $\ddot{\Theta}_m$ . . . 113
J.3 Measurement of the motors friction coefficient B_m . . . 114

K Measurement report - Code time **117**
K.1 Time measurement of the mentioned functions . . . 117

L Measurement report - Interrupt timing **120**
L.1 Measurement of motor encoder pulse . . . 120
L.2 Measurement of tachometer for object angle . . . 121
L.3 Measurement of Bluetooth . . . 123

M Measurement report - Size of the ring buffer **125**

N Measurement report - Drift in Gyro **127**
N.1 Measurement of gyro drift. . . . 127

O Measurement report - Velocity to PWM **130**
O.1 Measurement of PWM to velocity - Bottom-up . . . 130
O.2 Measurement of PWM to velocity - Top-bottom . . . 132

Chapter 1

Introduction

Since 1977 Denmark have had the Working Environment Act, that dictates rules for work environments [1]. This has protected the physical health of many workers, but the consensus is that the rules have made the job more expensive to carry out, since it often requires more time or multiple workers.
The statistics show that even with the current work environment rules, that there is still room for improvement. There have been reported 44,181 injuries in category 71 in the last five years, whereas category 71 is defined as "Physical overload of the body" [2].
In the same time-span there have also been reported 15,937 violations in group 50 [3], whereas group 50 is defined as violations that will cause "Muscle and skeletal problems". The type of business with the highest representation of group 50 incidents is "Construction and demolition of construction", which hereafter will be referred to as construction.
This means that the high amount of incidents of group 50 violations can be associated with the high amount of injuries regarding physical overload of the body.
If it was possible to make it easier for the employers to comply with the rules, it would be plausible to believe that this would reduce the amount of category 71 injuries. The reason for why the employers are violating the rules so often could be because it makes the work take longer, and thereby increasing the cost.
Because of this, this report will place its main focus on reducing the amount of injuries from physical overload in the construction industry, while accommodating the need for the employer to keep the cost low.
This leads to a initial problem formulation, which will be used in the problem analysis to define one problem to focus on.
The initial problem formulation is as stated in the following:

Is it possible to make a solution that lowers the amount of physical injuries, while maintaining or lowering the cost for the employer, focusing on the construction industry?

Chapter 2

Problem Analysis

The first section will focus on current working environment regulations, and standards in construction industries for lifting and carrying objects in Denmark will be specified. Also the most efficient way for carrying load will be summarised in the aspect of the user's posture. The next section is for showing some of the existing solutions for lifting long and heavy objects. Each pros and cons is going to be introduced, and finally, it will suggest this project's goal. The next section is an interview with a master carpenter, which is questioned about which problems he is faced with when carrying long loads. Finally, the last section covers economic cost when it comes to labor. This is to give an idea whether it is possible to create a solution that can be financially feasible.

2.1 Working environment regulations

Some countries these days have regulations that specify the rules for working environments for people working in industries, such as construction workers. These regulations specify a set of rules for several different areas of the working environments, which could be in the form of a maximum amount of work hours per week and how heavy an object is allowed to be, for the workers to lift it. In this report, the set of rules that will be taken into account will be based on the Danish Working Environment Authority (DWEA), and focusing on the rules for the lifting of heavy objects [4].
According to this set of rules, the maximum amount of kilo a worker is allowed to lift is depending on the way it is lifted. If the object can be held close to the center of the body, it is allowed to lift more, than if the object can only be held away from the body. The different colors on Figure 2.1 shows how many kg a person is allowed to carry or lift in different arm positions.

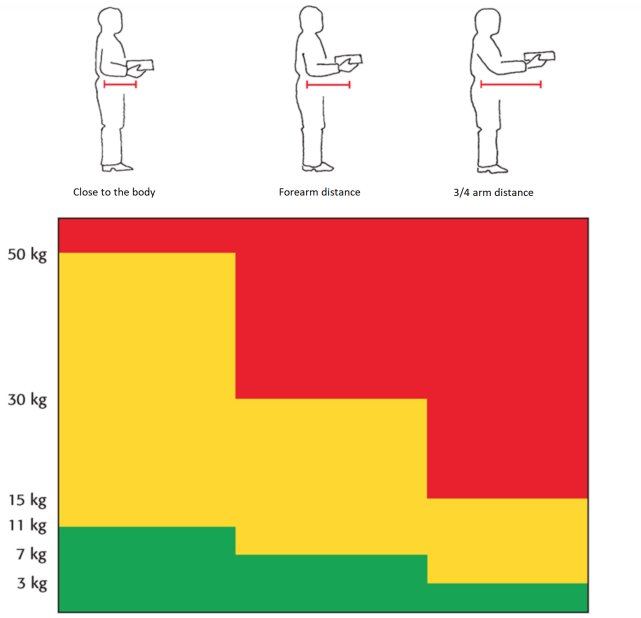


Figure 2.1: Maximum carrying load for a non-moving person in different arm positions, according to DWEA. [5]

The red zone is considered as an unhealthy way to move objects and has to be avoided at all times. Lifting in the green area will usually not cause any health problems because of the light weight. That said, it could still be a problem under the wrong circumstances. For example, if a person makes the same movement for an entire day, it could still cause an injury. Finally there is the yellow zone, which is a bit more complicated than the others. By lifting or carrying an object with a weight of maximum 50 kg, the object must be kept as close as possible to the center of the person's body. Furthermore, the work has to be done under the right conditions. The person doing the lift cannot move more than 2 meters and has to stand on steady ground to avoid slipping or getting out of balance [5]. If it is needed to move the object more than 2 meters, the restrictions are tightened further. The rule for carrying objects for a distance longer than two meters, the amount of kilo allowed from Figure 2.1 has to be divided by 2.5 [5]. This means that for remaining inside the yellow area, which is allowed under the right circumstances, the following maximum weight can be lifted:

- Close to the center of the body: 20 kg
- Forearm distance: 12 kg
- 3/4 arm distance: 6 kg

With all of this in mind, there are some very strict rules regarding the weight a worker is allowed to carry, seeing as many construction workers often make use of materials that are quite heavy. Additionally, as such objects come in various shapes and sizes, if the object is difficult to hold close to the center of the body, the rules become a lot more strict. Furthermore, if the object has to be moved more than two meters, the allowed weight to lift is slightly less.

Due to the need of complying with these rules, the lifting of heavy and long objects, which in particular happens frequently in the construction industry, can be troublesome for workers without hurting their body or wasting too many working hours acquiring help for lifting. For this reason, already existing solutions for helping with carrying heavy and long loads will be researched.

2.2 Previous Solutions

In this section, the focus will be on solutions already being used for lifting of long and heavy objects, whereas some of the most frequently used solutions for this problem will be described.

2.2.1 Multiple person lift

A solution that is often used to carry long and heavy loads, is by having multiple persons carrying it. Here a person placed in the front and back of the object carry it by hand, and by doing so split the weight of the object, while also having better mobility for carrying it around difficult corners. The benefit of adding an extra person to handle a heavy and awkward object does not only reduce the probability for injuries, but increase the amount of weight that is transportable. The extra person can help with lifting, tilting, pulling, pushing and loading the weight [6]. However, the chance of unexpected overload can still occur even when the object is lifted by two or more people, if the communication between them is poor. An example of this could be if they have not agreed on when to lift up or put down the object, or if one of them loses their grip on the object [7].
The center of gravity has an impact on how the weight is balanced between the individuals that are lifting, along with the technique that is used. Using the wrong technique can result in the weight being imperfectly balanced and an individual will experience overload under the lifting. The weight lifted by two should only be 70% of the maximum weight the two persons may lift individually combined [7]. This means two persons having the weight close to their body can lift a maximum of 70 kg, instead of the 50 + 50 kilos they normally can lift individually [8]. The maximum weight two persons can lift when moving a distance of 2 meter is calculated by $20\text{kg} + 20\text{kg} = 40\text{kg}$ for the maximum and $40\text{kg} \cdot 0.7 = 28\text{kg}$ for the 70%. When it comes to heavy and long objects, it is recommended for two people to take position on each end of the object, tilt the object to the side, put an assistant device under it, and then push it instead of lifting [9]. This is illustrated on Figure 2.2.

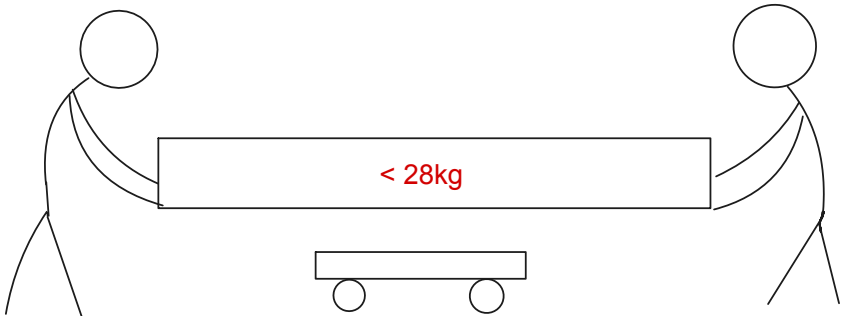


Figure 2.2: Illustration of two persons using a assisting lifting device.

2.2.2 Equipment with wheels

To give a better understanding of the different types of equipment with wheels that already exist for carrying heavy and long loads, a few of them will be looked into, and their strength and weaknesses will be described.

Trolly

A trolly is a tool normally used for carrying long or heavy objects. A picture of this can be seen on Figure 2.3. It has four wheels which sustain the flat plate. There is a bar that users are able to pull or push. Generally, most people pull this, because if a user pushes it, the object could be bounced off from the trolly, unless the object is small and heavy. In the case of a long object, one more person is essential, because each end of the object should be sustained. One should hold, and the other should pull the trolly [11].

Figure 2.3: Trolley [10].

Forklift

Among the tools with wheels, another way of moving long and heavy loads is the forklift, which can be seen on Figure 2.4. Forklifts are rated for loads at a specified maximum weight and a specified forward center of gravity. Normally, it has three or four wheels and one user driving it. For sustaining the long object, there are two 'L' shaped plates. The person then lifts the object onto these two plates, which afterwards can be lifted vertically by the forklift. There is a disadvantage of using the forklift for carrying very long loads, as the object has to be balanced on the two plates - and if the object is flexible, it may bend and touch the floor on each end. The forklift can also be very dangerous to use, as the forks are always pointing in the direction it is driving towards, which means there is a risk of severe damage if driven into another person. The safety is a very important element at the industrial work place, so drivers should have their own license [13].

Figure 2.4: Forklift [12]

Sideloader

Another option is a piece of equipment called a sideloader, which can be seen on Figure 2.5. This uses the same principle as the forklift, which was mentioned in the last paragraph. It has the same two 'L' shaped plates as the forklift, but instead of being mounted on the front, it is mounted on the side of the machine. This gives the advantage that it is able to travel through narrow aisles even with very long objects. Another advantage is that it is safer to use than the forklift, as there is no risk of driving the forks directly into anything. The disadvantage of using a sideloader for lifting long and heavy objects is the size and price. It takes up a lot of space and is more expensive than a regular forklift. [15]

Figure 2.5: Sideloader [14]

2.2.3 Automated Cart

Another solution is to use robots to carry the weight, this way no person needs to lift heavy objects, hereby drastically reducing the possibility of work injuries from carrying heavy loads. Companies like Toyota have already made indoor robots for warehouse use, like the TAE050 HD+ seen on Figure 2.6. These robots work by following tape placed on the ground, and carrying the load along it [17]. The problem with this type of robot is that it's designed for indoor use, and can only follow a piece of tape already placed down beforehand. This means that you can´t use it to lift objects outside, and you have to know the path beforehand and attach the tape on the ground. Since this robot is not specifically designed for carrying long objects, it only fixes the problem with carrying heavy weights, unless modification is made to the robot.

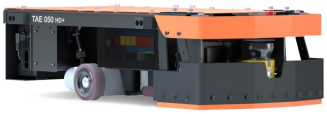


Figure 2.6: TAE050 HD+ Automatic Guided Vehicle. [16]

From what have been researched, different solutions already exist for carrying heavy loads, but for carrying long loads, the solution is to either use a big machine or acquire two persons for the lifting. As there is not a good solution for carrying long loads, the focus of this project will be to create a Cobot (collaborative robot) which can help with the carrying of long objects, without the need for multiple persons or big machinery.
To clarify which problems are most relevant, and which functionalities such a product should be capable of doing, an interview will be made with a master carpenter to gain insight from an experienced worker.

2.3 Interview

To get a better understanding of which possible problems are faced with multiple persons lift, and what functions a Cobot should have, an interview with a master carpenter who owns a carpenter company has been made. He was asked whether lifting long and heavy objects is a problem in his company and his thoughts on this unhealthy issue. The most

important points are listed below where the full interview can be read in Appendix A.

- Often experience carrying heavy and long objects like glued laminated timber (Glulam) and I-beams. The problem is not lifting heavy objects, but mainly long objects.
- He often use two persons to lift these objects, but can be problematic. Especially for small companies where the employees often work alone.
- Glulam and I-beam can cause trouble even for two persons.
- The ideal product should be able to carry heavy and long loads, have the ability to be remote controlled, have interchangeable heads so it can carry different types of objects, and lift the object vertically.
- If a product can fulfill all these needs and replace the second person carrying the object, he would be willing to pay up to 100.000 DKK.

2.3.1 Economic cost

Having multiple employees can be non-beneficial for the employer. Especially if a person has to move an awkward object from A to B, which in itself is a simple job, but could end up being very expensive. This could be because it is impossible for one person to lift the object in a proper way, so that assistance is needed. This will require the employer to pay the salary of two persons for a simple job. An example can be made, where two carpenters have to move an object that takes 10 minutes to move. The price can be calculated from a carpenters average hourly rate in Denmark, which is 206 kr per hour, and is equal to 34 kr for 10 minutes [18]. Assuming the situation for an object to be moved occurs more than once in a day means the price will increase as well. That is why this can end up being very expensive for the employer in the long term. It gets worsened if the assistance is called upon while the second person is occupied or in far reach, and the first person has to wait on the second one to arrive. For this reason, the time to move an object could go from 10 min to 30 min or more, and this increases the amount of money that the employer would have spent on moving the object.
This means that it is possible to create a product that helps save the workers body, and at the same time saves the employer money.

2.4 Conclusion

Throughout the analysis it has been found that lifting heavy and long object can be very harmful. The probability of this depends on the weight, the shape of the object, the way it is lifted, the distance it is carried and the length of the period the lifting has occured. A person is allowed to carry a maximum of 20kg when lifting the weight close to the center of the body, and 6 kg when the weight is lifted at 2/3 of arm distance. Previous solutions for this particular problem is using equipment such as a trolly, forklift, sideloader or an automated cart. These solutions mainly help with the heavy part of the problem, but most of these does not really help with lifting long loads, while those that do, has other disadvantages. Through the interview with the master carpenter it has been confirmed that carrying long

object indeed is a problem at construction sites, and the current solutions are not optimal. Two person lifting does not solve the whole problem, seeing as two persons are only allowed to carry up to 28kg when moving at a distance of over two meters. The employer might end up paying two employees for a simple job because of lack of better solution. For these reasons, it can be concluded that a product is needed at the construction place that is able to lift heavy and long object, can automatically follow, can be remotely controlled, able to lift vertically and it must have an interchangeable head to carry different shapes of loads.
From this a problem formulation has been created, that says the following:
Is it possible to build a Cobot that can automatically follow the person in front, and assist with the carrying of long objects?

Chapter 3

Technical analysis

3.1 Explanation of the idea

Since two persons are required to lift a long object, this product is focusing on solving that problem, by using a Cobot to replace the person carrying in the back. For long loads, it is desired to have a robot to control the rear end of the object, so that one person is able to move and turn around corners, only concentrating on his end of the load. With the back end of the load being mounted on the robot, the idea is that the robot will follow the exact steps that the person in the front takes, making it easier to avoid crashing into something. A sketch of how a user should use the Cobot can be seen on Figure 3.1.

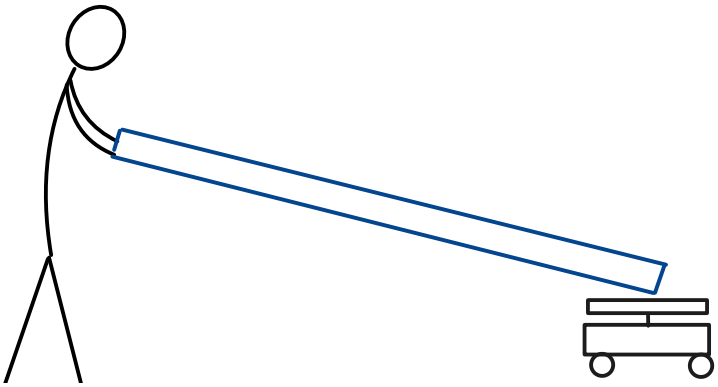
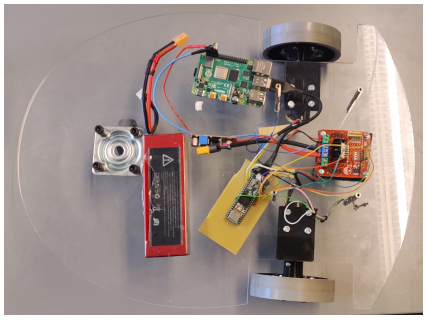
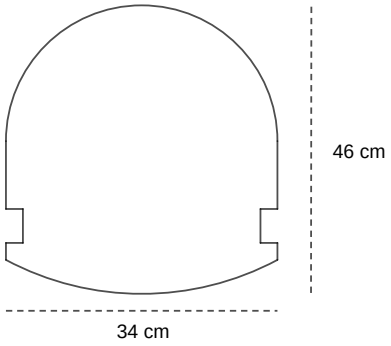


Figure 3.1: A sketch of an object being carried with help of a Cobot.

Aalborg University have provided a robot for use. This robot is a simple differential drive robot, which can be modified to suit the needs for solving this case. It uses a Teensy 3.6 as a microcontroller, a LiPo battery and a WB291111 as the motor controller. A picture of the robot can be seen on Figure 3.2a, and an illustration of its dimensions can be seen on Figure 3.2b The robot will have three wheels to make it move, with two wheels in the back and one in the front end. DC-motors will power the rear wheels so they can move forward and backwards when necessary. The rear wheels are placed on a locked axis so that the robot has to be turned by controlling the speed of the wheels. In the front, a supporting non-powered wheel are placed to ensure the balance of the robot and to simply follow the movement of the rear wheels.



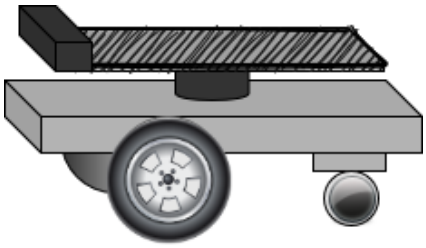
(a) An picture of the robot from the university.



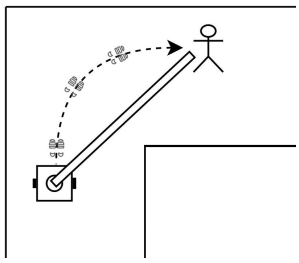
(b) Drawing of the dimensions of the robot

Figure 3.2: Two figures showing the robot

The way it follow the user, is by knowing the exact length of the object being carried, and measuring the angle the object has turned. It can then calculate when and how much it needs to turn, to make sure it follows the same path as the user. An illustration of this idea can be seen on Figure 3.3b. Here, the steps symbolises the path the user took, and the dotted line symbolises the path the robot is going to take. To also give the ability to control the robot manually, it will be possible to control the movements of the robot by a remote control. This ability would be useful in situations where small and heavy loads can be awkward to lift as a person, or when the destination is reached, and it needs to be moved into place. As this robot does not have the ability to measure the angel of an object, it needs to be modified so it can both carry an object, and measure the angle that object have turned. An illustrations of the modifications made can be seen on Figure 3.3.



(a) An illustration of the Cobot



(b) Drawing of the cobot when turning.

Figure 3.3: Two figures explaining the Cobot.

The upper part of the robot is the carrying area, which is a part that will be added to the robot. This area will rotate separately from the rest of the robot while locked to the load, to measure a possible angle between the robot and the load. It will then be possible to follow the exact route of the user by knowing the length of the load, as seen on Figure 3.3b. In order to do this, there should be some sort of user interface, where the length of the object can be set as an input.

Limitations of the prototype

As this will be a prototype to show the feasibility of the project, a few limitations will be made to place focus on the stated problem of replacing one person.

- **Deceleration (braking)** While transporting a heavy object, the power required to break will be higher. This problem will not be addressed in this project, as it is not an important problem in regards to the main idea.
- **Lifting the object vertically** A function wished by the carpenter that was interviewed was the ability to lift the load. As this will not help solve the main problem faced, it has been assessed to not be of a big enough importance to be included.
- **Interchange head** Another desire for the project by the carpenter interviewed was the ability to change the head of the robot, so it can carry different forms of objects. This has also been assessed to not be of importance in regards to the main idea behind the prototype.
- **Limit of carrying ability** As the robot that is used for this project is made of Plexiglas, and is not very durable, the required weight of 28 kg will not be met. At the same time, the ability to carry a heavy object does not change the main functionality of the Robot, so it is not deemed of important in regards to the main idea of the prototype.
- **Functional environment** The constructed prototype is only required to function in the hallway of AAU as the basic concept of the idea can be demonstrate on this hallway. Therefor every demands are considered fulfilled when it is achieved on the AAU hallway.

3.2 Biggest displacement

In this section, the error margin of the Cobots displacement according to where it should be, will be defined.
To set up an equation for the maximum displacement error allowed, the dimensions of the object carried must be defined. For this, the worst case scenario of the expected object length will be taken into account. According to the interviewed carpenter A, the objects that they often carry are I-beams, that are three to four meters long. Hence, the object length that will be used for calculating the error margin will be set as four meters, as the worst case scenario is then taken into account.
The space is highly dependent on the width of the hallway in which it is moved. Since these vary a lot, a hallway from Aalborg University has been measured, and will be used. These have a width of 1.85 m. For this scenario, the width of the object is set to the width of the Cobot.
By using Equation 3.1, the maximal error displacement for the Cobot can be calculated in these criterias. The proof for Equation 3.1 can be read in Appendix B.

$$\begin{aligned} \mathrm{E} &= \frac{(-\mathrm{L_O} - 2 \cdot \mathrm{W_O}) \cdot \sqrt{2} + 4 \cdot \mathrm{W_h}}{4 + 2 \cdot \sqrt{2}} \\ &= \frac{(-4 - 2 \cdot 0.30) \cdot \sqrt{2} + 4 \cdot 1.85}{4 + 2 \cdot \sqrt{2}} \\ &= 0.1310 \end{aligned} \tag{3.1}$$

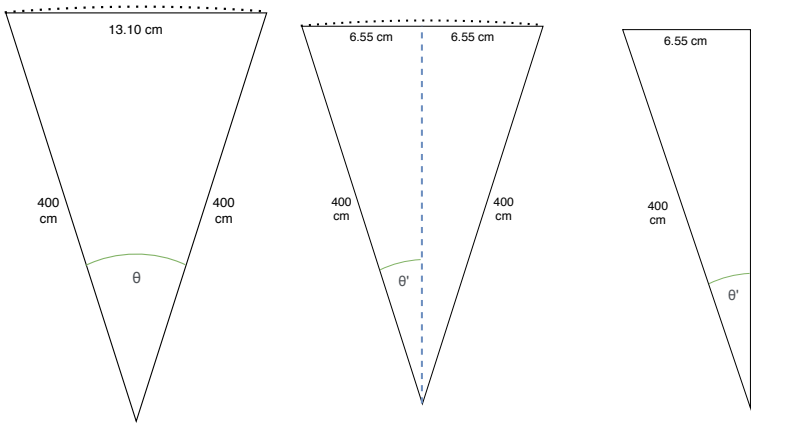
Where:

$\mathrm{L_O}$	The length of the object	[m]
$\mathrm{W_h}$	The width of the hallway	[m]
$\mathrm{W_O}$	The width of the object	[m]
E	The maximal error displacement	[m]

This results in a maximum displacement of 13.10 cm. This displacement will be guaranteed for an object with a length of up to four meters. From this, the maximum angle error allowed between the object and the Cobot can be calculated, which will be done in the next section.

3.3 Angle error margin

As the biggest displacement of the object has been calculated, it can be used to determine when an error in the measured angle is too much for the Cobot to operate successfully. This error margin specifies how much the actual angle can be different from the angle measured by the device. To set up an equation for the maximum angle error allowed, a maximum error in distance as well as the length of the object carried must be defined. The length of the object that will be taken into account is set to four meter, and the maximum displacement on the furthest edge of the object has previously been determined as 13.10 cm. By having these two values set, the maximum angle error, which is the difference between the measured and the actual angle, can be calculated.
This calculation will be made by using trigonometric functions, which can be seen on Figure 3.4. Figure 3.4a shows the object with 13.10 cm and 400 cm. θ is the angle error margin, 13.10 cm is the distance between two points which is made by angle error and 400 cm is the length of the object. As can be seen on Figure 3.4b, the triangle can be split up midways, hereby creating two perpendicular triangles. θ is divided so that θ' is made which can be seen on Figure 3.4c.



(a) The original triangle.

(b) The triangle split in two perpendicular triangles.

(c) One of the two perpendicular triangles.

Figure 3.4: For measuring the maximum angle

The calculations in Equation 3.2 are based on the perpendicular triangle on Figure 3.4c. At first, the angle θ' is calculated by using an arcsin function. Secondly, the angle θ can be calculated by multiplying the angle called θ' by two, resulting in the maximum allowed angle error margin, called θ.

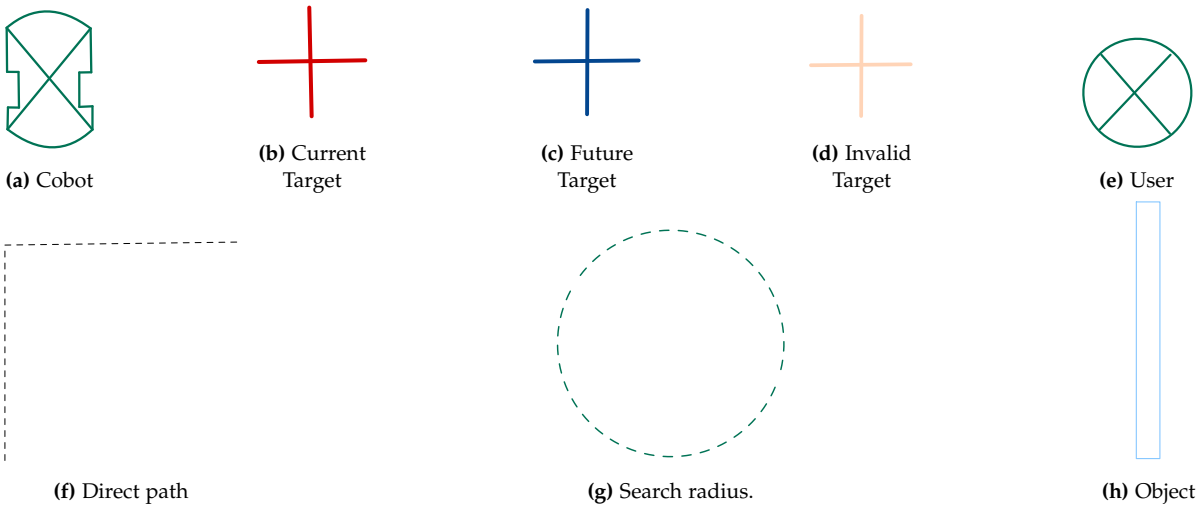
$$\begin{aligned} \frac{6.55}{400} &= \sin(\theta') \\ &\longrightarrow \\ \arcsin(\frac{6.55}{400}) &= \theta' \\ &\longrightarrow \\ \theta &= 2 \cdot \arcsin(\frac{6.55}{400}) \\ \theta &= 1.877° \end{aligned} \tag{3.2}$$

Since θ a value of 1.877°, this is the maximum angle error allowed for it to still be in compliance with the requirement, that an error in the angle measurement can only be allowed to move the other end of the object by 13.10 cm.
The demands for the precision of the Cobot have now been determined. The next part that will be looked into is the requirement to the speed of the robot.

3.4 Speed limit

To be able to decide a speed and an acceleration for the Cobot, it is decided that it would be preferable that the Cobot accommodates its speed and acceleration according to the user, instead of the user having to slow down or speed up according to what the Cobot needs. But it is most important that the Cobot's driving pattern is complies with previous set demands. This means that when the user and the Cobot travels in the same direction, the Cobot should maintain the same speed as the user.

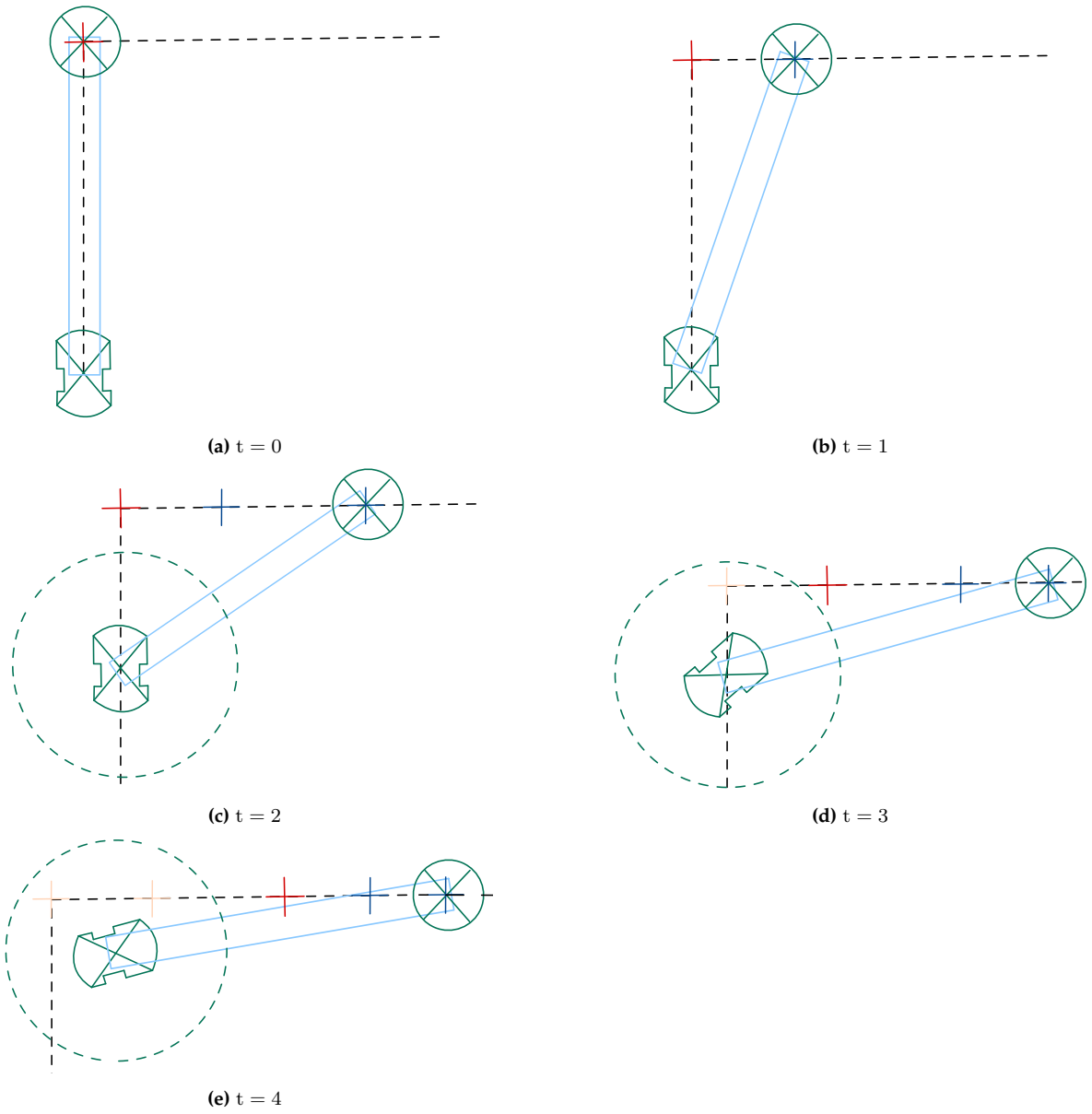
But to choose the maximum speed and maximum acceleration, another scenario has to be analyzed. This is because the speed of the Cobot is changed when the user takes a turn, and the worst case scenario is a 90° turn.
Equations for the speed and acceleration have been made for this scenario, which can be seen in Appendix C. Here it can be seen that as the Cobot approaches the 90° turn, its speed and acceleration also approaches infinity.
Since it would be impossible to reach infinity speed, a third scenario will be considered.
In this scenario, instead of the Cobot always driving in the user's exact path, it will instead have a search radius (r_s). This does so the point towards where the Cobot is aiming will be at least a distance of r_s away from the Cobot. The point that the Cobot is aiming for will be called a target point.
In the Figure series 3.6, it is sketched how it is thought to work. The Figure series 3.5 contains the symbol descriptions.



(a) Cobot **(b)** Current Target **(c)** Future Target **(d)** Invalid Target **(e)** User

(f) Direct path **(g)** Search radius. **(h)** Object

Figure 3.5: Symbols for Figure series 3.6.



(a) t = 0

(b) t = 1

(c) t = 2

(d) t = 3

(e) t = 4

Figure 3.6: Crude sketch-up of path following with a search radius.

As seen on Figure 3.6e, this way to follow the user has the down side that the Cobot will not follow the exact path of the user, but instead an approximation. This will become more clear in the following simulation.

3.4.1 Simulation

A simulation has been made in order to clarify the dynamics of this driving technique. The results are shown on Figure 3.7.
The user goes in a 'v' shape, where the angle in the 'v' is a 90° turn. The user keeps a constant velocity of 1.37 m/s during simulation, the user's velocity is set to 1.37 m/s as it

is the average speed of a person [19]. The users velocity is constant even though it is an unlikely scenario, as the user would probably slow down before the turn and then accelerate up to 1.37 m/s after the turn. It is done to simplify the simulation as it is assumed that it will not affect the maximum positive acceleration and maximum velocity. Therefore the singularity appearing at $t \approx 7.8s$ on the Figures 3.8 and 3.9 plots coming in this section should be ignored.
On Figure 3.7 the user and the Cobot are moving on the ground plane (xy-plane), where the scaling of the axis is 1 m. The user is the black dotted line, and the other coloured lines are simulations of the Cobot with different search radius (s_r) i meters. The Cobot always starts at (1,10), and the user starts at (5,10). The Cobot's initial direction will therefore be towards (5,10), as it will always be the first target point.

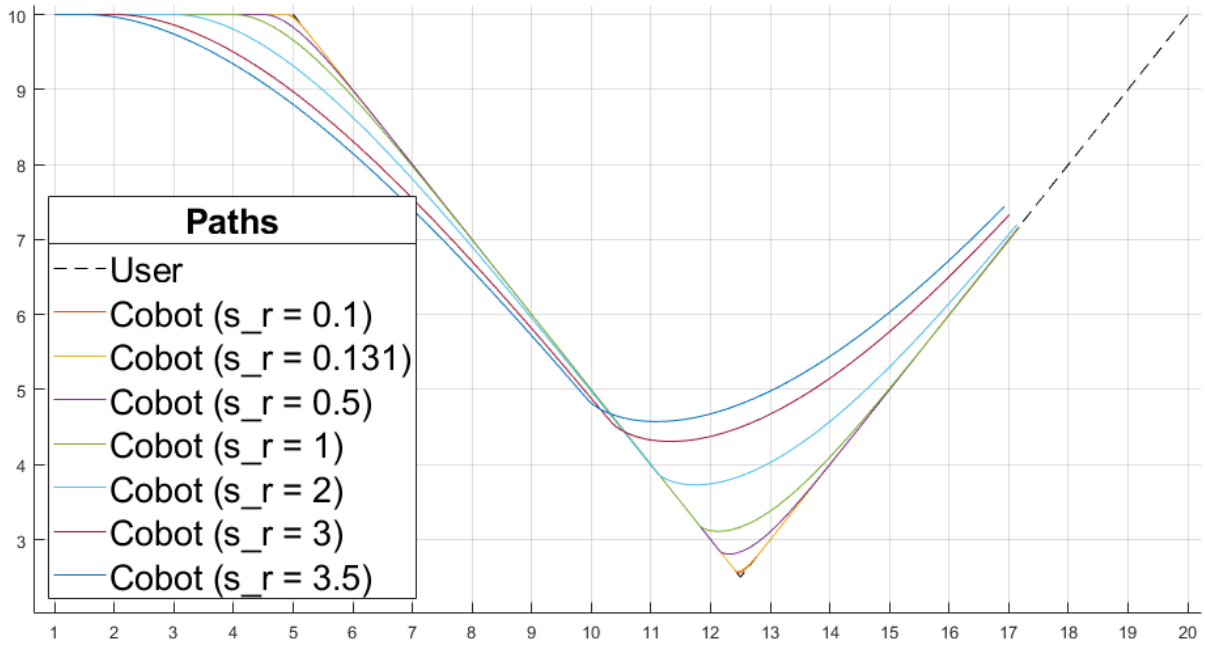
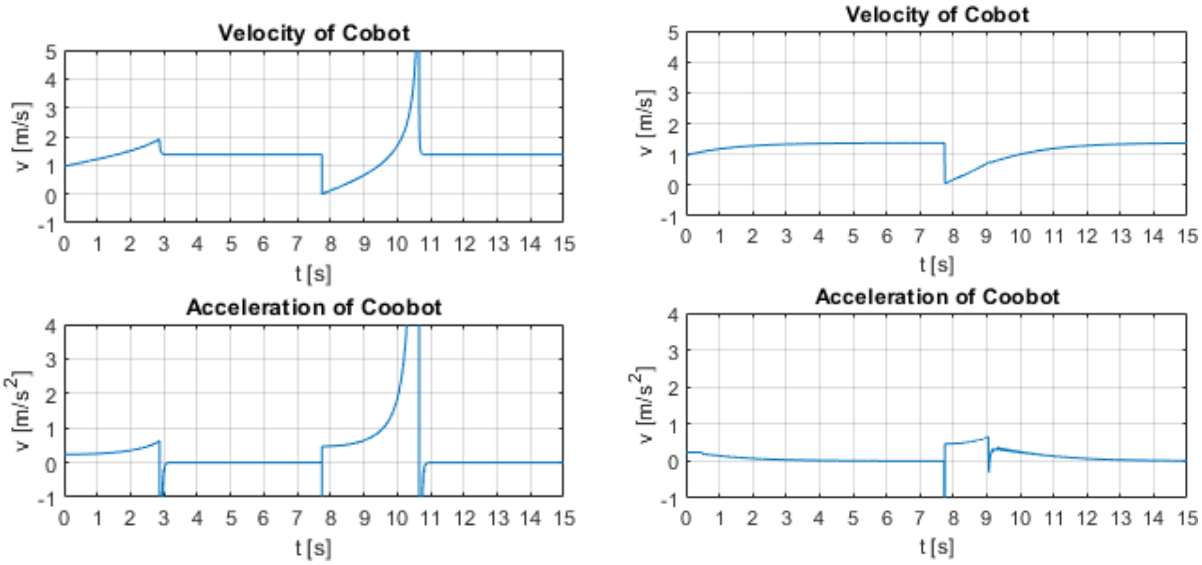


Figure 3.7: Simulated path according to different search radius (s_r).

On the plot it can be seen, that as the search radius gets bigger, the Cobot's deviation from the user's path also increases. It can also be seen that a larger search radius affects the time the Cobot uses to get back on the path of the user. Therefore it would be preferred to have a small search radius to be able to maintain path as the user.
The problem with having a smaller search radius is, that it will approach the singularity at the 90° turn, proven in Appendix C.
To get a clearer view on what the difference in search radius does, the plots for speed and acceleration of the Cobot with different search radius are shown on the Figures 3.8. On Figure 3.8a, the plots for a search radius of 0.1 m can be seen, while the plots for a search radius of 3.5 m can be seen on Figure 3.8b.



(a) s_r = 0.1 m

(b) s_r = 3.5 m

Figure 3.8: Speed and acceleration plots for two different s_r.

As it can be seen on Figure 3.8a, the Cobot is approaching a singularity at $t \approx 10.6$ s, as it had been shown through the calculations in Appendix C. On Figure 3.8b it can be seen that the Cobot never reaches a higher speed than the user when the search radius gets sufficiently large. The large deceleration (≈ 150 m/s^2) at $t \approx 7.8$ s are caused by a simplification in the simulation and should be ignored because of the assumptions made for this simplification.
The simulations have given new insight to the dynamics of the Cobot, but it is not possible from the simulation alone to find a maximum acceleration and a maximum speed for the Cobot.

As the search radius is directly impacting the amount of deviation the Cobot will take from the user's path, a search radius will be defined initially.

3.4.2 Setting the search radius

The search radius is set to maximum 0.131 m, this is done comply with the previous set demand of a maximum displacement of 0.131 m.
On Figure 3.9 the plots for speed and acceleration of the Cobot when s_r = 0.131 m can be seen. Here the singularity of concern at $t \approx 10.6$ s can be seen again.

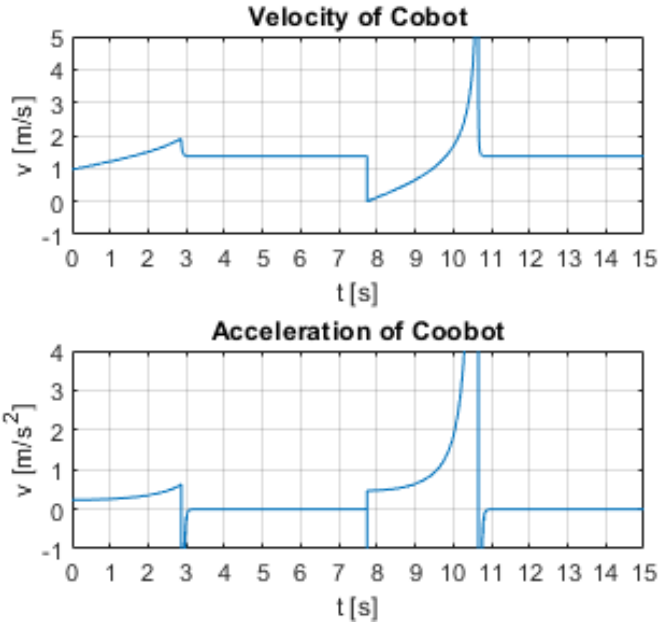


Figure 3.9: Speed and acceleration plots for the Cobot.

The velocity of the Cobot a t=10.6 peaks at ≈ 55 m/s. As this is not considered possible, either the acceleration or the velocity will have to be set first in order to determine the other. It is chosen to set a maximum velocity.

3.4.3 Setting a maximum velocity

As the simulation can not be used for setting the maximum velocity, it will be set using another method.
The maximum velocity will be set to the average velocity of a person walking. This is done so it can keep up with the user when they travel in the same direction. The average speed of a person is 1.37 m/s [19]. Therefore the Cobot's maximum velocity is set to $v_{Cobot,max} = 1.37$ m/s.
The downside of this is that it will force the user to slow down during a turn when $v_{Cobot,max}$ is reached.

3.4.4 Finding the max acceleration

As the $v_{Cobot,max}$ is set to 1.37 m/s, the velocity of the Cobot will flatten out to a constant of 1.37 instead of the high spike. The constant velocity results in a constant acceleration of 0 m/s^2.
Therefore the maximum acceleration will be found at the time where the Cobot reaches the speed of 1.37 m/s.
This happens at t = 9.807 s. At this time the Cobot has an acceleration of 1.328 m/s^2.

From this section it has been determined that the demanded maximum velocity of the Cobot will be $v_{Cobot,max} = 1.37$ m/s. And the maximum acceleration will be $a_{Cobot,max} = 1.328$ m/s. It can also be said that the search radius will be set to 13.1 cm or less, even though it

will first be used in the design chapter 5.
The last thing that is missing in this chapter is investigating the remote control for the Cobot. It is investigated to see how long it should reach.

3.5 Remote controlled Cobot

One feature the carpenter that was interviewed could see as useful, was the ability to remote control the Cobot. It could be useful in situations where the Cobot need to be fine controlled into a specific spot, or if it is used to move wide objects as a carrying mechanic. For this to be possible, the best way is to use a wireless communication system, as it gives the ability to send information over longer distance without the need to directly be where the Cobot is. As the longest object that needs to be carried is 4 meters long, the minimum distance the wireless communication system needs to communicate is 4 meters. But as there might be objects in the way, or other humans, the longer the range is the better. At the same time, because there needs to be a remote controller, it would be optimal to have a long battery life so it does not need to be recharged often. To find a good balance between communication distance and battery life, a minimum distance of 10 meters have been chosen.

3.6 Conclusion

From the technical analysis, the different demands for the Cobot to be able to replace a person when carrying long loads have been set. It has been determined that the Cobot needs to follow the same path as the person in front, to make sure that it does not drive into objects surrounding the path. To determine which path the user in front has taken, it needs to receive an input that consists of the length of the object carried, and the angle the object has been turned. A maximum length of the object has been set to four meters, and the hallway on the university has been chosen as the example location where to test all the demands. This has been done to calculate a maximum inaccuracy that is allowed to be measured between the Cobot and the object that is carried. To make sure that the angle of the object the Cobot measure is useful, the maximum allowed angle error margin has been calculated to be 1.877 °. It has also been decided to only focus on the main functionality of the Cobot, which is the part that allows it to follow the path of a person. Furthermore it sets all other functionalities such as carrying heavy weight and interchangeable head as functions that would be nice to have, but not required to demonstrate the feasibility of the concept. As the Cobot needs to be able to follow the user, it is important that the Cobot is able to keep the same speed as the user. To do this, a minimum velocity and acceleration that the Cobot needs to reach has been found. This has been found by simulating how the Cobot turns around a corner, and from this calculating the maximum acceleration and velocity needed to follow the user. This has been found to be 1.37 m/s and 1.328 $\mathrm{m/s^2}$. Lastly, to give the Cobot the ability to be controlled remotely, it needs to be able to receive a wireless signal from a distance of at least 10 meters away. From all of this, a list of demands has been made, which can be seen in Chapter 4.

Chapter 4

List of demands

This chapter will define the demands that have been made for the functionality of the Cobot. They consist of both technical and functional demands.
All the demands are to help accommodate the demand of replacing the rear person with a Cobot.

4.1 Functional demands

No.	Functional demand
1.1	**The Cobot needs to adjust its speed depending on the speed of the person walking in front.** To make sure that the Cobot does not push or slow down the person walking in front, it should have the ability to vary its speed depending on the speed of the person.
1.2	**The Cobot needs to be able to be controlled remotely.** To make sure it can finely be controlled, and moved into position, it should have the ability to be remote controlled so it is not only controlled by the path of the person in front.
1.3	**There needs to be a physical input on the Cobot that can specify the length of the object carried.** To easily give the ability to change the length of the object carried, there should be a physical input on the robot as to not depend on a pre-programmed length.
1.4	**The Cobot needs to function in the hallway of AAU.** To give a specific environment the Cobot for testing all demands

Table 4.1: Functional demands for Cobot.

4.2 Technical demands

No.	Technical demands	Specification	Source
2.1	The Cobot needs to be able to receive a wireless signal from a distance of at least	10 m	Section 3.5
2.2	The Cobot should be able follow a person carrying an object up to	4 m	Section 3.2
2.3	The Cobot's maximal displacement margin according to its target path should be (with a object length of up to 4 meters)	$\pm$ 13.10 cm	Section 3.2
2.4	The minimum resolution of the angel measured between the Cobot and the object should be	$< 1.877°$	Section 3.3
2.5	The Cobot should be able to drive with a speed up to	1.37 m/s with a total weight of 28 kg.	Section 3.4
2.6	The Cobot should be able to accelerate with an acceleration up to	1.328 $\mathrm{m/s^2}$ with a total weight of 28 kg.	Section 3.4

Table 4.2: The list of technical demands required for the Cobot

Chapter 5

Design of the product

Throughout the analysis, it has been determined what is needed for the Cobot to help workers carrying long objects and how the Cobot should work. The realization of the desired Cobot requires that specific design parameters are being determined. These parameters are designed to accommodate the requirements mentioned in Chapter 4. This chapter is going through all the necessary parts for the Cobot to work as desired by explaining, calculating and implementing everything, from the working of the algorithm to the construction of the Cobot itself. For every section, the implementation of the desired functionality will be explained by using flowcharts and code-snippets. As the code-snippets only shows the relevant part of the code, the complete code for the Cobot can be seen in the Github repository for the project [20].

5.1 Algorithm for path following

According to the demands, the Cobot should be able to follow the path of the user in front that is carrying the front end of an object. In order to do so, there must be implemented an algorithm like described in Section 3.4, that makes it possible to detect the path that the user in front has taken, and steer the Cobot through the same path. For this, an algorithm that stores and navigates a rear end through previous points of the front end has been given by Henrik Schiøler, the supervisor of this project, of which the algorithm had been developed through a former project at Aalborg University. It was originally designed for a trailer attached to a tractor, that needed the trailer to go through the specific points that the tractor had passed through. This would then need then be adjusted to fit the needs of this project, to make the Cobot follow the path of the user.
The algorithm requires five inputs to perform these operations, where each of these will be defined below:

- L_O, which is the length of the object to be carried. This is used for determining the location in the coordinate system that the user is currently at.
- vq, which is the forward velocity of the Cobot. This is used for determining the movement of the Cobot.
- θ_O, which is the angle between the Cobot and the object. This is used for determining which points the user has gone through.
- omz, which is the angular velocity of the object. This is used for adjusting the saved coordinates.
- dT, which is the sampling time. Between each reading, the time passed should be measured in order to properly differentiate the angular measurements, as well as determining the distance moved per sample.

Brief explanation of the working principle

For introducing how the algorithm works, a non-detailed explanation of the working principle will be made. A more detailed explanation of each part of the algorithm will come later in this section. By knowing the length of the object (L_O) and the angle between the direction of the Cobot and the object (θ_O), the points that the user has gone through can be implemented in a coordinate system and saved in a ring buffer. While the Cobot is moving, the forward velocity (vq) combined with the rotational velocity of the object (omz) can be used for adjusting the previous points according to the movement. This can be done by subtracting a translation vector with a length corresponding to the distance the Cobot has moved since the last sample, from all previous points. Afterwards, all the stored points will be rotated, so that the object is always pointing towards the x-axis with the Cobot in (0,0). By doing it like this, a new point can be inserted at the point (L_O,0). From these points, the Cobot should be able to calculate how much to turn in order to reach the desired target point. When the Cobot then needs to locate the next point to travel to, it will go through all of the points in the ring buffer to find the point that is closest to the Cobot within specific search parameters as seen in Figure 5.1, and return the angle between the object and the concerned point. Since the object is always pointing along the x-axis, the angle that the Cobot needs to turn in order to reach the desired point can then be calculated, by subtracting the absolute angle of the object according to the Cobot from the output angle of the algorithm.

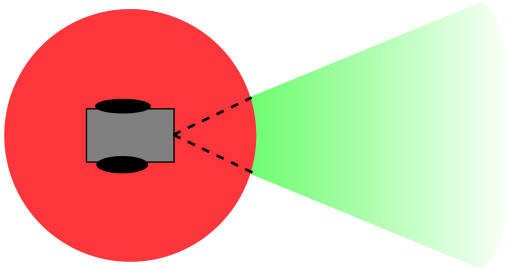


Figure 5.1: The idea of the search angle and radius.

To explain the working principle in detail, the algorithm is split up into different parts, which will be explained in the following. Initially, the functioning of the ring buffer will be described. Afterwards it will be explained how a search radius and an angle limitation have been implemented to improve stability of the navigation process. After these parts have been explained, the process of the algorithm can be described in detail. This is split up into two parts, with the first one being the propagation of the stored points, that adjusts all of the previous points according to the Cobot's movement, and the second part is choosing the right point to steer towards according to the direction of the object. After this angle has been found from the algorithm, the method for calculating the turning angle for the Cobot can be found. As the last part, an appropriate size of the ring buffer as well as the sampling time will be determined.

5.1.1 The ring buffer

In order to adjust the stored coordinates to describe the path of the user, it has to manipulate all of these according to where the Cobot has moved for each new sample. The history of the points that the user has traveled through are saved in a 2-dimensional coordinate system. Each time a new sample is made, all previous coordinates stored will be adjusted and the current coordinates of the user is stored. All of the sampled coordinates of the user will be stored in a ring buffer in terms of x- and y-coordinates, meaning that each coordinate for a specific sample is stored within a buffer, which after a set number of samples will start overwriting the earliest coordinates in the buffer. The ring buffer will have a head value, that determines where in the ring buffer new coordinates should be stored. An illustration of the ring buffer can be seen on Figure 5.2. For example, if the head value is seven, the coordinate will be stored in the seventh position of the buffer. After the data has been stored in position seven, head will be incremented, and the next coordinates will be stored in position zero.

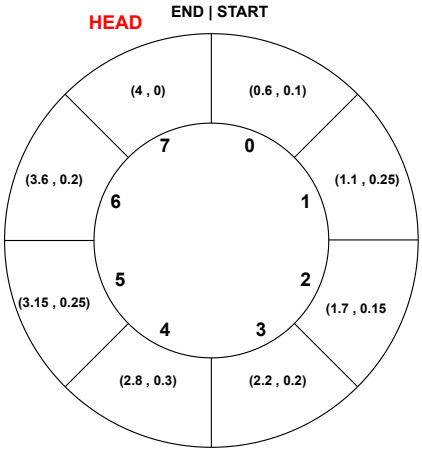


Figure 5.2: Illustration of the ring buffer.

5.1.2 Angle limitation

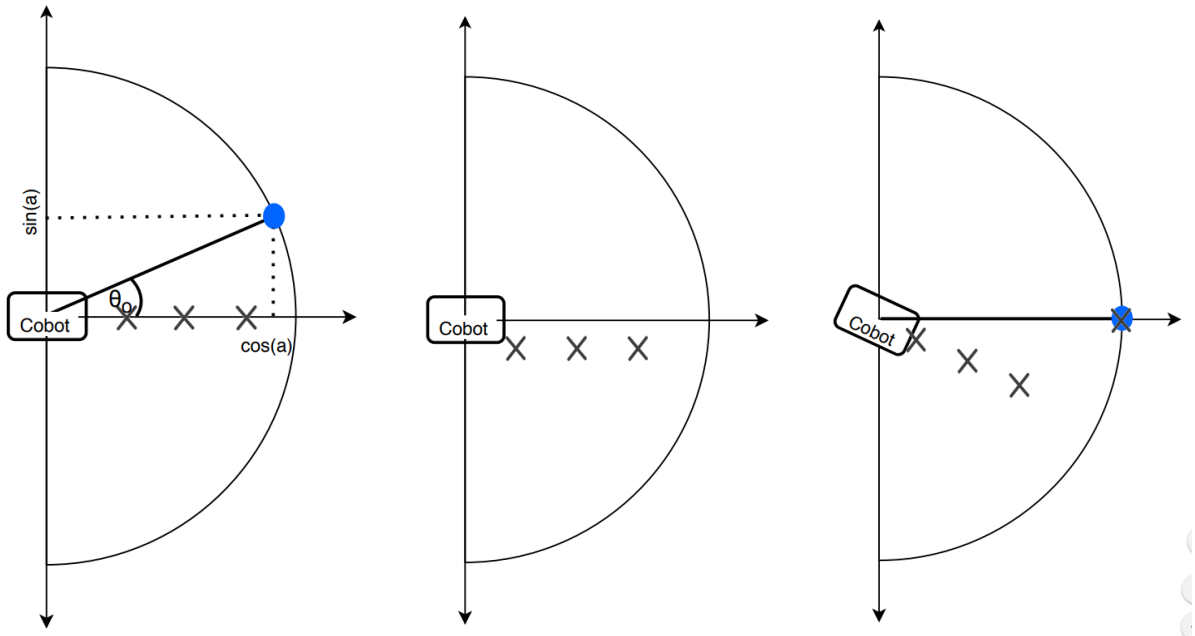
The Cobot needs to have an angle limitation to improve stability by avoiding steering towards points that it has already gone through. This works by limiting the searching field of target point by setting a specific angle, for which it will only search for points within. By having an angle limitation of less than $\pm 90°$ according to the direction of the object, the points behind the Cobot will be rejected, which is desired. By lowering the angle limitation further than 90°, the Cobot will be less prone to steer towards stored points that it has already been through in the case of very sharp turns. Although, by approaching 0°, the field of view can become too narrow, resulting in semi-sharp turns of the user also being rejected. As a search radius will be implemented as well, which will refrain the Cobot from making these sharp turns, the angle limitation is set to $90°$ to only reject points behind the Cobot. As approaching both 90° and $0°$ have disadvantages, a middle ground of 45° for the angle limitation has been chosen in order to accomodate the problem. Although this is set, it is a parameter that should be adjusted by experimental use of the Cobot for a more proper value.

5.1.3 Search radius

The Cobot needs to have a search radius to improve stability by avoiding sharp turns, like it has been described in Section 3.4. In the section, the search radius were determined to be 13.1 cm, which will be used for the algorithm.

5.1.4 Propagation of the stored points

When the Cobot moves, the location of the future points it has to reach will be changed by the distance the user and the Cobot has moved since the last time sample.
The way the algorithm keeps track of where the Cobot is, is by always having the Cobot in a local coordinate system where it is always located in the origin (0,0). Since the Cobot is fixed in this location, the local x- and y-coordinates are always being moved negatively along the x-axis, which is done by translating and rotating previous buffer points. An example of this can be seen on Figure 5.3, whereas each step will be explained one by one.
On Figure 5.3a it is illustrated that the Cobot has been moving straight forward for some time, which means that the user in the front and the Cobot have an angle of zero degrees between them. Now, the user turns some degrees equal to θ_O to the left side, which creates an angle difference between the user and the direction of the Cobot. The user in front is shown as a blue dot, and the line between the blue dot and the Cobot is the object.



(a) Rotation of object, with the crosses symbolising points in the ring buffer.

(b) Points in the ring buffer after translation.

(c) Points in the ring buffer after translation and rotation.

Figure 5.3: Example of the translation and rotation of the ring buffer points.

All of the previously stored coordinates of the user need to be adjusted by the amount the object has moved since the last sample. This is performed by subtracting a translation vector corresponding to the distance and direction it has moved. An illustration of the translation

can be seen on Figure 5.3b The translation vector is calculated as the change in location of the Cobot between the current and the latest sample. This can be calculated by cosine and sine to the angle θ_O, multiplied by the distance travelled since the last sample. The distance moved since the last sample can be found by multiplying vq by dT. This change in position of the Cobot since the last sample will then be subtracted from the previous points stored in the ring buffer. The calculation for the translation vector can be seen in Equation 5.1.

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} \cos(\theta_O) \cdot vq \cdot dT \\ \sin(\theta_O) \cdot vq \cdot dT \end{bmatrix} \tag{5.1}$$

Since it is desired to keep the user and thereby the direction of the object pointing along the x-axis, a rotation of previously measured points will be performed. An illustration of this rotation can be seen in the coordinate system on Figure 5.3c. The rotation is calculated by multiplying a rotation matrix, formed by the angular velocity of the object (omz) multiplied by -dT multiplied with each of the previously translated points. The calculation for the rotation matrix can be seen in Equation 5.2.

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \cos(omz \cdot (-dT)) & -\sin(omz \cdot (-dT)) \\ \sin(omz \cdot (-dT)) & \cos(omz \cdot (-dT)) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.2}$$

The Cobot's local coordinate system is now rotated the same amount of degrees as the object has turned according to the global coordinate system since the last sample, resulting in the object now pointing along the x-axis again.
The propagation of all previous points of the user are performed each time a new sample is made. After this has been done, a new coordinate of (L_O,0) is being inserted at the position of the head in the ring buffer. Since the object is pointing along the x-axis, this point resembles the current position of the user.

5.1.5 Finding the steering angle

Since the ring buffer will keep the previous location points and update these points as the Cobot moves, the functionality of the algorithm is to output the angle the Cobot has to turn according to the object, in order to get to the nearest target point. In order to find the nearest target point, the two additional parameters of a search radius and an angle limitation are used. An example of how the Cobot utilizes these parameters for locating target points can be seen on Figure 5.4. In this example, there is a circle of 13.1 cm symbolising the search radius, as well as two rays with an angle α of 45° symbolising the angle limitation, and the crosses are symbolising the history of the user's path. Here the red crosses show the points that are inside the search radius, and are therefore ignored. The orange crosses show the points that are outside the angle limitation and therefore also ignored. The green crosses show the points that can be considered as potential target points for the Cobot to steer towards. Since it is searched for the green point that are closest to the origin, the green point with a circle around is chosen as the next target point, of which the angle between the point and the x-axis according to the origin will be returned.

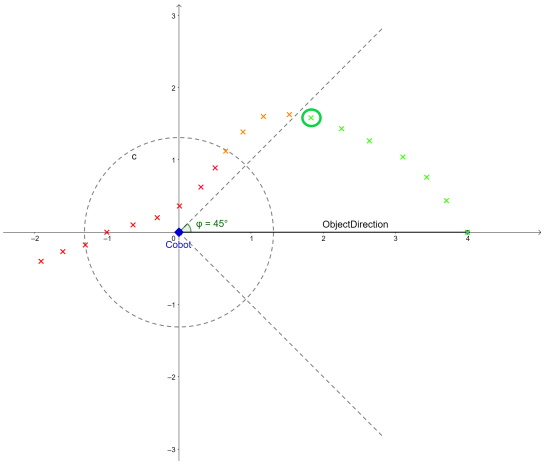


Figure 5.4: Example of locating target points.

When the Cobot is searching for the next point to steer towards, it will choose the point that is closest while also satisfying the parameters. For this example, the next target point will be (2,0), as this is the closest point that is outside the search radius while still inside the angular search field, and the algorithm will output the angle α. As the algorithm is searching for points according to the direction of the object, the output of the algorithm is not the angle that the Cobot should steer towards. Therefore, the angle that the Cobot has to turn to point towards the target point will be calculated.

Calculating the turning angle for the Cobot

As the algorithm calculates and returns the angle between the direction of the object and the point that the Cobot should aim for, a further calculation has to be made in order to determine the angle that the Cobot has to turn. Since the Cobot needs to measure the absolute angle between the Cobot and the direction of the object (θ_O), this angle can be used for calculating the angle that the Cobot has to turn in order to point towards the target point. An illustration of these two angles can be seen on Figure 5.5.

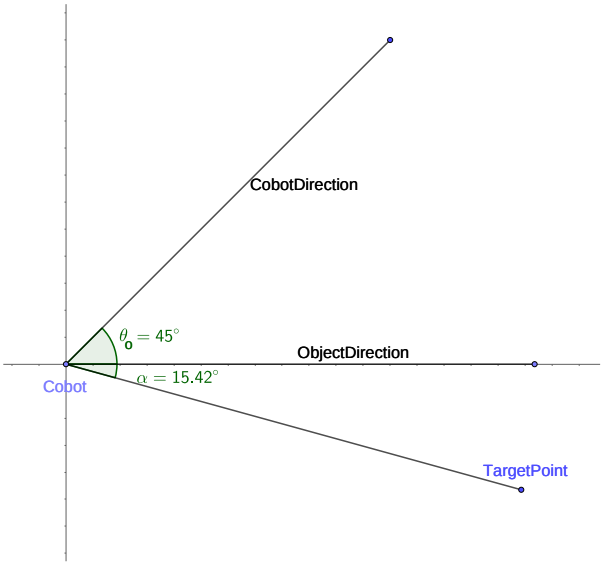


Figure 5.5: The angle from the measuring device (θ_O) and the angle output from the algorithm (α).

The angle that the Cobot needs to turn to reach the desired point is the angle between the Cobot's direction and the TargetPoint. If the TargetPoint is placed below the x-axis, the algorithm will output an angle α of negative 15.42°. In the case where θ_O is equal to 45°, the calculation for determining the angle that the Cobot needs to turn can be seen in Equation 5.3.

$$\begin{aligned} \alpha - \theta_O &= \text{angleToTurn} \\ &\Updownarrow \\ -15.42° - 45° &= -60.42° \end{aligned} \tag{5.3}$$

For the example above, this means that the Cobot has to turn -60.42°, which makes it point towards the desired point. This means that by subtracting θ_O from α, the angle that the Cobot has to turn in order to move towards the desired point can be found.

5.1.6 Defining ring buffer size and choosing sample time

It is essential that the buffer size is large enough, so that the points inside the buffer does not get overwritten before they have been reached. At the same time, the ring buffer must not be too large, as that could cause the algorithm to take longer to compute than the sampling time and thereby not leave enough time for the rest of the code in other functions to run through before the next sample. The sampling time is important as well, as this time defines the time between each sample, which means that lowering the sample time results in more samples, and less time to run the algorithm. If the sampling time is too high, there will be fewer points in the ring buffer, which will make the Cobot's tracking more inaccurate. The sampling time can be too low as well, which will result in more measurements in the ring buffer, requiring a larger ring buffer which thereby increases the computation time. The optimal size of the ring buffer as well as the sampling time can be calculated by using the minimum and maximum speed of the Cobot, as well as the maximum distance desired between two samples. As this

desired distance between samples has to be determined in order to make these calculations, it has been set to two cm. The reason for this is that it provides a large amount of samples, which is assumed will supply the Cobot with sufficient samples to produce an accurate following of the user's path. Initially, the maximum sampling time allowed in order to acquire a distance of two cm between each sample for a four meter object is calculated, using the maximum speed of the Cobot. This maximum speed has been measured to 0.645 m/s, whereas the measurement report for this can be seen in Appendix O. To ensure that this is complied with, the calculations will be performed with a maximum speed of 0.7 m/s. The calculation for the maximum sampling time can be seen in Equation 5.4.

$$\begin{aligned} t_{min} &= \frac{L_O}{v_{Cobot,max}} = \frac{4m}{0.7\frac{m}{s}} = 5.71s \\ dT &= \frac{t_{min}}{\frac{L_O}{d_s}} = \frac{5.71s}{\frac{4m}{0.02m}} = 0.0285s \approx 28ms \end{aligned} \tag{5.4}$$

Where:

L_O	The length of the object	[m]
$v_{Cobot,max}$	The maximum speed of the Cobot	[m/s]
t_{min}	The time it takes to reach four meters at full speed	[s]
d_s	The sampling distance	[m]
dT	The sampling time	[ms]

The calculation gives a sampling time of 28 ms, which will be the value of dT. The buffer size will be calculated using this sampling time, the minimum speed of the Cobot and the time that the remaining functions takes to run. The minimum speed has been measured to 0.11285 m/s, whereas the measurement report for this can be seen in Appendix O. To ensure that this is complied with, the calculations will be performed with a minimum speed of 0.1 m/s. The remaining functions for the algorithm to work has been timed, which can be seen in Appendix K. From this it has been found that the functions without interrupts take 0.16 ms to run. As it is also necessary to take into account the interrupts, they have been timed as well, which can be seen in Appendix L. Here the overall time that all interrupts add to the time the function takes to run once has been found to be 0.04 ms, resulting in a total time of 0.20 ms for all of the remaining functions to run. The calculation for the minimum buffer size can be seen in Equation 5.5.

$$\begin{aligned} t_{max} &= \frac{L_O}{v_{Cobot,min}} = \frac{4m}{0.1\frac{m}{s}} = 40s \\ RBS &= \frac{t_{max}}{dT} = \frac{40s}{0.028s} = 1428,57 \approx 1429 \end{aligned} \tag{5.5}$$

Where:

t_{max}	The time it take to reach four meters at the slowest speed	[s]
$v_{Cobot_{min}}$	The minimum speed of the Cobot	[m/s]
RBS	The ring buffer size	[-]
dT	The sampling time	[s]

To make sure the sampling time is high enough for all functions to run through, the algorithm's time is measured with a ring buffer's size of 1429, the measurement can be read in Appendix M. The time measured is 18 ms, which added to the 0.20 ms of computation time for the remaining functions results in a total processing time of 18.2 ms. It can thereby be concluded that the sampling time of 28 ms is high enough to run all functions and interrupts, while still being sufficient to comply with the desired maximum distance of two cm between each sample.

5.1.7 Implementation of the algorithm

The implementation of the algorithm will be split up in several functions to improve the overview of the code, and explained separately. Firstly, the main function used for calling the algorithm will be explained, followed by the function that has to adjust all of the points in the ring buffer, and lastly the function for finding the angle to steer towards will be described. All of them will be explained with the help of flowcharts and code-snippets.

algorithm()

First and foremost, the function to call the algorithm from the main program will be described. A flowchart of the function can be seen on Figure 5.6.

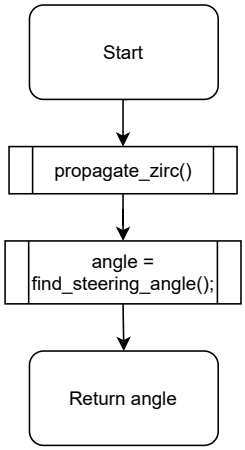


Figure 5.6: Flowchart for the function algorithm().

When the function is called, it will run the function propagate_zirc, which adjusts every point inside the ring buffer, and inserts the value (object_length,0) in the ring buffer at the element corresponding to the value of head, as this value represents the length of the object from the origin throughout the x-axis. Then it increases the value of head so that it points to the next value in the ring buffer for the next time this function is called. This function will be

explained in detail later in this section. After this, it will call the function find_steering_angle, which goes through all the stored points in the ring buffer, and returns the angle between the object and the point that is closest to the origin, while still being inside the search radius and angular search field. This function will be explained in detail later in this section.
The code for the function algorithm() can be seen in Code-snippet 5.1. The function has four inputs: gam is the angle θ_O, omz is the angular velocity of the object, vq is the forward velocity of the Cobot, dT is the sampling time, and L_O is the object length.

```
double algorithm(double gam, double omz, double vq, float dT, float L_O) {
    double angle;
    propagate_zirc(vq, omz, gam, zirc, dT, L_O);
    angle = find_steering_angle(zirc, 3.14 / 5);
    return angle;
}
```

Code 5.1: Code-snippet for the main function of the algorithm.

propagate_zirc()

This is a function made for adjusting the values in the ring buffer. It goes through each of the points in the ring buffer, and adjusts them according to the movement of the Cobot since the last sample. A flowchart of the function can be seen on Figure 5.7.

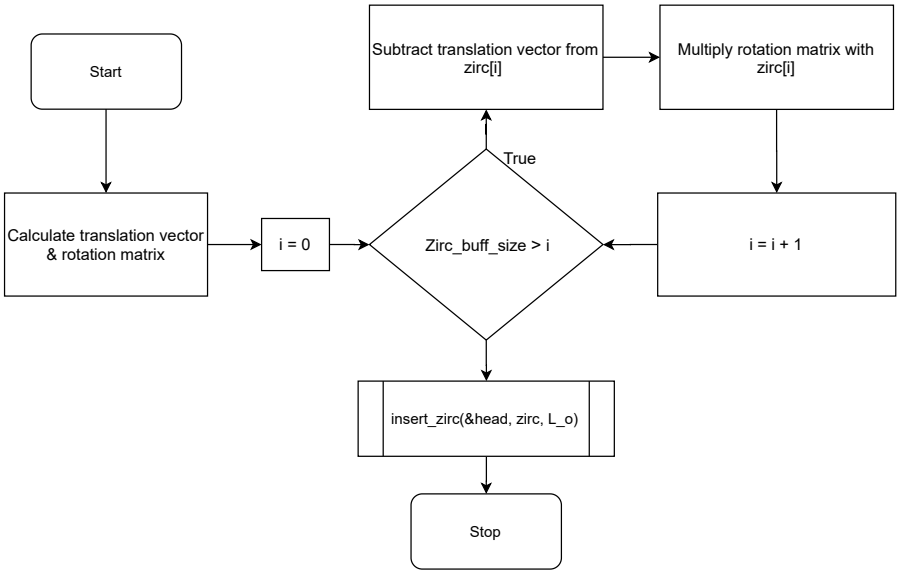


Figure 5.7: Flowchart for the function propagate_zirc().

Initially the translation vector is calculated using the Cobot's speed vq, the sampling time dT and the angle θ_O. Afterwards the rotation matrix is calculated using the angular velocity of the object omz, and the negative sampling time (-dT) for making the points turn the opposite direction of the objects turning direction. The translation vector is now being subtracted from the value corresponding to i in the ring buffer for updating the coordinates according to the movement of the Cobot since the last sample. After the subtraction is performed on this

point in the ring buffer, it is rotated by multiplying the new point by the rotation matrix. After the point has been translated and rotated, i is incremented by one, now pointing to the next point inside the ring buffer. It will increment the value of i until it reaches the length of the ring buffer, meaning that all points inside the ring buffer has been translated and rotated. As the last part, it will insert a new point in the ring buffer at the point in the ring buffer corresponding to the value of head.
The code for the function propagate_zirc() can be seen in Code-snippet 5.2. The function has five inputs: vq is the forward velocity of the Cobot, omz is the angular velocity of the object, gam is the angle θ_O, zirc is the ring buffer, dT is the sampling time, and L_O is the object length.

```
void propagate_zirc(double vq, double omz, double gam, vector *zirc, float dT,
    float L_O) {
    matrix roto;
    vector translate;
    rotmat((-dT) * omz, roto);
    translate[0] = cos(gam) * vq * dT;
    translate[1] = sin(gam) * vq * dT;
    for (int i = 0; i < zirc_buff_size + 1; i++)
    {
        sub(zirc[i], translate, zirc[i]);
        mult(roto, zirc[i], zirc[i]);
    }
    insert_zirc(&head, zirc, L_O);
}
```

Code 5.2: Code-snippet for the propagation of the stored points.

insert_zirc()

This function inserts a new point in the ring buffer corresponding to the value of head, which is the current position in the ring buffer. The code for the function insert_zirc() can be seen in Code-snippet 5.3. The function has two inputs: head is the current position in the ring buffer, zirc is the ring buffer, and L_O is the object length.

```
void insert_zirc(int *head, vector *zirc, float L_O){
  vector v = {L_O, 0.0};
  zirc[*head][0] = v[0];
  zirc[*head][1] = v[1];
  (*head)++;
  *head = *head % L;
}
```

Code 5.3: Code-snippet for inserting a new point in the ring buffer.

find_steering_angle()

This is a function made for locating the next target point and outputting the angle from the object towards the point. The objective is to find the point that is closest to the Cobot, within the specified search radius and angular search field. A flowchart of the function can be seen on Figure 5.8.

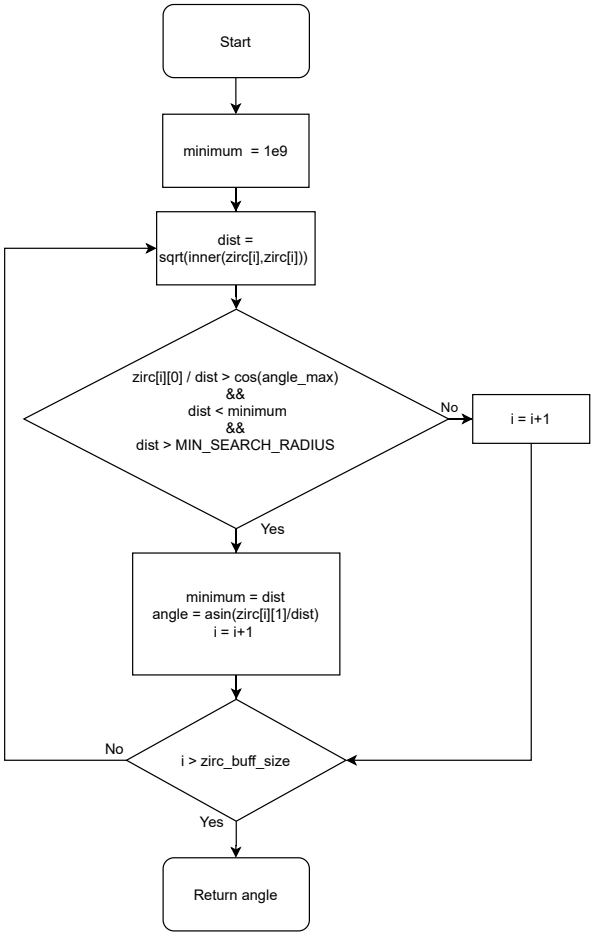


Figure 5.8: Flowchart for the function find_steering_angle().

The function starts by initializing the iteration value *i* to zero. Afterwards it uses a variable called dist to determine the distance from the origin to the point corresponding to *i* in the ring buffer. Then it checks three conditions: if the angle from the origin to the point, which can be calculated by the x-coordinate divided by dist, is within the searching angle, if dist is lower than the minimum value, and if dist is higher than the search radius. If all conditions are satisfied, the value of dist becomes the new value of minimum, and the angle from the origin to the point is saved in the variable called angle. After this procedure, or if all conditions are not satisfied, the value of *i* is incremented, and the procedure is run for the next point in the ring buffer. The value of *i* will be incremented until it reaches the size of the ring buffer, to which it will end since all points in the ring buffer has been inspected. As the last part, it returns the value of angle, which now is the angle of the point closest to the ring buffer within the search angle and radius.
The code for the function find_steering_angle() can be seen in Code-snippet 5.4. The function has two inputs: zirc is the ring buffer and angle_max is the search angle. The value of MIN_SEARCH_RADIUS has been predefined to the value of the desired search radius.

```
double find_steering_angle(vector *zirc, double angle_max){
```

```
    double dist;
    double angle = 0.0;
    double minimum = 1e9;
    for (int i = 0; i < zirc_buff_size; i++){
        dist = sqrt(inner(zirc[i], zirc[i]));
        if (zirc[i][0] / dist > cos(angle_max) && dist < minimum && dist >
    MIN_SEARCH_RADIUS){
            minimum = dist;
            angle = asin(zirc[i][1] / dist);
        }
    }
    return angle;
}
```

Code 5.4: Code-snippet for finding the steering angle.

As the working principle of the algorithm has now been explained, it will now be researched how to get the needed measurements for the algorithm to work. These include the two angular measurements of the angle between the Cobot and the object (θ_O) as well as the rotational velocity of the object (omz), the forward velocity of the Cobot vq and lastly the length of the object (L_O).

5.2 Measurements needed for the algorithm

In this section it will be documented how the four measurements needed for the algorithm will be measured. The first measurement that will be described is vq, followed by a section concerning the two angular measurements, θ_O and omz. Final measurement needed is L_O.

5.2.1 Measurement of vq

When the Cobot moves, it is essential to know its velocity (vq) to calculate how far the Cobot has been moving since the last sample. The motors for this project are installed with encoders, which means that it is possible to keep track of the motors and thereby know the amount the wheels have been turning. The encoders on the motors are incremental (explained in appendix G) and include both an A and B encoder output, by which it is possible to count the raising or falling edge each period, as seen on Figure 5.9.

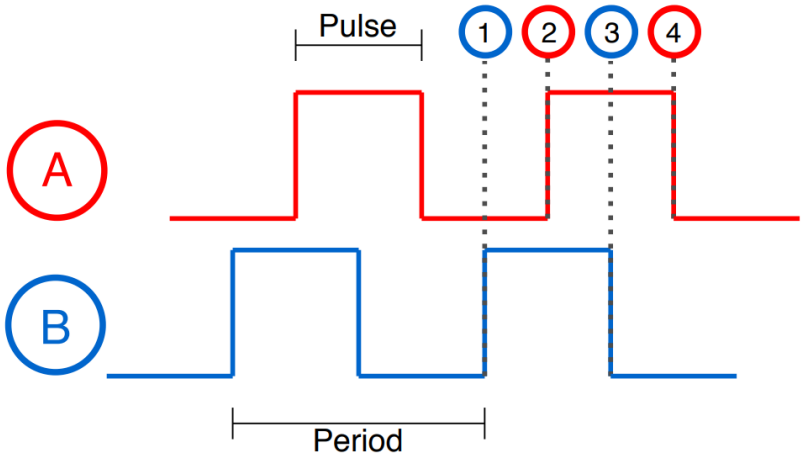


Figure 5.9: Square wave explanation.

From the encoder both of the pulses generated will be used. By using both it is possible to determine the direction of the object. This is possible as one pulse is 90° delayed in phase according to the other. One of the motor encoder outputs for each wheel will be used to count the ppr. According to the web page where the motor can be bought, the encoder can provide 3,591 counts per revolution (cpr) because of the gearing [21]. That means that when the wheel has turned a whole turn or 360 degrees, the encoder has reached 3,591 counts.

To calculate the velocity vq in the unit $\mathrm{m/s}$, it should first be known how many revolutions have been made in one second. By dividing the counted pulses between samples with the total counts for one revolution (3,591) and again divided with the sampling time, the revolutions per second (RPS) is found as seen in Equation 5.6.

$$\mathrm{RPS} = \frac{\left(\frac{\mathrm{ppr_{count}}}{\mathrm{ppr}}\right)}{\mathrm{dT}} \tag{5.6}$$

Where:

RPS	Revolutions per seconds	[-]
$\mathrm{ppr_{count}}$	Pulses counted in one sample	[-]
ppr	Pulses per revolution (3591)	[-]

The measured RPS is only for one motor encoder, but in the case where the Cobot moves straight forward, the RPS on the left and the right wheel is the same. For calculating the velocity on each wheel, it can be found by the circumference of the wheel multiplied with the RPS as seen in Equation 5.7.

$$\mathrm{v_{L/R}} = \mathrm{r_w} \cdot 2 \cdot \pi \cdot \mathrm{RPS} \tag{5.7}$$

Where:

r_w	The radius of the wheel	[m]
$v_{l/R}$	The velocity of left/right wheel	[m/s]

To find the total forward velocity of the Cobot, vq, the velocity of the left and right wheel must be added together and divided by 2 as seen in Equation 5.8.

$$vq = \frac{v_L + v_R}{2} \tag{5.8}$$

Implementation of measuring vq

The implementation of vq can take place now that the calculations above have been made. A flowchart of the function getRPSsincelast() can be seen on Figure 5.10 Firstly, the amount of encoder pulses on each wheel is measured since the last sample and then calculated to RPS.
The RPS is then used in to return the velocity of the left and right wheel. Lastly, the vq is calculated in Code-snippet 5.5.

```
float get_vq(){ // Calculates the forward velocity of the
    motors from vr and vl
    float
 vr = rightencoder.getspeed(wheel_radius, motor_encoder_ppr);
    float
 vl = leftencoder.getspeed(wheel_radius, motor_encoder_ppr);
    float vq = (vr+vl)/2;
    return vq;}
```

Code 5.5: The output vq

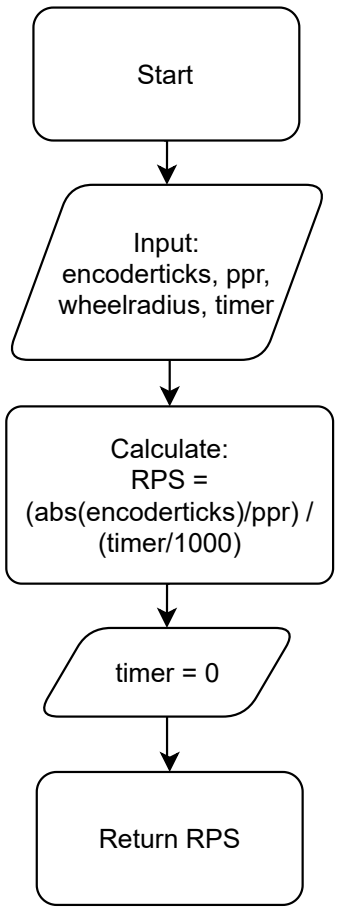
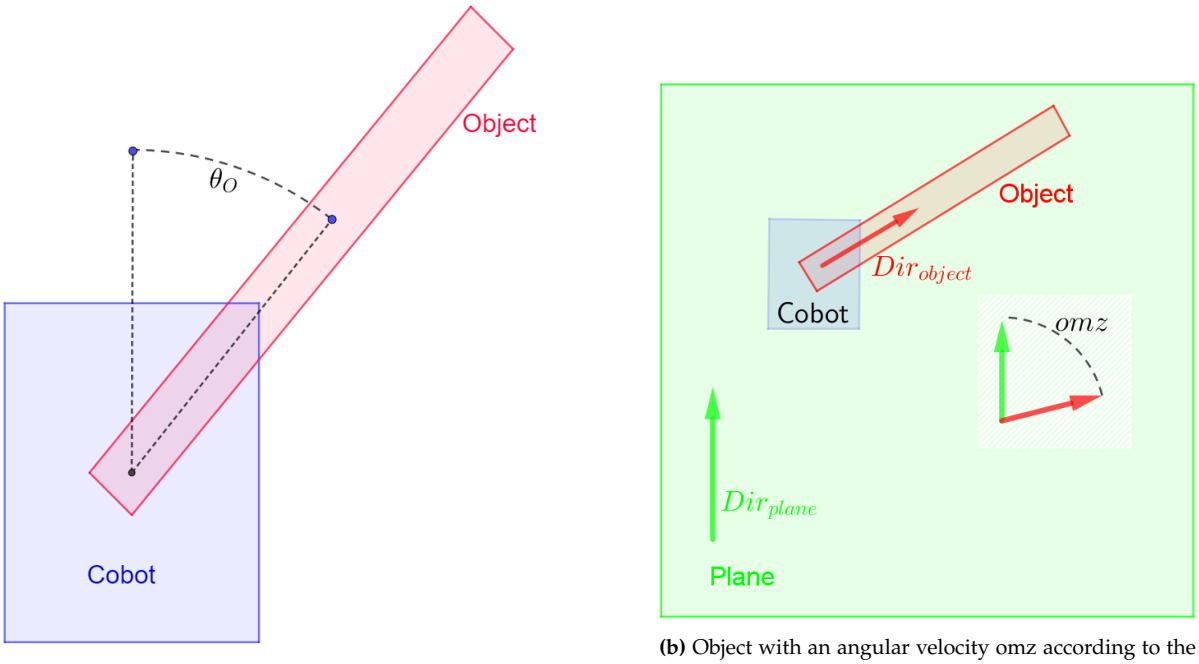


Figure 5.10: Flowchart getRPSsincelast()

Now that the explanation and implementations of the forward velocity is done the angle measurements can be explained and implemented in the following.

5.2.2 Measurements of θ_O and omz

To be able to calculate the Cobot's trajectory, the angle between the Cobot and the object (θ_O) and the angular velocity of the object (omz) needs to be measured. On Figure 5.11a, a sketch of what the angle θ_O is able to be seen. While a sketch of the angular velocity omz is, can be seen on Figure 5.11b.



(a) Object with an angle θ_O according to the Cobot.

(b) Object with an angular velocity omz according to the plane.

Figure 5.11: The two angular measurements needed for the algorithm

How θ_O is measured

In order to figure out which technology is best suited for measuring θ_O
an analysis has been made. This analysis can be read in Appendix G. Here it was concluded that an incremental encoder would be the best suited method for measuring θ_O. The chosen encoder is one from Sparkfun with 1024 pulses per revolution (PPR) [22]. The 1024 PPR gives a precision margin of 0.35°. The encoder has three output signals, two (output A and B) with the same functionality as explained in Section 5.2.1 and has 1024 PPR. The last output (output z) is used as a reference point and has a PPR of 1.

The theory behind how an incremental encoder works is explained in Section 5.2.1.
As the objective here is to determine the exact angle, each pulse should trigger the microcontroller into incrementing or decrementing a counter to keep track of the pulses. This counter can afterwards be used for calculating θ_O. The calculation of θ_O is done by using Equation 5.9.

$$\theta_O = \frac{2 \cdot \pi}{\text{PPR}} \cdot \text{counter} \tag{5.9}$$

Where:

θ_O	Angle between the Cobot and the object	[rad]
PPR	Number of pulses per revolution	[-]
counter	Variable keeping track of the pulses	[-]

The implementation of θ_O

The function for the measurement of angle θ_O consists of two subfunctions: pulseA(), getObjectAngle(). The detailed explanation of all functions concerning the encoder can be seen in Appendix H.2. The main function of keeping track of the counter is the function pulseA(). This is an interrupt service routine and triggers for every rising edge of encoder's output A. When the function is triggered the direction of the angle is found by checking if output B of the encoder is also HIGH. Hereafter, the amount of counts increases or decreases accordingly. The flowchart for this function can be seen on Figure 5.12.
The function getObjectAngle() is the function for calculating the angle θ_O. This function can be seen in Code-snippet 5.6. Here θ_O is the variable objectAngle and is returned to the function call.

```
float getObjectAngle() {
    float objectAngle;
    objectAngle = (2*M_PI / 1024.0) *
counter;
    return objectAngle;
}
```

Code 5.6: Definition of getObjectAngle().

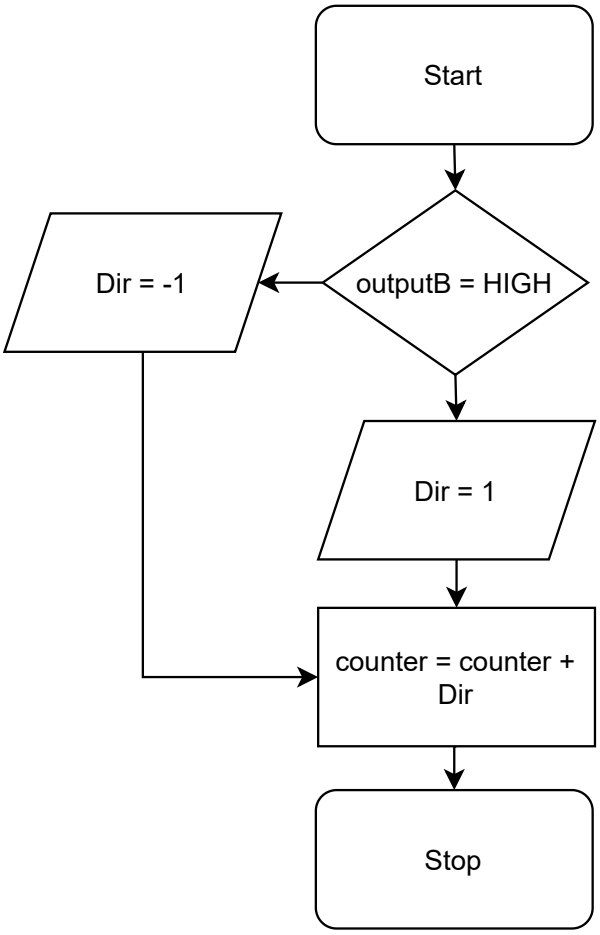


Figure 5.12: Flowchart of pulseA()

Now that the measurement of θ_O has been established, the measurement of omz will be explained in the next section.

How omz is measured

As it will be described later in Section 5.6, all of the electronics are placed on the Cobot's bottom plate, while the object is placed on an turn-table on top. Because of this, the wires (between the micro-controller and measuring device) would have to run from the bottom plate up to the turn-table in order to measure omz directly. Since this would add constant stress to the wires as the Cobot and the turn-table turn independently, a direct measurement of omz has been deemed undesirable.
Instead, a calculation of omz will be done using a measurement of the change in the Cobot's direction according to the ground plane ($\Delta\theta_C$), combined with a change in θ_O ($\Delta\theta_O$).
In order to get $\Delta\theta_C$, a measurement of θ_C has to be made. In the next section it will described how θ_C is measured.
Measurement of θ_C

A sketch of the angle θ_C can be seen on Figure 5.13. From this it can be seen that the Cobot is moving and rotating on the plane, whereby it is the rotation of the Cobot that will be measured.

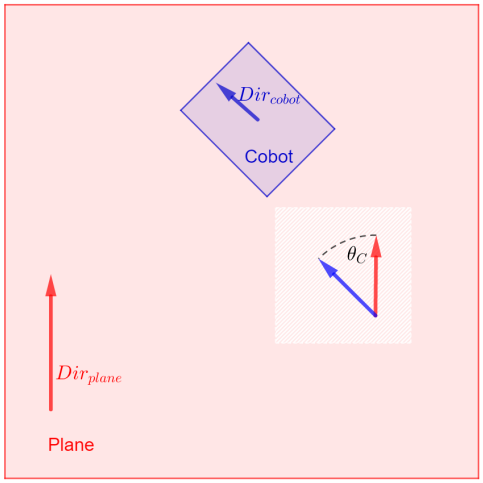


Figure 5.13: Cobot with an angle θ_C to the plane.

One way of measuring the rotation is by keeping track of the wheels. But for stability reasons it has been decided to keep track of θ_C using another method. This is done as the wheels might slip, and thereby resulting in measurements that are inaccurate.
Through an analysis in Appendix G, it has been chosen that θ_C should be measured by the module GY-87 as it has an on-board gyroscope and accelerometer [23]. It was also picked as it abilities a measurement of the Cobot's rotation around itself according to the ground plane even-though the Cobot is translating on the ground.
It was also noted in the analysis that gyroscopes and accelerometers as the GY-87 has a tendency to drift. Because of this a measurement report of the drift for the GY-87 has been made, which can be read in Appendix N. The results of this measurement report shows that after 20 seconds, the drift around its own z-axis has a constant speed of 0.41 deg/min. It is assumed that this drifting is sufficiently low and thereby only contribute to instability by a negligible extent. This is assumed as it was calculated in Section 5.1.6, that the Cobot will take a maximum of 20 seconds before overwriting the the oldest coordinates in the ring buffer, which results in a maximum of 0.136° inaccuracy for a single point.
In order to calculate omz, $\Delta\theta_C$ has to be determined, which will be done in the next section.
Measurement of $\Delta\theta_C$
To communicate with the GY-87 module, a class called "Gyro" has been made. A detailed documentation about the class can be read in Appendix H.1. The Gyro class is mainly using methods from the "i2cdevlib", and is made in order to simplify the process of communication [24].
On a side note it can be said that after the implementation it had been realized that the gyro library was made in an less that optimal way. More on this can be read in the section 8.6 in the Discussion.

Besides the two public methods Gyro::resetGyro and Gyro::gyroSetup, there is a method used for reading the change in angle since last call, called Gyro::readDAng. The flowchart for this method can be seen on Figure 5.14.
On this flowchart it can be seen that old_ypr is set to the current ypr, where ypr stores the absolute values for the module's turn around it's own axes (yaw, pitch and roll).
Afterwards ypr is updated by calling a private method Gyro::readAng().
At last dAng is then calculated as the difference between the two ypr measurements (ypr and old_ypr). As dAng stores the change of yaw, pitch and roll, $\Delta\theta_C$ is equal to dAng[0].

Now where the change in θ_C has been determined from the measurement from the Gyro, the only variable missing to calculate omz is $\Delta\theta_O$ which was measured by the encoder. Therefore the calculation of $\Delta\theta_O$ will be explained in the next section.

Calculation of $\Delta\theta_O$

To find the change in θ_O ($\Delta\theta_O$), the function getDiffObjectAng() has been made. This function can be seen in Code-snippet 5.7. Here old_objectAngle is set to angle, where angle is θ_O. And then angle is updated by getObjectAngle(). At last diff_objectAngle is calculated as the difference between angle and old_objectAngle.

```
float
 getDiffObjectAng(float *objectAngle, float *old_objectAngle){

    float diff_objectAngle;

    *old_objectAngle = *objectAngle;
    *objectAngle = getObjectAngle();
    diff_objectAngle = *objectAngle - *old_objectAngle;

    return diff_objectAngle;
}
```

Code 5.7: Definition of getDiffObjectAng().

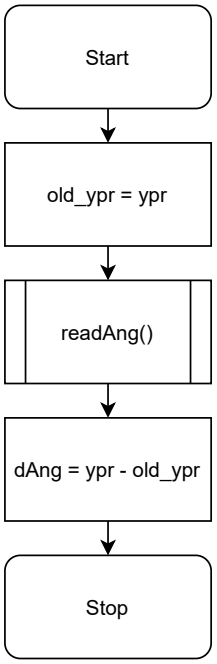


Figure 5.14: Flowchart for readDAng().

Now omz can be calculated, which will be explained in the next section.

Calculating omz

$\Delta\theta_C$ and $\Delta\theta_O$ will now be used for calculating omz.
This is done by summing the angular velocity of θ_C ($\omega_{\theta,C}$) and the angular velocity of θ_O ($\omega_{\theta,O}$).
To make the calculation between variables, the function calOMZ is made. It can be read in the Code-snippet 5.8. It has four input parameters. The input parameter omz is a pointer and it is in this pointer the calculated omz will be saved. The inputs angC and angDiff should be fed with $\Delta\theta_C$ and $\Delta\theta_O$. The final input variable is dT, and should be the sampling time between an old and a new measurement.
On a side note it can be said that calculating everything into angular velocities is rather inefficient as it is later calculated back into a change in angle. More about this can i read in the Discussion 8.8

```
uint8_t calOMZ(double& omz, float angC, float angDiff, double dT){
    float w_angC = 0.0; //temp var to store ang velocity of angC
    float w_angDiff = 0.0; //temp var to store ang velocity of andDiff
    w_angC = angC / dT;
    w_angDiff = angDiff / dT;
    omz = w_angC + w_angDiff;
    return 0;
}
```

Code 5.8: Definition of calOMZ().

Now that vq, θ_O and omz has been explained, the last parameter that is missing for the algorithm is L_O. This will be explained in the next section.

5.2.3 Measurement of L_O

The algorithm needs the length of the object in order to calculate the position of the user. It was wanted to be an adjustable input, but due to the lack of time and the lower priority this has not been implemented. Instead the length of the object is fixed and hard coded in the software before the program runs.

5.3 Motor

This section will focus on everything related to the motors mounted on the Cobot. It starts with analysing the steering technique used on the Cobot, to find out if the technique used is the best for the purpose of this project, or if another would be more optimal. It also goes into how much force the motors can deliver, and whether or not it can move the required weight at the required velocity and acceleration. Lastly it will be explained how the motor driver works, and what have been done to make a system that can drive the motors with the required velocity and desired angle to turn.

5.3.1 Steering technique

As the Cobot for this project only has two fixed-direction steering wheels and a castor wheel in front, it will only be able to be steered by differential drive. Since this has already determined the steering technique to be used, other potential techniques will be analyzed, in order to determine whether differential drive is the best suited for the ideal product of this project, or another technique would be preferable for further product development. The analysis of different types of steering techniques can be seen in Appendix D.
From the analysis it can be seen that four wheels with independent drive would actually be the most optimal way of steering the Cobot in construction sites, due to the need of making sharp turns with minimal deviation. Differential drive is also going to work, but will not offer as smooth and stable movement of the Cobot as independent drive.

As it is now confirmed that differential drive will work, and no new motor or wheel type needs to be used, the force the motors can move with will be researched to figure out if the required weight can be moved.

5.3.2 What is the max weight the Cobot can move

To figure out if the motors supplied on the Cobot can move the required 28 kg, at an acceleration of 1.328 $\mathrm{m/s^2}$ and a speed of 1.37 m/s as mentioned in technical demand 4.2, the total mass that the motors can move needs to be calculated, as well as the torque the motor can deliver needs to be measured. All the calculations in this section are simply to give an estimate of the force required.To do this, a freebody diagram of one of the wheels and motors has been created, which can be seen on Figure 5.15. As the two motors mounted on the Cobot are the same, the same freebody diagram can be used twice.

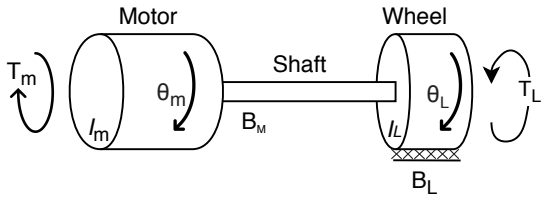


Figure 5.15: Freebody diagram for one steering wheel of the Cobot.

Where:

τ_m	The torque of the motor	$[\mathrm{N \cdot m}]$
$\mathrm{I_m}$	The motors moment of inertia	$[\mathrm{kg \cdot m^2}]$
$\mathrm{I_L}$	The wheels moment of inertia	$[\mathrm{kg \cdot m^2}]$
θ_m	The motors angular velocity	[rad/s]
θ_L	The wheels angular velocity	[rad/s]
$\mathrm{B_m}$	The friction coefficient of the motor	[-]
$\mathrm{B_L}$	The friction coefficient of the wheel	[-]
τ_L	The torque required to move	$[\mathrm{N \cdot m}]$

As it is assumed that there is no sway in the shaft, the total angular velocity can be said as $\theta_\mathrm{t} = \theta_\mathrm{m} = \theta_\mathrm{L}$. An equation of the system can be created, which can be seen in Equation 5.10. This equation does not take several things into account, such as a slanted hill, the friction of the wind and so forth. Taking everything into account that may impact the system would make the equation needlessly complicated. Because of this, this equation will only be used for giving an estimated force required.

$$\tau_\mathrm{m} - \tau_\mathrm{L} = (\mathrm{I_m} + \mathrm{I_L}) \cdot \ddot{\Theta}_\mathrm{t} - \mathrm{B_L} \cdot \dot{\Theta}_\mathrm{L} - \mathrm{B_m} \cdot \dot{\Theta}_\mathrm{m} \tag{5.10}$$

As it is the wheels that are pushing the load, the equation for force on a wheel can be used for τ_L as seen in Equation 5.11. This equation is gotten from the dynamics of how a torque applied to the center of a wheel affects the forward torque it applies as seen on Figure 5.16.

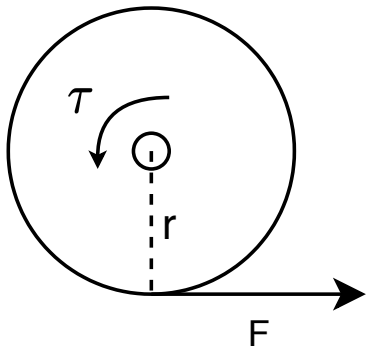


Figure 5.16: How the force reacts on a wheel

$$\begin{aligned} \tau_{\mathrm{L}} &= \mathrm{r} \cdot \mathrm{F} \\ &= \mathrm{M} \cdot \mathrm{r}^2 \cdot \mathrm{A} \end{aligned} \tag{5.11}$$

Where:

M	The mass that the motor should push	[kg]
F	The forward force	[N]
r	The radius of the wheels	[m]
A	The acceleration of the wheel	[m/s]

As it is the mass that the Cobot can load that wants to be found, τ_{L} in Equation 5.10 can be replaced with τ_{L} in Equation in 5.11, and then isolated for M.

$$\begin{aligned} \tau_{\mathrm{m}} - \tau_{\mathrm{L}} &= (\mathrm{I_m} + \mathrm{I_L}) \cdot \ddot{\Theta}_{\mathrm{t}} - \mathrm{B_L} \cdot \dot{\Theta}_{\mathrm{L}} - \mathrm{B_m} \cdot \dot{\Theta}_{\mathrm{m}} \\ \tau_{\mathrm{m}} - \mathrm{M} \cdot \mathrm{r}^2 \cdot \mathrm{A} &= (\mathrm{I_m} + \mathrm{I_L}) \cdot \ddot{\Theta}_{\mathrm{t}} - \mathrm{B_L} \cdot \dot{\Theta}_{\mathrm{L}} - \mathrm{B_m} \cdot \dot{\Theta}_{\mathrm{m}} \\ \mathrm{M} &= \frac{(\mathrm{I_m} + \mathrm{I_L}) \cdot \ddot{\Theta}_{\mathrm{t}} + \mathrm{B_L} \cdot \dot{\Theta}_{\mathrm{L}} + \mathrm{B_M} \cdot \dot{\Theta}_{\mathrm{M}} + \tau_{\mathrm{m}}}{\mathrm{A} \cdot \mathrm{r}^2} \end{aligned} \tag{5.12}$$

Now that the equation for the total mass that can be moved is found, the variables need to be found.

Measurement results

As many of these values depend on the type of motor that is used, it is possible to measure them from the motor. In Appendix J the measurements of the variables $\dot{\Theta}_{\mathrm{m}}$, $\ddot{\Theta}_{\mathrm{m}}$ and $\mathrm{B_m}$ are made. τ_{t} is not measured, as the motor is a geared motor, and stalling the output shaft is likely to damage the gears.
In Table 5.1 the results from the motor measurement can be seen.

Measurement	Result	
$\dot{\Theta}_{\mathrm{m}}$	13.61 $\mathrm{rad/s}$	0.6806 $\mathrm{m/s}$
$\ddot{\Theta}_{\mathrm{m}}$	312.65 $\mathrm{rad/s^2}$	15.63 $\mathrm{m/s^2}$
$\mathrm{B_m}$	0.0060	-

Table 5.1: Measurement results from measurement of motor.

From this it can be seen that the wanted velocity of 1.37 $\mathrm{m/s}$ cannot be reached with the motors that are used, but the acceleration can. To fix this, the diameter of the wheel should either be increased, or another type of motor should be used.
After this τ_{t}, $\mathrm{I_m}$, $\mathrm{I_L}$, the radius of the wheel and $\mathrm{B_L}$ needs to be found.

Motor torque

As the motor is a geared motor, the motor can easily be harmed if the motor's torque is tested, as it stalls the motors. Instead of risking harming the motor, the stall torque of the motor has instead been read from the webpage where it has been bought. This gives a stalling torque of 17 $\mathrm{kg \cdot cm}$ at 5.6 A, which can be converted to 1.667 $\mathrm{N \cdot m}$ [21].

Motors moment of inertia $\mathrm{I_m}$

The motor's moment of inertia is calculated by the motor's shaft as seen in Equation 5.13. From the webpage where the motor is bought, the different values can be seen for the shaft, and from this the moment of inertia can be calculated [21]. The only information that is not known, is what material the shaft is made from, but according to the site Industrial Electrical Repair service (IER) [25], the most common material used for ordinary motor shafts are mild steel, which has a density of 7850 $\frac{\mathrm{kg}}{\mathrm{m^3}}$.

$$\begin{aligned} \mathrm{I_m} &= \frac{1}{2} \cdot \pi \cdot \mathrm{L} \cdot \rho \cdot \mathrm{r}^4 \\ &= \frac{1}{2} \cdot \pi \cdot 0.0125 \cdot 7850 \cdot 0.002^4 \\ &= 2.466 \cdot 10^{-9} \end{aligned} \tag{5.13}$$

Where:

L is the rotor length	[m]
ρ is the rotor density	[$\frac{\mathrm{kg}}{\mathrm{m^3}}$]
r is the rotors radius	[m]

Wheels moment of inertia $\mathrm{I_L}$

To calculate the inertia of the wheel, the same formula used for the shaft can be used as the wheel is also a cylinder. Here the length is the width of the wheel, the radius is the radius of the wheel and the density is the density of the soft rubber that the wheel is made off which

is 1100 [26]. This calculation can be seen in Equation 5.14.

$$\begin{aligned} \mathrm{I_L} &= \frac{1}{2} \cdot \pi \cdot 0.025 \cdot 1100 \cdot 0.05^4 \\ &= 0.269 \cdot 10^{-3} \end{aligned} \tag{5.14}$$

Wheel friction coefficient $\mathrm{B_L}$

To find the friction coefficient of the wheel, the material of the wheel and the material it rubs against need to be known, which in this case is the surface in the hallway. The wheels are made of rubber, and for the floor of the hallway the material has been chosen as concrete, as it resembles the material the floors at the university is made of. This gives a friction coefficient $\mathrm{B_L}$ = 0.6 [26].

Radius of the wheel

To find this, the radius on the wheels used for this Cobot has been measure to be 0.05m.

5.3.3 Weight that can be moved

As all the variables are now found, they can be put into Equation 5.12. As there are two motors on the Cobot, the total weight can be multiplied with two. As the demand is only 1.328 $\mathrm{m/s^2}$ it does not need to maintain the measured 15.61 $\mathrm{m/s^2}$, and can instead use the 1.328 $\mathrm{m/s^2}$.

$$\begin{aligned} \mathrm{M} &= 2 \cdot \left(\frac{(\mathrm{I_m} + \mathrm{I_L}) \cdot \ddot{\Theta}_\mathrm{t} + \mathrm{B_L} \cdot \dot{\Theta}_\mathrm{L} + \mathrm{B_M} \cdot \dot{\Theta}_\mathrm{M} + \tau_\mathrm{m}}{\ddot{\Theta}_\mathrm{t} \cdot \mathrm{r}^2}\right) \\ &= 2 \cdot \left(\frac{(2.466 \cdot 10^{-14} + 2.699 \cdot 10^{-9}) \cdot (-26.56) + 0.6 \cdot 13.61 + 0.006 \cdot 13.61 + 1.667}{1.328 \cdot 0.05^2}\right) \\ &= 14.93[\mathrm{kg}] \end{aligned} \tag{5.15}$$

5.3.4 Conclusion of force calculations

From this section it has been found out that the Cobot cannot move the required 28 kg, but only 14.93 kg. The weight of the Cobot is 4.70 Kg. Since the weight that can be moved is 14.93 Kg including the Cobot, the weight that can carry except its weight is 10.23 Kg. The fact that the weight that can be transported is lower than required does not impact the functionality of the prototype, and would be fixable, only requiring a better motor. As for the velocity of the Cobot not being able to be met, this will impact the speed that the person in front can walk, and will make the Cobot slower to move. Though this can also be fixed with a stronger motor.

As there now is a deeper understanding of the type of motor and what it can do, it can now be researched and explained how to drive these motors from a microcontroller.

5.3.5 How to control DC motors

As a micro controller is used for controlling the Cobot, the micro controller also needs to control the motors. But as the Teensy 3.6 can only output a voltage of 3.3 V and has a I/O

limit of 25 mA, it cannot run the motors directly [27]. For this reason a controller need to be used.

H-Bridge - WB291111

On the Cobot that was supplied from the university, a H-bridge was already mounted and used to control the motors. This H-bridge is of the model WB291111, which uses the L298 driver [28] as can be seen on Figure 5.17. It drives the two mounted motors by receiving an enable signal for each motor, which makes the motors drive when it is high, and not drive when it is low. It also has two input pins for each motor, for a total of four, and by controlling which input pin is high it controls which direction the motors drives. But as it uses BJT transistor as the switches, there is a drop in the voltage of 2V between the input and the output. This limits the maximum velocity and torque it can deliver to the motors, with the used battery. A possible solution to this can be read about in the discussion in Chapter 8.3.

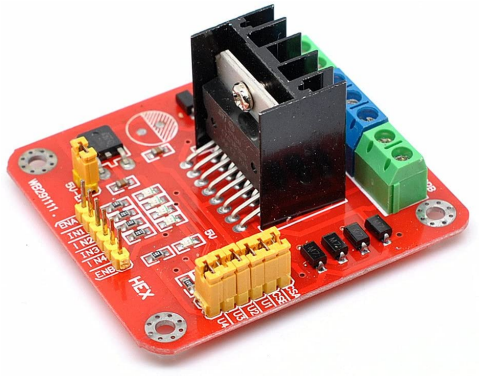
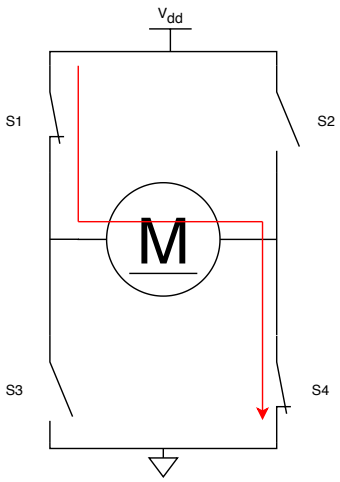
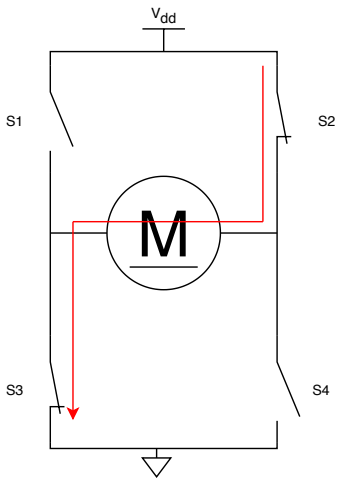


Figure 5.17: A picture of WB291111 [29].

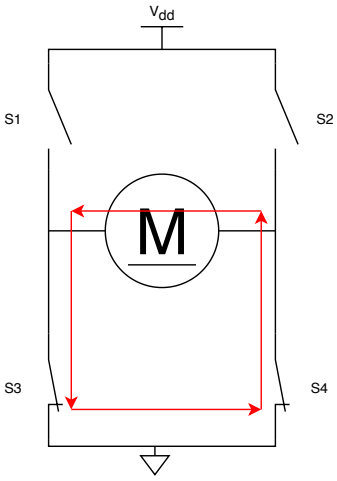
The way an H-bridge works is with the use of four transistors. By changing which transistor are on, it can open and close the current so it runs in two different directions, and this way control if the motor should run in the forward or backward direction. An illustration of how this works can be seen on Figure 5.19, where switches are used instead of transistors.



(a) H-bridge driving the motor forwards.



(b) H-bridge driving the motor backwards.



(c) H-bridge in break mode.

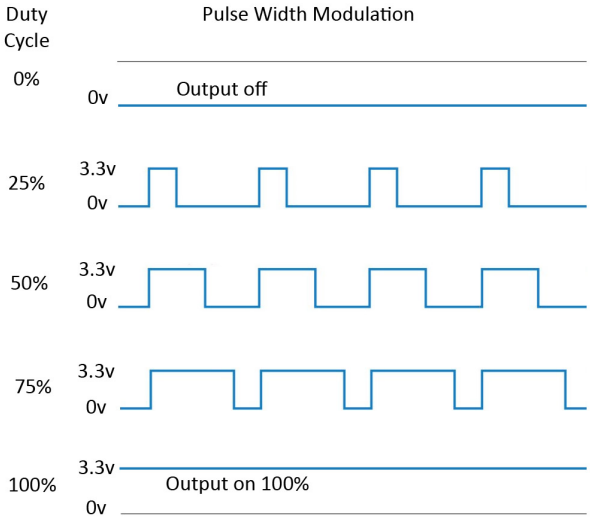
Figure 5.18: Illustrations of how an H-bridge works.

On Figure 5.18a, it shows how when the switch S1 and S3 is connected, the power will flow from the left side of the motor, to the right side and hereby driving the motor in one direction. And Figure 5.18b illustrates the opposite. It is also possible to stop the motor from running, by switching either S1 and S2 or S3 and S4, this way there is 0 V over the motor and all the energy the motor generates because of the inertia of the object being spun is dissipated as heat in the transistors and the wire of the motor as illustrated on Figure 5.18c [30]. The way the switches turn OFF or ON, which in this case are the transistors, is by applying a voltage to the transistor. As the L298 driver uses darlington cobbled transistor [28], by applying a higher voltage to the base of the transistors, more current can flow through them and be supplied to the motors, and by lowering the voltage, less current can run through. But as a microcontroller is not build for delivering a variable DC voltage, an often used way of controlling the speed of a motor is by outputting a pulse width modulated (PWM) signal instead of a variable DC voltage. This gives both the ability for the microcontroller to

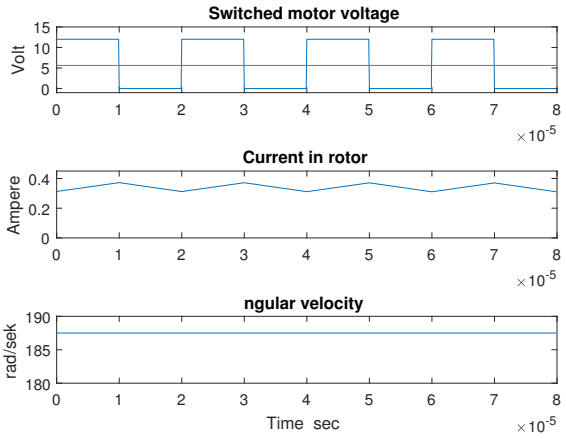
precisely control when the motors is ON/OFF and the speed of the motor, but it is also more efficient that what is possible with a DC variable control [31].

PWM

The way a PWM signal works is by instead of limiting the voltage, it limit the amount of time the voltage is ON according to the frequency you send out, this is called duty cycle. A picture of how duty cycle works can be seen on Figure 5.19a. Here it shows that the voltage is always either 0 or 3.3 V, and the only thing that is changed is how often it is 3.3 V compared to 0 V. On Figure 5.19b, it shows what happens with the current going through a motor when the PWM signal is ON or OFF.



(a) How PWM works [32].



(b) Comparison between V, A and velocity on a general motor [31].

Figure 5.19: Illustrations how a PWM signal works.

The reason the motor keeps on rotating even when the PWM signal is OFF is because of the motors moment of inertia. Which means that because what is being rotated has a mass, that mass will take some time to loose its energy. But as that will still give a variable torque of the motor, it is important to match the PWM signal that controls the motor with the inductans of the motor. This is because the inductor creates a lowpass filter, which means that when the signal switches from ON to OFF, it will slowly discharge instead of doing it instantaneously [31]. This can be used to limit the ripple seen on Figure 5.19b of the current.
As it is now known how to control the motor with a microcontroller, it will now be explained how this is implemented so that it is possible to control a differential drive Cobot.

5.3.6 Implementing the controller

The purpose of the motor controller is to control the speed of the individual wheels, and use this to both drive forwards and backwards, but also turn the Cobot. The way the Cobot turns is by setting a different speed on each wheel, as the Cobot will turn towards the wheel with the lowest speed. As the output of the algorithm is an angle the Cobot needs to turn,

the motor controller needs to translate that angle to a difference in velocity between the two motors. To make it easier to program the controller, a library made for the specific H-bridge is used. This removes the need to create a custom function for driving forward, backward and so forth. Instead a method called Drivemotors(angle,velocity,dir, dwheel, rwheel, timestep) has been created. This function controls the speed and direction of the two motors. It does this by receiving an angle to change, a velocity and the direction the motors should turn, the distance between each wheel, the radius of the wheel, and in what time i need to turn the received angle. From this it calculates an individual speed for the left and right motor. A flowchart for the entire drivemotors function can be seen in Appendix H.3, instead a few chosen functions will be explained here instead.

How to translate an angle to a difference in speed

The most essential job of the motor controller is to translate an angle to a difference in speed between the two motors. But to specify how much it should turn, it needs to be know how long it has to turn, which means that instead of an angle to turn, the controller need to know an angular velocity. As it is know that the algorithm is called every 28ms and it returns an angle, the angular velocity can be calculated as seen in Equation 5.16.

$$\omega_{\mathrm{t}} = \frac{\mathrm{angle}}{\mathrm{dT}} \tag{5.16}$$

Where:

ω_{t}	The angular velocity to turn with	[rad/s]
angle	The angle to turn	[rad]
dT	The sample time	[s]

By then knowing the angular velocity to turn with, the linear velocity of each wheel can be calculated as seen in Equation 5.17 and 5.18 [33].

$$\mathrm{v_r} = \frac{2 \cdot \mathrm{vq} + \omega_{\mathrm{t}} \cdot \mathrm{L}}{2} \tag{5.17}$$

$$\mathrm{vl} = \frac{2 \cdot \mathrm{vq} - \omega_{\mathrm{t}} \cdot \mathrm{L}}{2} \tag{5.18}$$

Where:

vq	The total forward velocity	[m/s]
$\mathrm{v_r}$	The linear velocity of the right wheel	[m/s]
$\mathrm{v_l}$	The linear velocity of the left wheel	[m/s]
ω_{t}	The rotating angular velocity	[rad/s]
R	The wheels radius	[m]
L	The length between the two wheel	[m]

As it is now known how to translate an angle to a difference in velocity, this angle now need to be converted to a PWM signal.

Velocity to PWM

In the arduino language it is possible to specify a PWM signal going from 0, to a max of 255, here 255 illustrates a duty cycle of 100 %. To convert a velocity to a fitting PWM signal a PWM scalar is needed. In order to get a fitting scalar, a measurement report has been made. This report investigates the velocity of the Cobot compared to the PWM scalar. The measurement report can be read in Appendix O. It has been made with both the PWM value going from 0-255, and from 255-0 to test the min/max velocity from a standstill and when the Cobot is already in motion.
On Figure 5.20, the measured velocity of the left and right wheel ($\mathrm{Meas_L}$, $\mathrm{Meas_R}$) can be seen according to the PWM scalar. Here it can clearly be seen that the correlation is not linear. This figure is for the test from 255-0 PWM, as this would give the best result for converting the velocity to the PWM. Because of this four different regressions has been made. Two fifth order polynomial ($\mathrm{g_l(n)}$ and $\mathrm{g_r(u)}$) and two linear ($\mathrm{h_l(m)}$ and $\mathrm{h_r(m)}$). The two polynomials are made using all measurement points corresponding to L and R. The two linear regressions are made using the lowest six measurement points.
The equations for the four regressions are shown in Equation 5.19.

$$\begin{aligned}
\mathrm{h_l(m)} &= 88.49096480 \cdot \mathrm{m} + 20.00828596 \\
\mathrm{h_r(m)} &= 89.18740852 \cdot \mathrm{m} + 20.95870863 \\
\mathrm{g_l(n)} &= 47323.64476 \cdot \mathrm{n}^5 - 73301.95932 \cdot \mathrm{n}^4 + 41948.01625 \cdot \mathrm{n}^3 - 10479.81958 \cdot \mathrm{n}^2 + \\
&\quad 1147.623332 \cdot \mathrm{n} - 10.00008440 \\
\mathrm{g_r(u)} &= 51822.73443 \cdot \mathrm{u}^5 - 76084.16644 \cdot \mathrm{u}^4 + 40949.44821 \cdot \mathrm{u}^3 - 9464.747440 \cdot \mathrm{u}^2 + \\
&\quad 941.8133747 \cdot \mathrm{u} + 1.845390312
\end{aligned} \tag{5.19}$$

Where:

m	is in the interval [0,0.326[	[m/s]
n	is in the interval [0.326,0.6582]	[m/s]
u	is in the interval [0.326,0.6375]	

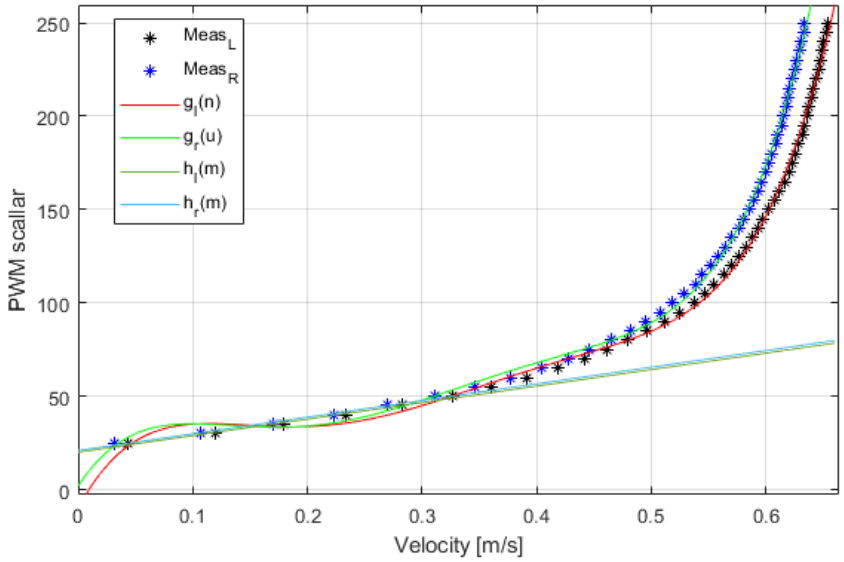


Figure 5.20: Figure of velocity and PWM points, and the four regression plots

These will be used inside an if statement, so the velocity is converted to the appropriate PWM signal as seen in Code-snippet 5.9.

```
    if (0 <= Vr <= 0.513 || 0 >= Vr >= -0.513 ) { // Checks which equation to
use to calculate the PWM
        PWMR = round(89.18740852*abs(Vr)+20.95870863);
    }
    else {
        PWMR = round(51822.73443*pow(abs(Vr),5)- 76084.16644*pow(abs(Vr),4)
+40949.44821*pow(abs(Vr),3)- 9464.747440*pow(abs(Vr),2)+941.8133747*abs(Vr)
+1.845390312);
    }
    if (0 <= Vl <= 0.513 || 0 >= Vl >= -0.513 ) {
        PWML = round(89.18740852*abs(Vr)+20.95870863);
    }
    else {
        PWML = round(47323.64476*pow(abs(Vr),5)- 73301.95932*pow(abs(Vr),4)
+41948.01625*pow(abs(Vr),3)- 10479.81958*pow(abs(Vr),2)+1147.623332*abs(Vr)
 10.00008440);
    }
```

Code 5.9: Code for determining the right PWM from velocity .

It first checks if the velocity for the right motor is over 0 and under 0.513 and if it is it uses the first order equation to calculate the PWM, if its not, it uses the fifth order equation. It then does the same for the left velocity.

Speed controller

As the algorithm only correct the degree to turn with if it has not turned as expected, the Cobot has no way of knowing if it actually turns at the speed it is set to. For this reason a speed controller should be implemented, which checks the total forward velocity set from

the Bluetooth module and compares it to the velocity measured from the motor encoder. If there is a difference it should then compare the two velocities and then recalculate a new target velocity. But because of time constraints it has not been implemented, more on this can be read about in the the discussion in Chapter 8.5.

Limit PWM value

One problem that was observed under the test of PWM to velocity, which can be seen in Appendix O, was that with the motors used, the minimum velocity that the motors can turn with is not zero, but instead 0.4387 m/s if the motor start from a standstill. This means that the different speed that can be set between each motor is smaller then expected, and could give problem with regards to turning. More on this will be discussed in the discussion in Chapter 8.1.
Although one problem that can be fixed, is when the PWM signal is calculated and results in a value higher than 255, the motor would still turn but only at the maximum speed. This could be a problem in case of a sharp turn, where the left motor speed might be 255, and the right motor speed is 325. In this case both motors would turn at 255, meaning that the Cobot would not turn but just drive straight forward. To limit the PWM value, and at the same time make sure that both motors still turn with a linear difference, a while loop has been implemented in the code as shown in the flowchart on Figure 5.21.

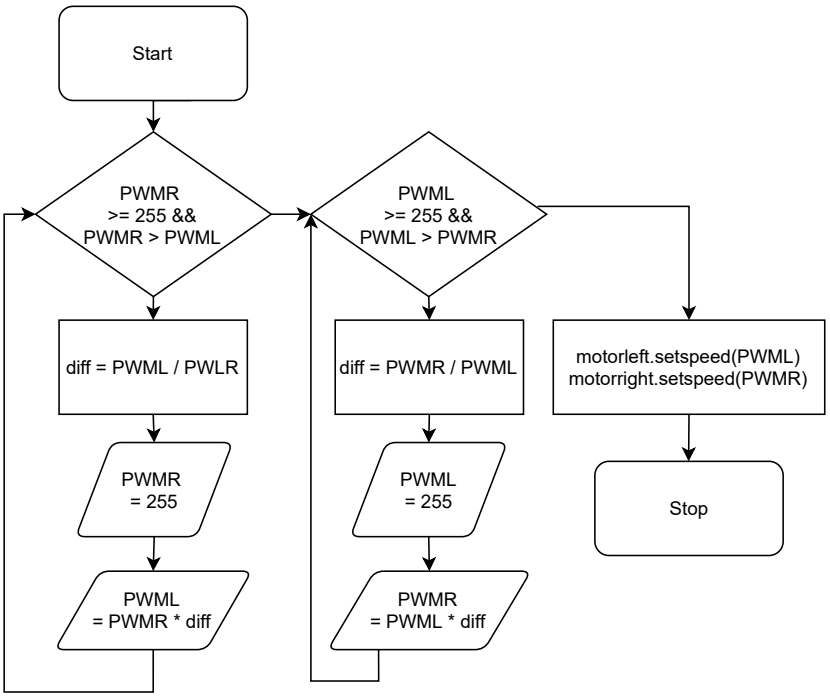


Figure 5.21: Flowchart - Limit PWM.

As it is now known how to control the wheels, how to comply with the functional demand 1.1 (The Cobot needs to adjust its speed depending on the speed of the personwalking in

front) listed in in chapter 4 will be researched.

5.4 Detect change in speed

As the user in front will most likely change his walking speed as it suits the terrain, the Cobot should also change its speed accordingly.
To be able to select the best solution for this, a few ideas will be listed and explained below. For each idea, its pros and cons will be listed, in order to be able to choose a solution.

5.4.1 Solution 1: Lifting/lowering the object

In this solution the speed is changed by either lifting or lowering the object. A sketch of this idea can be seen on Figure 5.22. Here some margin of θ_s is defined where the Cobot keeps its speed (where the acceleration is 0). When the user lifts the object, so the angle θ_s becomes larger, the acceleration will be larger than 0, and thereby increasing the speed. Lowering the speed is done the same way, by lowering the object so θ_s becomes smaller, the acceleration accordingly becomes lower than 0.

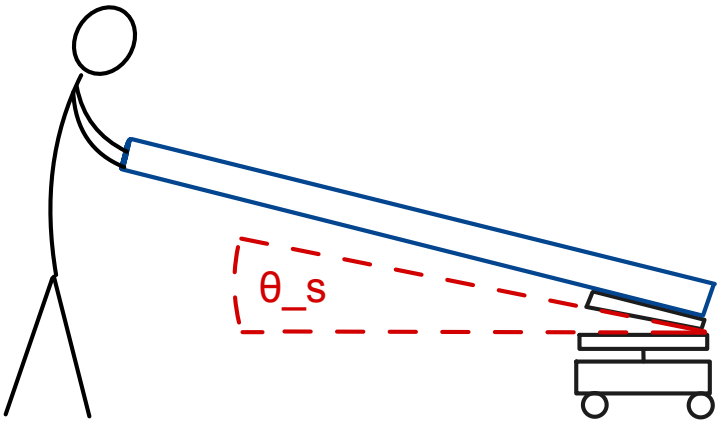


Figure 5.22: Illustration for solution 1.

Pros/cons

+ The user does not need to remove his hands from the object.
- The user has to use extra force to lift the object higher for the time of acceleration.
- Needs a physical measurement of angle θ_s.
- Height difference of users.
- Threshold height has to be implemented so the speed is not affected by difference in the terrain.

5.4.2 Solution 2: Pull/push the object

In this idea the user raises the speed of the Cobot by pulling the object. The Cobot should then measure its current speed and keep this speed. To slow down again, the user pushes on

the object, so the Cobot is slowed down. When slowed down the Cobot measures the speed, and keeps it.

Pros/cons

+ Does not need extra measurements
+ Does not need the user to remove his hands from the object.
- The user has to be able to pull the Cobot and the object to start.
- The motors has to be turnable from the wheel side.
- The object might be too heavy for the user to pull it.

5.4.3 Solution 3: Remote control

As one of the functionalities for the Cobot is the ability to be remotely controlled, it would also be possible to use this remote controller to control the speed of the Cobot when it is not in manual control mode.

Pros/cons

+ Easy to implement, as the connection already needs to be made between the remote-control and the Cobot.
- Requires the user to use (at least) one hand to operate the speed control.
- Requires a remote control to use the fundamental feature of the Cobot.

5.4.4 Conclusion of speed control

As it is a prototype that will be built during this report, solution three is selected, as it is the easiest to implement and gives the possibility to show the concept of the product. This solution is selected even though it has the disadvantage that it requires the user to remove his hands from the object, which is not feasible for the ideal product.

As solution three was chosen, it now need to be defined what communication technology that should be used, and how the user should communicate with it

5.5 Remote Control

As one of the desired functionalities of the Cobot is for it to be able to be remotely controlled, a controller will be designed. Furthermore it is decided in Section 5.4 that the change in speed while following a user should be done through the controller.

In order to implement the controller, both the controller and the Cobot should comply with the same protocol. In the next section this protocol will be defined.

5.5.1 Communication protocol

This section will define the protocol that both devices has to follow in order to communicate. First a technology will be defined, and thereafter the framing of messages.

Technology

As they both have to use the same communication technology, an analysis has been made in order to determine the one that is best suited for this project. This analysis can be read in Appendix E. From this the technology Bluetooth Low Energy (BLE) has been chosen.
In order to get insight on BLE, a short analysis of the key characteristics has been made, which can be read in Appendix F. As BLE defines all of the lower level protocols the next section will define the top layer message framing.

Framing

From the phone, four different parameters can be sent. These and the framing of these can be seen in Table 5.2.

Parameter	**Frame**	Option	Unit
Direction of Cobot	*string*	"forward" or "backwards"	-
Angle to turn	"angle:*float*"	$-\infty...\infty$	deg/s
Velocity the Cobot should drive at	"velocity:*float*"	$0...\infty$	m/s
Manual or Path follow	*string*	"manualdrive" or "autodrive"	-

Table 5.2: Framing of commands.

The command angle: and velocity: can have a floating number sent with it, with a length of up to six characters. This limitation allows for more then one command to be send in the same string. Each of these commands can be sent either separately or together. For example if the Cobot should turn 5.3 with degrees/s, the string bellow should be sent:

"angle:5.3"

If it should drive forward with a velocity of 0.3 and also turn with 7.32 degrees/s the following string can be sent:

"angle:7.32, velocity: 0.3"

It then recalculates that angle in degrees to rad, so the motor controller can receive it. As the motor encoder recalculates it an angle velocity, dT is set to one second for when it is in manualdrive.It should also be noted that for all velocities sent higher than the max velocities of the Cobot, will result in the max velocity possible for the Cobot, this is limited in the motor controller.
Now that the protocol has been defined, the development of the controller and the receiver can be performed. In the next section, the design of the controller will be documented.

5.5.2 Transmitter - remote

One of the reasons BLE was chosen is that it is already implemented in phones, and thereby removing the need for designing new hardware for the controller. The idea is that an app is supposed to be developed, where the user would be able to intuitively control the Cobot, via a virtual joystick and a few buttons. But since the app is not necessary to show the concept of the Cobot, a simplification of this will be used.
This means that on the phone, another app will be used [34]. This app create a serial connection to the Bluetooth module, which makes it possible to send text string and data from the phone to the Cobot. In the next section the development of the receiver part will be described.

5.5.3 Receiver - Cobot

On the receiver end, it has been decided to use the BLE module HM-10 [35]. In order to use the module, there has been developed two functions.One function for receiving the raw data from the remote controller, and another function that parses the raw data so the Cobot can separate the values from the string received. In the next section it can be read how the communication with the remote control has been implemented.The two functions implemented are receiveData() and parsebluetoothdata(angle, velocity). These two will be explained in the two following sections, starting with receiveData().

receiveData()

The receiveData() function reads the raw data from the remote controller, and saves it as a String. It uses the library called SoftwareSerial to read data from the HM-10 Bluetooth module. This has later been found to be a bad implementation, as the Teensy has multiple hardware serial ports that could have been used, which has build in 64-bit buffer. The code for this part can be seen in Code-snippet 5.10. The code checks if any data has been received, and if this is the case, it then saves it as a string and returns it to the function call. If no data has been received, it returns a false statement.

```
String receiveData() {
    String data = "";
    HM10.listen(); //
    if (HM10.available() > 0) {
        data = HM10.readString();
        return data;
    }
    else
    return String("false");
}
```

Code 5.10: Code-snippet for receiveData function

parsebluetoothdata

Parsebluetoothdata is the function that translates the data received from the remote controller, into the desired variables.
To explain the causality of the function, a flowchart has been created that can be seen on Figure 5.23.

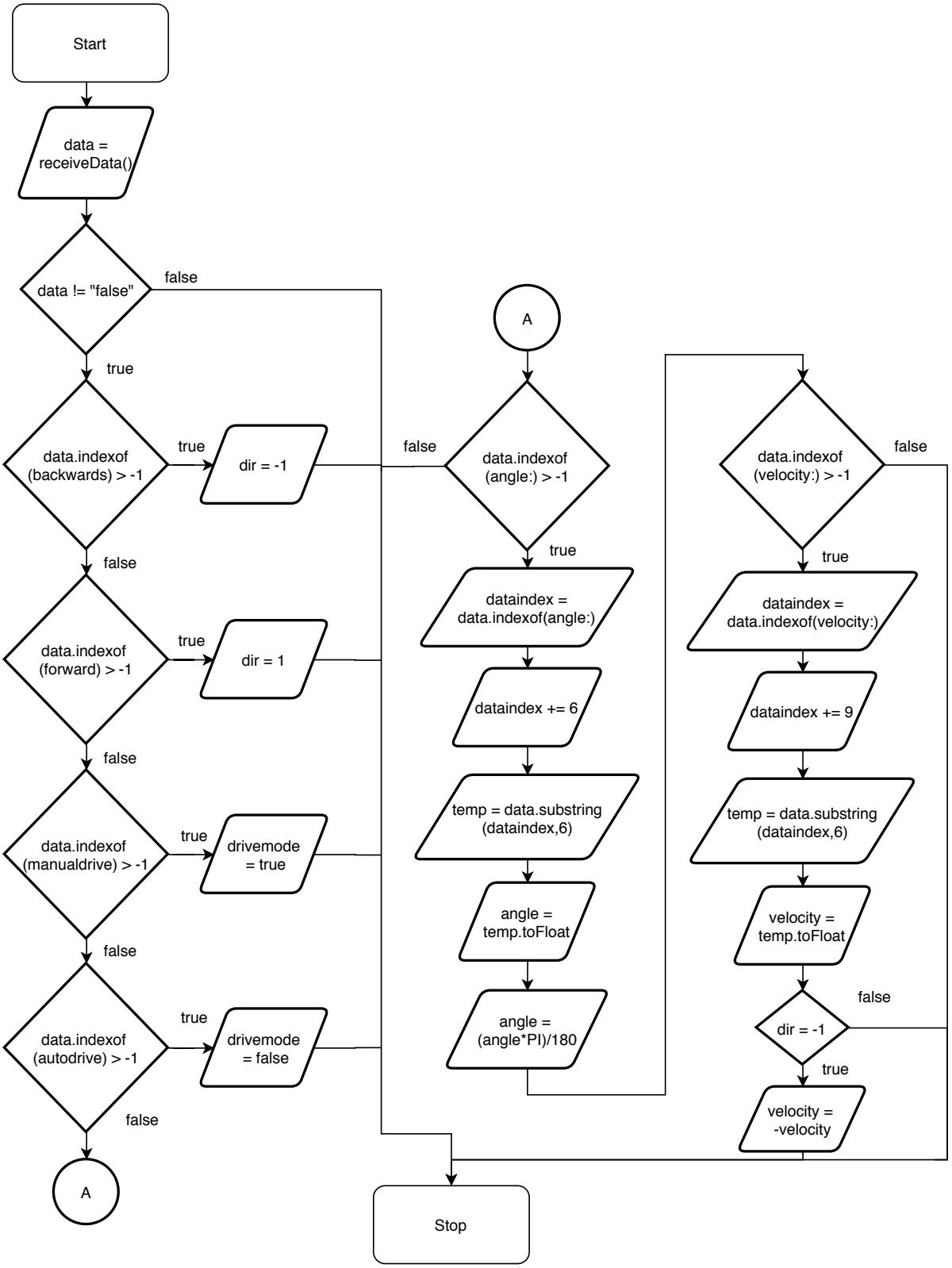


Figure 5.23: Flowchart for the bluetooth function parsebluetoothdata.

In the next section the construction of the Cobot will be covered.

5.6 Construction of the Cobot

In this section, the construction of the Cobot will be explained with the thoughts and decisions that has been made. The main idea through the construction is to design a solid and robust Cobot that can comply with all demands listed in Section 4.2. The Cobot is a modification of a robot model given by AAU. The whole model consists of a platform of plexiglass, two back wheels of hard rubber and a castor wheel in front. The plexiglass is shaped as shown on Figure 5.24, and it has been used for the base view for the Cobot, of which the rest of the Cobot has been built upon. In Table 5.3 a list of the components which are used for the Cobot are introduced, focusing on the type of material and usage. The wiring diagram of the Cobot can be seen in Appendix I.

	Material type	**Usage**
Robot	Made by AAU	Base platform of cobot
Tachometer	Rotary Encoder - 1024 P/R (Quadrature)	Angle measurement
Accelerometer	HW-290	Acceleration measurement
Arduino	Teensy 3.6	Implementing whole code
Battery	iMax B6AC Li-po charger	Battery for Cobot
H-bridge	WB291111	Control the motors
Perforated table	Wooden boards	Sustaining the object and being rotated by the user
Ball rollers	Ball Transfer Unit, 15.875 mm, with mounting holes	Reducing friction between plate
3x plates	Wooden boards	Different layers of turn-table
Bluetooth	HM-10 BLE Bluetooth 4.0 module	Remote control

Table 5.3: Material type and usage for each element.

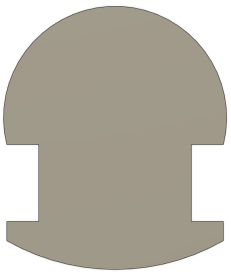
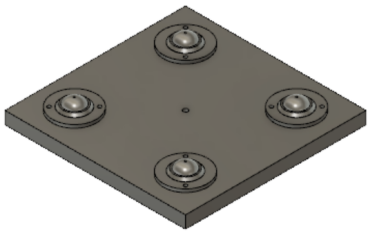


Figure 5.24: Base view of the Cobot.

5.6.1 The turn-table

The functional demand 2.2 from Section 4.2 states that "The Cobot needs to follow the same path that the person walking in front takes. This demand is complied with by measuring the angle between the Cobot and the load on top, so that the Cobot can use the measured angle to navigate. Therefor, a turn-table has been designed. The turn-table consists of two wooden plates attached to each other with four ball rollers between. The purpose of the turn-table is to have a platform where loads can be tied upon and is able to rotate without rotating the rest of the Cobot, so the angle between the Cobot and object can be measured. To do this, the

top-layer of the turntable can rotate and the bottom-layer stays fixed to the Cobot and is not able to rotate. The purpose of the bottom-layer is to make the Cobot more firm and solid. In order to reduce the amount of friction between the two layers, four ball rollers has been mounted on the four corners of the bottom-layer, for the top-layer to rotate upon. The ball rollers have been placed within a radius of 11.5 cm from the center of the plates with equal spacing to each ball roller. This has been done to make all four ball-rollers always support the top layer no matter how it is rotated.
The turn-table and the ball rollers can be seen on Figure 5.25a and 5.25b.



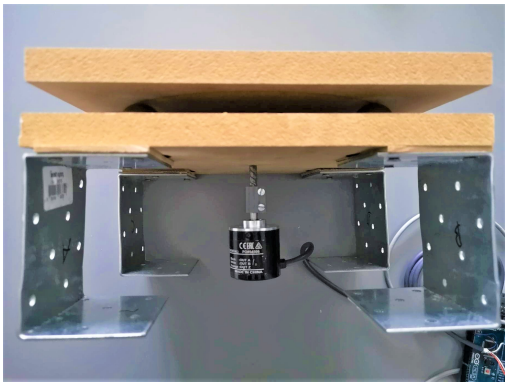
(a) Bottom layer of the turn-table with ball-rollers attached.

(b) Turntable with mounting brackets.

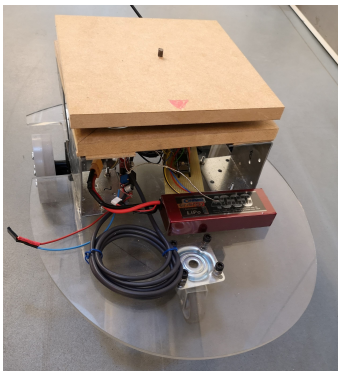
Figure 5.25: Illustrations of the turn-table and mounting brackets.

5.6.2 Mounting of tachometer

The tachometer has been chosen for the angle measurement between the object and the Cobot. The tachometer is placed underneath the turn-table upon the plexiglass to be able to measure the angle of the turn-table when connected. To make space for the tachometer between the turn-table and the plexiglass, four stands of perforated plates have been attached under the four corners of the turn-table to lift it up, as seen on Figure 5.26a. In order for the tachometer to measure the angle of the turn-table, a hole has been drilled through the middle of both parts of the turn-table. Hereafter a rod of iron has been mounted on the top-layer and connected to the tip of the tachometer to measure the rotation as seen on Figure 5.26a. As the tachometer will break if too much downwards or sideways force is applied to it, it is important to minimize this force as much as possible. This has been achieved by having the weight press down on the ball rollers, instead of the iron rod connected to the tachometer. As for the sideways force which is applied when the object is pulled, this is minimized by the iron rod pressing against the wooden part of the turn-table when it is pulled to either side, instead of pulling on the tachometer. The end result of the Cobot design is shown on Figure 5.26.



(a) The tachometer connection.



(b) The whole Cobot.

Figure 5.26: The tachometer attachment and the final Cobot.

5.7 Main code

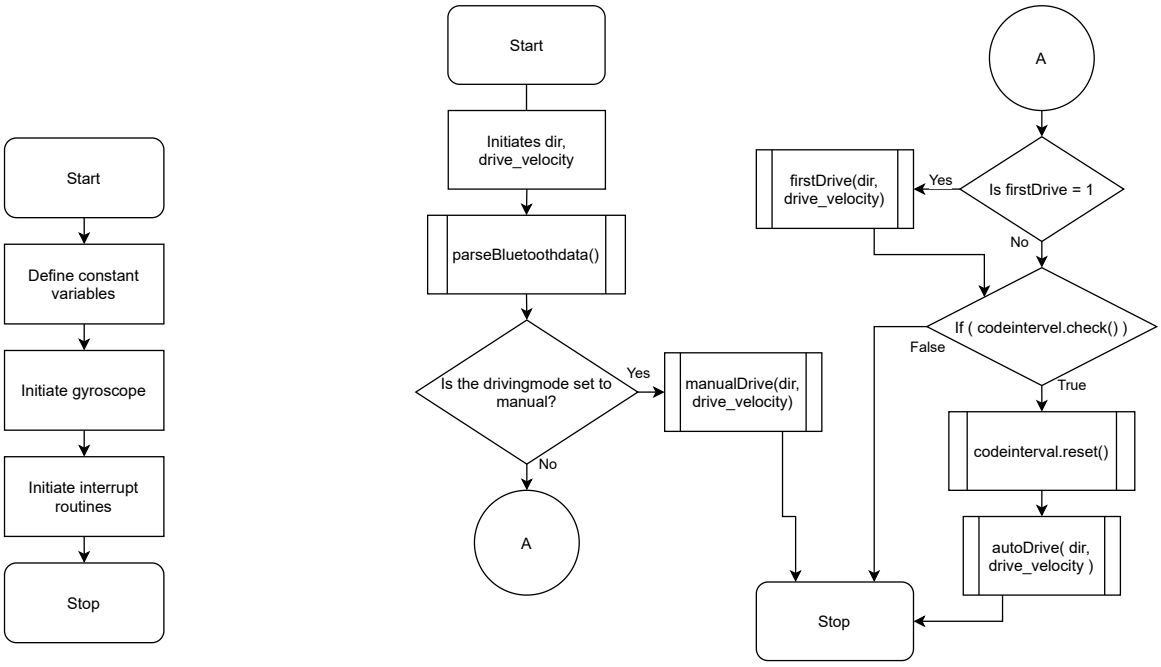
To give a better overview of how the system works, two flowcharts have been created for the system. A setup flowchart that configures all systems, which can be seen on Figure 5.27a, and a main loop flowchart that shows how the system continually runs, which can be seen on Figure 5.27b. In the Appendix H.4 a flowchart of all the drive modes functions can be found, including the first drive function.

Main setup

The main setup can be seen on Figure 5.27a and runs as the first thing at startup. Here the system variables are defined. Afterwards the gyroscope is setup, which is done by setting up the on-board Digital Motion Processor (DMP). The DMP allows the module to make calculations and measurements on its own, so the microcontroller is free to process other functions [36]. As a final thing, it initiates the interrupt routines. These are interrupts for the motor encoder and the encoder measuring the angle between the Cobot and the object.

Main Loop

The main loop is in charge of running the different parts of code in the right order. A flowchart of this can be seen on Figure 5.27b. First, it sets up the driving direction and the speed and afterwards checks which drive mode to choose, whether it is getting a Bluetooth signal or not. If it is not set to manual drive, it will the then run the autoDrive function every 28 ms according to the calcultions done in Section 5.1.6. These 28ms are set as the *codeinterval*.



(a) Main setup.

(b) Main loop

Figure 5.27: Flowcharts for the main code.

5.8 Conclusion on design

Throughout the design chapter, the main functionality has been clarified and implemented. As the main functionality is highly depending on the algorithm, a lot of focus have been put into this and the measurements needed to make it work. The four measurements the algorithm needs are the length of the carried object (L_O), the angle between the Cobot and the object (θ_O), the angular velocity of the object (omz) and the forward velocity of the Cobot (vq).
The measurement of angle θ_O has been implemented as an incremental encoder connected to the turn-table on the Cobot. The angular velocity omz has not been implemented as a direct measurement on the turn table using a gyro. Omz is instead calculated using both θ_C and θ_O.
To control the speed of the Cobot, a motor controller has been developed. This controller is in charge of turning the Cobot a certain angle, and make the motors drive with the desired speed. The exact velocity of the Cobot, vq, is measured by the motor controller and thereafter used in the algorithm. From these variables the algorithm returns an angle that, when subtracted from the object angle, outputs the amount the Cobot should turn to reach the current target point.
The motors used on the Cobot were also measured and tested to see if the motors could supply the required force. From this, it was concluded that the Cobot will not drive at the demanded speed of 1.37 m/s, but instead only at 0.68 m/s.

The speed control has been implemented using a phone as the remote control and therethrough sending a wanted speed. Furthermore, it got the capability to be manually controlled if needed. It was possible to use a phone as the remote, as the wireless technology chosen for the remote is Bluetooth BLE.
The main focus of the physical construction of the Cobot has been on how to develop the turn-table where the carried object can be placed, and at the same time measure the angle that the table has been turned. This has been done by having two wooden plates lifted over the base of the Cobot with ball rollers in between, of which the top one has a metal rod mounted on it, so an encoder can measure the rotation on the rod.

As the construction of the prototype of the Cobot has been finished, all of the demands will be tested in the following chapter.

Chapter 6

Test of demands

In order to confirm that the Cobot works as required, a test procedure is made for each demand listed in Chapter 4. This way it is assured that every demand is being tested and complied with. The demands are divided into two categories, functional demands and technical demands. For each demand, a test procedure will be listed, and what is required for the test to be deemed successful.
As most of the original demands listed have been written from a customers viewpoint instead of a technical viewpoint, these demands have been modified to fit the solution developed throughout the Design chapter in Chapter 5. For all the tests that have been modified, a short limitation of the test will be introduced. More about this can be read about in the discussion in Section 8.11.
As the Cobot did not end up working as intended, some of the demands are not tested, but for all the demands, a test procedure has been created. More about the final state of the Cobot can be read about in the discussion in Section 8.12.

6.1 Functional demands

Test of functional demand 1.1

"The Cobot needs to adjust its speed depending on the speed of the person walking in front".

Limitation for the test:
The method for adjusting the speed of the Cobot according to the person in front, has been implemented by sending a command via Bluetooth with the desired speed. Therefore this test is performed by sending a message with the desired speed and verifying that the Cobot is driving at this speed, while the Cobot is in autoDrive mode.

Setup:
The Cobot is placed in one end of a hallway with a length of at least four meter, facing the other end. From the starting point of the Cobot, a point A that is two meters away is marked. An additional two meters away, point B is marked. An illustration of this path can be seen on Figure 6.1.A stopwatch is needed for this test. A smartphone application is used for controlling the speed by sending a velocity command to the Cobot.

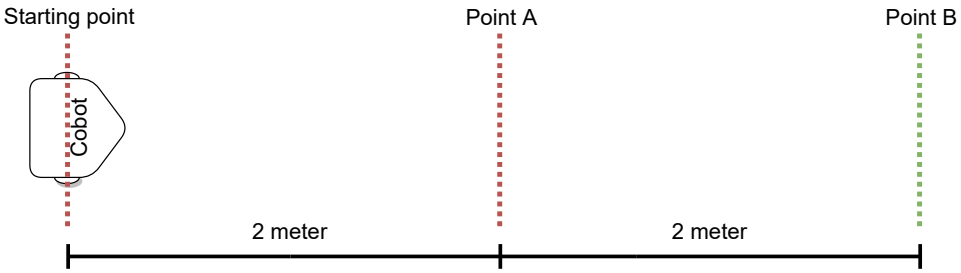


Figure 6.1: Test path for functional demand 1.1.

Step no.	Description
1	Turn on the Cobot and connect the remote to the Cobot's bluetooth module.
2	Send a message through the smartphone application containing "velocity:0.3", to set the speed to 0.3 m/s.
3	When the Cobot passes point A send a message through the smartphone application containing "velocity:0.5", to set the speed to 0.5 m/s.
4	Start the timer on the stopwatch when the Cobot passes point A.
5	Stop the timer on the stopwatch when the Cobot passes point B.
6	Repeat step 2, 3, 4 and 5 three times from the starting point.
7	The speed of the Cobot for each of the different speeds can be measured by dividing the distance of two meter by the time it has taken to reach the point.

Table 6.1: The test procedure for functional demand 1.1.

Requirements for test to be deemed successful:
If the calculated speed is corresponding to 0.5 m/s for all three measurements, the test is deemed successful.

Test results:
This demand was not able to be tested.

Test of functional demand 1.2

"The Cobot needs to be able to be controlled remotely".
Setup:
A mobile application is used for sending commands for specific velocities and turning angles. A path is marked on the floor as can be seen on Figure 6.2, as of which the Cobot is placed at point A.

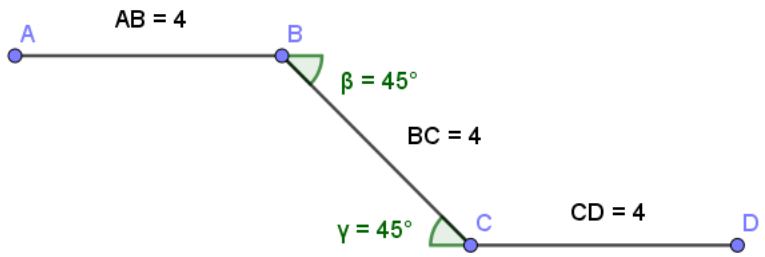


Figure 6.2: The test path for functional demand 1.2.

Step no.	Description
1	Turn on the Cobot and connect the remote to the Cobot. Send a signal to turn it to manual drive mode.
2	Place the Cobot at point A and send messages to the Cobot containing "angle:0.0" and "velocity:0.3", making it drive straight forward.
3	As the Cobot reaches point B, send a message containing "angle:45.0". One second after, send a message containing "angle:0.0".
4	As the Cobot reaches point C, send a message containing "angle:-45.0". One second after, send a message containing "angle:0.0".
5	The Cobot should now approach point D, and as it reaches this point, send a message containing "velocity:0.0" to stop the Cobot.

Table 6.2: The test procedure for functional demand 1.2.

Requirements for test to be deemed successful:
The path of the Cobot should follow the instructions that the user has sent. If the instructions are matched correctly to the ones specified, the path should look like the one on Figure 6.2. If the path of the Cobot is close to the one from the instruction sent, the test is deemed successful.

Test results:
This demand was not able to be tested.

Test of functional demand 1.3

"There needs to be a physical input on the Cobot that can specify the length of the object carried".

This test has not been performed, since the physical buttons has not been implemented on the Cobot. For this reason, this test has been deemed unsuccessful.

Test of functional demand 1.4

"The Cobot needs to function in the hallway of AAU".

Requirements for test to be deemed successful:
The test of this demand depends on the test of all the other tests. This means that the test is only deemed successful, if all other tests are deemed successful in the AAU hallway.

6.2 Technical demands

Test of technical demand 2.1

"The Cobot needs to be able to receive a wireless signal from a distance of at least 10 m".

Setup:
The test has been done on a hallway of at least 10 meter with no obstacles in-between and the front of the Cobot pointing towards the user.

Step no.	Description
1	Place the Cobot 10 meter away from the user.
2	Turn on the Cobot and connect the remote to the Cobot. Send a signal to turn it to manual drive mode.
3	Send a message through the smartphone application to the Cobot containing:"angle:10" and "velocity:0.4".
4	Visually check that the Cobot has turned while driving forward.
5	Repeat step 1, 2, 3 and 4, but instead with the following message: "angle:15" and "velocity:0.5".

Table 6.3: The test procedure for technical demand 2.1.

Requirements for test to be deemed successful:
If the Cobot drives according to the message sent for each of the two runs, the test is deemed successful.
Measurement results:
For both of the run throughs, the Cobot was able to receive both the command to turn to manual drive mode, and also start turning while driving forwards, at a distance of 10 meter. Thereby this demand has been deemed successful.

Test of technical demand 2.2

"The Cobot should be able to follow a person carrying an object up to 4 m".

As the technical demand 2.3 requires that the Cobot follows the same path of the user in front, this demand will be tested with technical demand 2.3.
If the technical demand 2.3 is deemed successful, this demand will be deemed successful as well.

Test of technical demand 2.3

"The Cobot's maximal displacement margin according to its target path should be (with a object length of up to 4 meters) < 13.10 cm".

Limitation for the test:
Due to the availability of space, the four meter object has been replaced with a two meter object.
Setup: The Cobot will be following a person walking in front while carrying a two meter long object. A path is marked on the floor in the hallway, which can be seen on Figure 6.3. The Cobot is starting in point A and the user is starting in point B with the object.

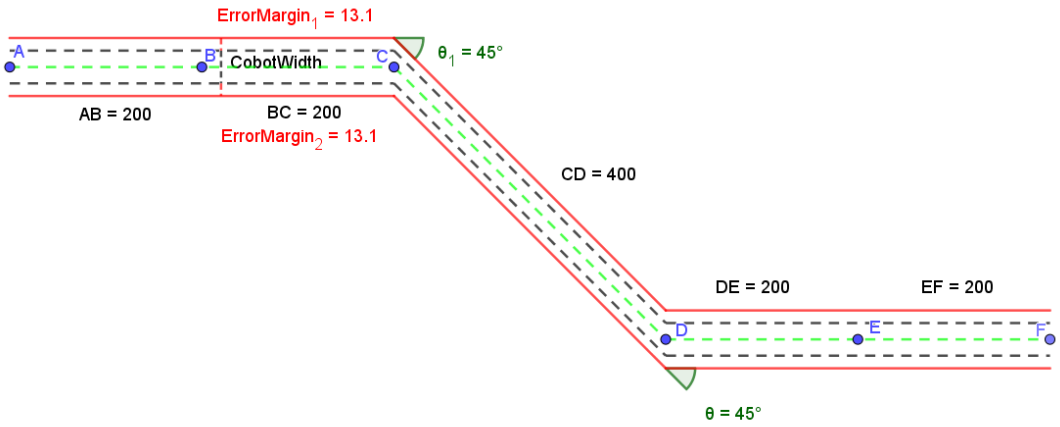


Figure 6.3: The test path for technical demand 2.3.

Step no.	Description
1	Turn on the Cobot and connect the remote to the Cobot.
2	The user sends a message to the Cobot containing "velocity:0.3"and starts walking with the object.
3	The user goes through the points C, D, E and F.
4	As the user reaches point F, the user sends a message to the Cobot containing "velocity:0.0" and stops walking.
5	Place the Cobot at point A and the user at point B with the object. Repeat step 2,3 and 4 an additional time.

Table 6.4: The procedure for technical demand 2.3.

Requirements for test to be deemed successful:
The Cobot should be observed while driving to verify that it stays within the two red lines for the whole path in order for the test to be deemed successful. If the Cobot stays within the red lines for both of the runthroughs, this test will be deemed successful.

Test results:
This demand was not able to be tested.

Test of technical demand 2.4

"The minimum resolution of the angle measured between the Cobot and the object should be less than 1.877°".

Setup:
The Cobot will be placed one meter from a wall. A laser pointer has been mounted on top of the turn-table, pointing towards the wall. The setup can be seen on Figure 6.4.

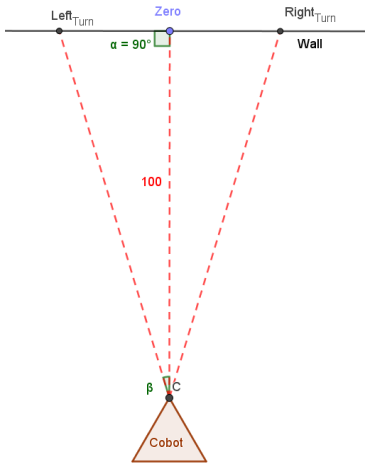


Figure 6.4: Setup for test of demand 2.4.

Step no.	Description
1	Turn on the Cobot and set it to manual drive mode, where the motors are set not to drive and open the Serial monitor
2	Turn the turn-table to an angle of 0° on the serial monitor and mark the place on the wall the laser pointer is pointing.
3	Turn the table to a point where it says 355.08° on the serial monitor and mark the point from the laser on the wall. Measure the distance between this point and the zero point.
4	Turn the table to a point where it says -355.08° on the serial monitor and mark the point from the laser on the wall. Measure the distance between this point and the zero point.
5	Repeat step 2, 3 and 4 three times.

Table 6.5: The test procedure for technical demand 2.4.

Requirements for test to be deemed successful:
The difference between the angle shown on the serial monitor and the angle measured should be less than 1.877° for every test.

Measurement results:
The test results of the measurements can be seen in Table 6.6a and 6.6b which shows the angle measured when turning to left and the angle turning to right as well. The two measurements have been verified by calculating the actual angle and find the deviation, which have been done by using Equation 6.1.

$$\beta = \tan^{-1}(\frac{\text{dist}}{100}) \tag{6.1}$$

Measured by Cobot encoder	Calculated	Deviation
-355.08	-356.31	1.23
-355.08	-355.25	0.17
-355.08	-355.09	0.01

(a) Table for left turn.

Measured by Cobot encoder	Calculated	Deviation
355.08	355.19	0.11
355.08	355.14	0.06
355.08	355.20	0.12

(b) Table for right turn.

Table 6.6: Left and right test results of angle turn.

Conclusion of the test:
Because the angles measured by the calculated does not deviate more than 1.877 ° from the angles measured by the tachometer the test has been deemed successful.

Test of technical demand 2.5

"The Cobot should be able to drive with a speed up to 1.37 m/s with a total weight of 28 kg".

Limitation for the test:
The Cobot has been tested carrying 28 kg, but as the wheels have been 3D printed and are made of plastic, they were the limiting factor of this. Therefore this test has been performed without any additional weight. More about this problem can be read in the discussion in Chapter 8. Another limitation is, that the motors that has been used, has been measured to have a maximum velocity of 0.6 m/s which can be seen in the measurement report in Appendix J. For this reason the test will only be made to see if Cobot is able to reach the motors speed limit of 0.6 m/s.

Setup: The points shown in Figure 6.5 are marked on the floor, of which the Cobot is placed in point A. A stopwatch is needed for this test.

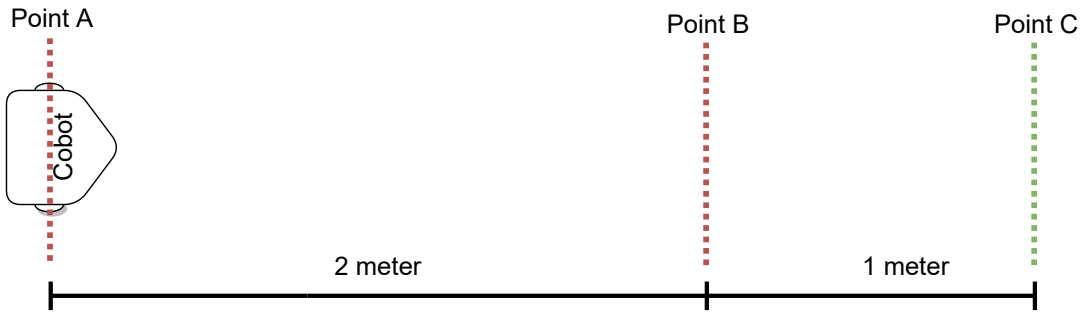


Figure 6.5: Test path for technical demand 2.5.

Step no.	Description
1	Turn on the Cobot and connect the remote to the Cobot. Send a signal to turn it to manual drive mode.
2	Place the Cobot at point A, pointing towards point B.
3	Send a message through the smartphone application to the Cobot containing: "angle:0" and "velocity:2", to set it drive straight at the fastest speed.
4	Start the timer when the Cobot passes point B, and stop the timer when the Cobot passes point C. Document the timing result.
5	Step 2, 3 and 4 are performed two additional times.

Table 6.7: The test procedure for technical demand 2.5, 2.6 and 2.7.

Requirements for test to be deemed successful:
If the maximum speed of each of the three measurements is at least 0.6 m/s, the test is deemed successful.
Measurement results:
Since the distance traveled over the timed period is one meter, the average speed can be calculated by dividing one by the documented time for each of the three measurements.The measured result can be seen in Table 6.8.

Distance	Time	Velocity
1m	1.98s	0.505m/s
1m	1.96s	0.51m/s
1m	1.93s	0.518m/s

Table 6.8: Measurement result for velocity

Conclusion of the test:
As the measured speed is less than 0.6 m/s for all measurements, this test has been deemed unsuccessful.

Test of technical demand 2.6

"The Cobot should be able to accelerate with an acceleration up to $1.328\mathrm{m/s^2}$ with a total weight of 28 kg".

Limitation for the test:
The Cobot has been tested carrying 28 kg, but as the wheels have been 3D printed and are of plastic, they were the limiting factor of this. Therefore this test has been performed without any additional weight. More about this problem can be read in the discussion in Chapter 8.

Setup:
The lines shown on Figure 6.6 are marked on the floor, and the Cobot is placed on the starting line. A phone is used for controlling the Cobot and a stopwatch is needed for this test.

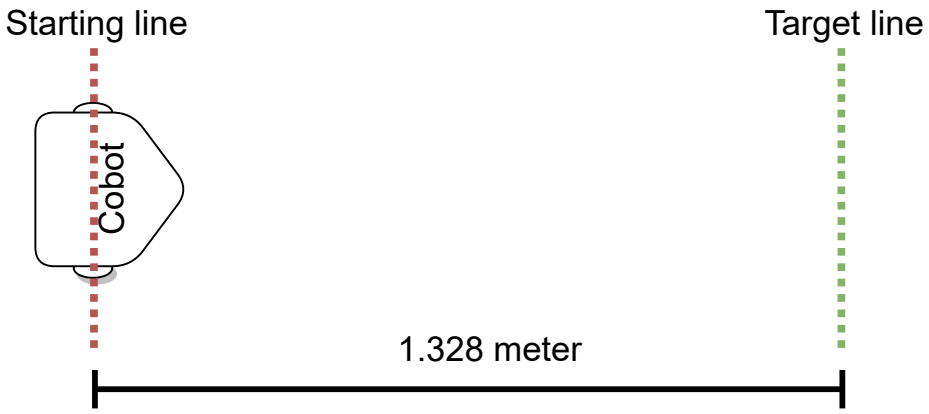


Figure 6.6: Test path for technical demand 2.6.

Step no.	Description
1	Turn on the Cobot and connect the remote to the Cobot. Send a signal to turn it to manual drive mode.
2	Place the Cobot at the starting line, pointing towards the target line.
3	Send a message through the smartphone application to the Cobot containing: "angle:0" and "velocity:2", to set it drive straight at the fastest speed. At the exact same time as the velocity is sent, start the stopwatch.
4	As soon as the stopwatch reaches one second, the position of the Cobot is marked to verify if it has passed the target line.
5	Step 2, 3 and 4 are performed two additional times.

Table 6.9: The test procedure for technical demand 2.5, 2.6 and 2.7.

Requirements for test to be deemed successful:
If the Cobot passes the target line for all of the three tests, within the time limit, the test is deemed successful.

Measurement results:
This could not be tested.

Chapter 7

Conclusion

From the introduction to this report, the following problem statement has been made:
Is it possible to make a solution that lowers the amount of physical injuries, while maintaining or lowering the cost for the employer, focusing on the construction industry?
Through the problem analysis, the main problem has been found to be the lifting of heavy and long object, and by researching previous solutions, it was concluded that none of these solutions solved the problem. From this, as well as an interview with a carpenter, it has been concluded that a Cobot that can automatically follow a person in front and assist with the carrying of long loads is a possible solution to this problem. Through a technical analysis, what is required for a Cobot to solve this problem has been analysed, and a list of demands has been set. Afterwards, it was researched how to create a Cobot that are able to comply with this list of demands.
An algorithm that makes it possible to store the previous coordinates of the user in front has been implemented on a Cobot, whereas sensors have been used for supplying the needed measurements for the algorithm. The report has covered which technology to use for each measurement, and how these have been implemented. As the Cobot should be able to navigate to the desired coordinates, a motor controller for steering the Cobot with differential drive has also been implemented.
As all of the needed functionalities of the Cobot has been determined and implemented, a test of all the listed demands have been performed. Due to sudden errors in both the software design as well as physical error, some demands have not been tested while some have been tested and failed. As all of the listed demands have not been complied with, the Cobot that has been developed throughout this report does not solve the problem listed in the problem statement. Although, as each individual part works, as well as it was also possible to get the Cobot to roughly follow a person, it is assumed possible to create a fully functional Cobot.

Chapter 8

Discussion

This chapter covers some discussions about the contents of this project. It covers the points that has not been conducted or could have been done in a better way, if there were no limitations of availability or time. Moreover, in the last section of this chapter, the current state of the Cobot is described and documented, along with the reasoning for the Cobot not being able to comply with some of the demands.

8.1 The minimum and maximum velocity of the motor limits the product

As it has been found out in the measurement report in Appendix O, the motor has a minimum velocity of 0.461 m/s and a maximum of 0.645 m/s, when starting from 0 m/s. Firstly, this gives a limit on the maximum velocity of the Cobot, as it is not the required 1.37 m/s. It also limits how precise we can turn, as there is a small difference between the min and max velocity. In order to fix these problems, new motors should be used which both give a higher maximum velocity and a lower minimum velocity. Another optimisation might also be to tune the PWM frequency so it fits with the coil inside the motor. This way a more linear torque will be delivered to the motors which should make it possible to turn at a lower velocity.

8.2 Logic power output of H-bridge gets hot & missing 5V supply to the Teensy

To power the encoders the logic output from the H-bridge has been used, which can deliver 5 V up to 36mA, but because its only meant as a logic level and not to deliver power, the power supply chip gets very hot [29]. As there also needs to be implemented a way to power the Teensy with 5V without a need for a USB device always connected to it, a 12V to 5V transformer needs to be used anyway for the final product.

8.3 Voltage drop across the L298

The driver that has been used for the H-bridge uses the L298 chip, which uses Darlington coupled BJT transistor as the switches. The problem with this is that there is a 2 V drop between the input and the output. This limits the maximum torque and velocity that is possible to be reached, instead a H-bridge that uses MOSFET driver would have been better, as there would be almost no voltage drop between the input and the output.

8.4 Incremental encoder failure

The encoder that is mounted on the turn-table, which has been used for reading the angle of the object, has been unstable while running the test of demands. The Cobot is mounted with an incremental encoder but the zero point/reset point has been inaccurate due to drifting

which means that we cannot rely on the readings the encoder gives to the user. This inaccuracy was tested and it was found out that it was a software problem that we did not have the time to fix. As we see it, there are four ways to fix this problem:

1. **Use a gyroscope to measure the angle** We could have used a gyroscope that measures the angle of the bar, but as the bar rotates it would have had to either use wires to communicate, which limits the amount that the bar can rotate, or use a way to wirelessly transfer the data to the Teensy, which would be more expensive and require the user to recharge the transmitter.

2. **Program better software that is faster (not digitalRead) or interrupt priority** To measure the encoder pulses, we have used the Arduino function digitalRead, which is a very slow way of reading an I/O pin from the Arduino. Here we could have implemented the object encoder the same way the motor encoder is implemented, which uses a fast way of reading the pulses. This would have reduced the chance of missing pulses, but not removed it completely. It would also have been possible to program interrupt priorities, so that the bar encoder has the highest priority. Then with the Teensy used, it should have been fast enough to never miss a pulse, but it might instead miss a pulse from the motor encoder, which would not have been as important since it only needs to give an approximate value.

3. **Use a dedicated MCU that only reads the encoder** The reason why the Teensy misses some pulses, is because it also have other tasks that it needs to do, so instead a dedicated MCU could be used, which only job would be to measure the pulses from the object encoder. As long as the dedicated MCU is fast enough to keep up with the encoder, no pulses would ever be missed. Though this would also result in a more expensive solution.

4. **Use an absolute encoder** With the use of an absolute encoder, the Teensy would always be able to know where the encoder is, and there would not be the need for interrupts as it can just read the location when it needs it. The problem with this is that an absolute encoder is also much more expensive than an incremental encoder.

8.5 Missing speed controller

As the user is supposed to choose the driving velocity from a phone, the Cobot should also always try to maintain that total forward velocity at all times. A simple P controller could have been implemented, which calculates an error between the wanted velocity and the measured velocity, and then multiplies the error with a constant. As it is implemented now, there is no control whether or not the Cobot actually has the wanted velocity, and it is just assumed that it always does. This is not smart as there is no way of knowing if there is an external element that changes the correlation between the PWM scalar and the velocity. But as there were enough problems with getting the rest of the functionality for the Cobot to work, this problem had been pushed aside. And as it was originally assumed that the user would regulate the velocity of the Cobot without the need to send the velocity, but through

another means where the Cobot automatically detects the speed of the user, the user would have acted as a kind of velocity controller.

8.6 Bad implementation of gyro

In section 5.2.2 it can be read that we have realized that the gyro implementation was made in a less than optimal way. This is due to the fact that in order to calculate omz, the angular velocity of θ_{O} ($\omega_{\theta,\mathrm{O}}$) and of θ_{C} ($\omega_{\theta,\mathrm{C}}$) are needed.
$\omega_{\theta_{\mathrm{C}}}$ could have been obtained from the gyro directly, but instead it was programmed so that the Gyro class outputted the exact angle (θ_{O}) and that was then again recalculated into an $\omega_{\theta,\mathrm{C}}$. It was realized so late, that instead of changing it in the code and the report, it was noted here.

8.7 Searching angle limitation

In Section 5.1 it is explained how the angle limitation is implemented, which is set to 45°. However, the way this limit has been decided is by estimating what would be a good angle. Setting the angle limitation could have been done with more tests with various angles by making the Cobot drive. Then, figuring out what the most optimal angle would have been to make the Cobot more stable.

8.8 Ineffective use of omz

From the algorithm it is known that it should have the input omz with the unit rad/s.
In order to calculate this the following equations has been implemented.

$$\begin{aligned} \omega_{\mathrm{C}} &= \frac{\Delta\theta_{\mathrm{C}}}{\mathrm{dT}} \\ \omega_{\mathrm{O}} &= \frac{\Delta\theta_{\mathrm{O}}}{\mathrm{dT}} \\ \mathrm{omz} &= \omega_{\mathrm{C}} + \omega_{\mathrm{O}} \end{aligned} \tag{8.1}$$

In the algorithm, omz is only used in the following function call:

$$\mathrm{rotmat}((-\mathrm{dT}) \cdot \mathrm{omz}, \mathrm{roto});$$

From this it can be seen that omz is in fact used as $-\Delta\theta_{\mathrm{omz}}$ rad.
This means that instead of calculating the angular velocities, the change in angle would have been sufficient.
But as the algorithm was given to us, we did not want to make changes to it, as to when we are troubleshooting the Cobot, we could "exclude" the algorithm as long as it returns a degree that matches our simulations.

8.9 Compare HW-290 acceleration measurement with motor encoder for better results

As it is implemented now, to get the velocity the Cobot has moved, the values from the motor encoders are read. But as this does not take into account if the wheels slip, it could have been

a good feature to compare the results, so it could be known if the wheels slip, and with this regulate the inputs to the algorithm accordingly.

8.10 Overfilling the algorithm buffer with values

As the buffer has a set limit size, if the user sets the velocity to 0 while in the middle of using its autoDrive function, the code would keep on running and after some time all the values in the buffer would have been overwritten with the same value. In this case, a good feature would be to recognize when the user's velocity is zero while the change of the object's angle is zero, it would then stop the program until a new change is detected. This way the buffer would not get overwritten, and when the user then decided to continue, the Cobot would not have forgotten the user's previous points.

8.11 The demands listed for the product are not proper technical demands

Most of the demands listed in the Chapter 4 are made from the viewpoint of a customer who is interested in a product. It would have been better to write new technical demands after the Design Section that if complied with, would be consider as the original demands tested. This would have made the test of demands much more accurate, so there was no need to rewrite the tests with limitations.

8.12 The state of the Cobot

Here we will make a note about the Cobot's state just before delivery of the report the 19th of December 2020.
The first thing we will go into is an informal test made in early December, as this contains the only recording of the Cobot driving.

Informal walking test

In early December an informal test of the Cobot was made (test 1). This was filmed, and can be seen in the Github repository under [20]:
master/Video_of_Cobot/test_1.mp4

In this informal test it can be seen that the Cobot is driving unstable, but is still able to follow the user in front.
Since test 1 we changed a lot of fundamental calculations, because they where later found to be wrong. The 16th of December 2020 it was again driving in the same way with about the same amount of unsteadiness. This is strange as a lot of the software was completely changed, including some of the inputs to the algorithm which was found out to be wrong.
But as it finally worked again the 16th late in the night, we began some of the tests of the demands on the 17th.

Wheel breakdown

Initially it went well, as some tests were completed, but then we came to the tests for weight on the Cobot, where the 3D printed wheels broke down. So in the night between the 17th and the 18th we drew and printed some new parts for the wheels, which fit after some minor

changes. Then we began the testing again. Beginning these we found out that some major errors were introduced in the software. These will be explained in more detail in the next section.

Software not able to continuously run

When it comes to the code, it was not perfectly successful. In the code, the main function's role is assigned to the loop function. There are a lot of functions for making the Cobot drive, and each function works well when it was conducted on its own. However, on the 18th December 2020, when all of the functions were conducted simultaneously in the loop function, the loop function broke down and made the Teensy stop running after one or two cycles in the loop. After spending the whole day trying to fix this error, it was still unclear where it came from and what the solution would be.
Conclusively, the main problem is that when all the functions are gathered in the loop function, the Teensy is not able to run the software. Before the exam date, continous work will be put into trying to get the Cobot to work to be able to show a working example of it.

Appendix A

Interview with carpenter

On September 10, an interview was made with a self-employed carpenter, with the company existing of himself and three other employees. They have worked with many different sorts of tasks within the construction industry, making them ideal to interview about difficulties regarding heavy lifting in their line of work. The interview was conducted with the company called JK Byg, of which master carpenter Jesper Kristensen was the person who was spoken with.
A full transcript of the interview is listed below.

Interviewer: Hi Jesper. First of all, thank you for taking the time to do this interview with us. We are a group of electronic engineering students from the University of Aalborg, and we would like to ask you a few questions about your job, but first, could you please give us an short introduction of yourself?

Jesper: Sure. My name is Jesper Kristensen, and I am in charge of a carpentry company called JK Byg in Midtjylland, where I am charge of my three employees.

Interviewer: Can you give us any insight of the tasks you are usually working with? Like, are you primarily building new houses, bigger construction projects, or something else?

Jesper: Yes. We actually have a lot of variety, as we both build houses, perform many renovations and have also been in charge of some bigger commercial constructions. We also often work as rental workers, where we work for another carpentry company with larger projects - we actually spent many months working for McDonald's in the last couple of years, working together with another carpentry company, both renovating and building new restaurants all over Denmark.

Interviewer: Sounds very interesting, it seems like you have a lot of experience with various sorts of tasks in the construction industry. The purpose of this interview is actually about heavy lifting in the construction industry, as we wish to help construction workers with the difficulty of all this heavy lifting. Do you often experience problems regarding heavy lifts?

Jesper: Yes, we often experience that. We frequently have to carry heavy objects like glulam and I-beam, which is troublesome for us, as they are often both long and heavy. For these kind of problems, we have to be two men, as we cannot lift those objects alone given their length.

Interviewer: Oh okay. Does that mean that when you are working alone, an additional worker has to come by when you need to help these objects? And is it something that happens regularly?

Jesper: Well, at first, when I did not have any employees, it was a big problem for me. Now that I have 3 employees besides myself, it does not happen as often, because I can manage my workers to work in teams of two. But it happens frequently in smaller companies where it is not possible to work in two-man teams, which takes up a lot of time if not planned correctly.

Interviewer: Which objects causes you the most trouble lifting?

Jesper: That would definitely be glulam and I-beams like mentioned before. Objects like these, that are both long and heavy, are very difficult to lift around as only one man, but even as two men they can be troublesome. Generally, it is only the long objects that are problematic for us, like above three meters long. These are both heavy, taking a toll on the body, but can also be difficult to maneuvre around when approaching corners and such.

Interviewer: Okay, so you already have a solution for objects that are heavy, but not long?

Jesper: Yes we do. If they are not long, we can use either a window trolley, sack cart or a pallet lifter, and just drag them onto these.

Interviewer: Alright. What if you have long objects, do you have any solutions for that?

Jesper: Not really, most of the time we need to be two men for lifting heavier loads. But if it only has to be lifted for a short distance, we can sometimes get away with using a furniture slider.

Interviewer: Oh okay. So how often would you say that you need to do two-man jobs on an annual basis? We know that this question might be difficult to answer.

Jesper: That is very difficult to answer, as it is hard to say. But in case of looking for advantages of discarding the need for two-man jobs, it would definitely be useful, as it provides the ability to do more jobs as one man. It would also mean that their bodies will be spared, meaning that the workers will be able to work for more years, which would be beneficial to everyone.

Interviewer: Good point. Do you believe that a product, that makes these two-man lifts possible to perform with as only one man, would be appealing to companies in the construction industry?

Jesper: Definitely. As of now, I have not heard of any product that can accomodate this problem. I think that many larger companies would be interested in buying a such product, especially companies that are building multi-storey buildings. If it can solve the problem with an appropriate pricing, I think it could be a succesful product.

Interviewer: That sounds great - what do you think would be considered an appropriate pricing? Like, what would you be willing to pay for it?

Jesper: I would say like 75.000-100.000 DKK would be a price that most companies would be willing to pay for it, including myself, if the product solves the problem in an efficient matter.

Interviewer: Okay, thank you for that. Do you have any additional desires for the product, to make it more appealing?

Jesper: Hm, yes I do. It would improve the product if it could be implemented with a four-wheel steering system, and the ability to lift vertically up to a height of three or four meters. And if it is possible, it would be great if the bearing part can be interchangeable with a plate, so that it can also carry drywall and other big rectangular objects. Finally, it would also be beneficial with the ability to remotely control it.

Interviewer: Great, we will try to take that into account while designing our product. For what cases do you think the remote control could be used, in case we will have to consider those in our design?

Jesper: I think that it will be a useful feature for the adjustment of the object when it has reached its final destination, so that it does not have to be moved manually. Also, for transporting plates of drywall or similar objects, the ability to remote control it in combination with the interchangeable bearing head, would make it very helpful for lifting such objects in general.

Interviewer: Okay, thank you very much. I think that we have everything we need now. Thank you for taking the time to participate in this interview.

Jesper: You're welcome. Let me know if you have any further questions, and how the final product turns out - it seems like you have an interesting case.

Interviewer: We will definitely do that, thank you. Have a nice day.

Jesper: You too. Bye bye.

Appendix B

Prove for error margin equation

In this Appendix, it will be explained how the equation for error margin was developed.
In developing this equation it is assumed that the symmetric scenario is critical. Where the width of the hallway on both sides on the corner is the same. This has not been further investigated.
Because of this symmetrically, it does not matter if the object is carried, from the bottom-up going right, or from the right going down. Referring to Figure B.1.
For this scenario a small simulation in Geogebra was setup to see at where the critical point in the lift is. Here it was found out that the critical point is after turning 45°, as it is done on Figure B.1.
On Figure B.1, the sketch used to develop the equation can be seen.

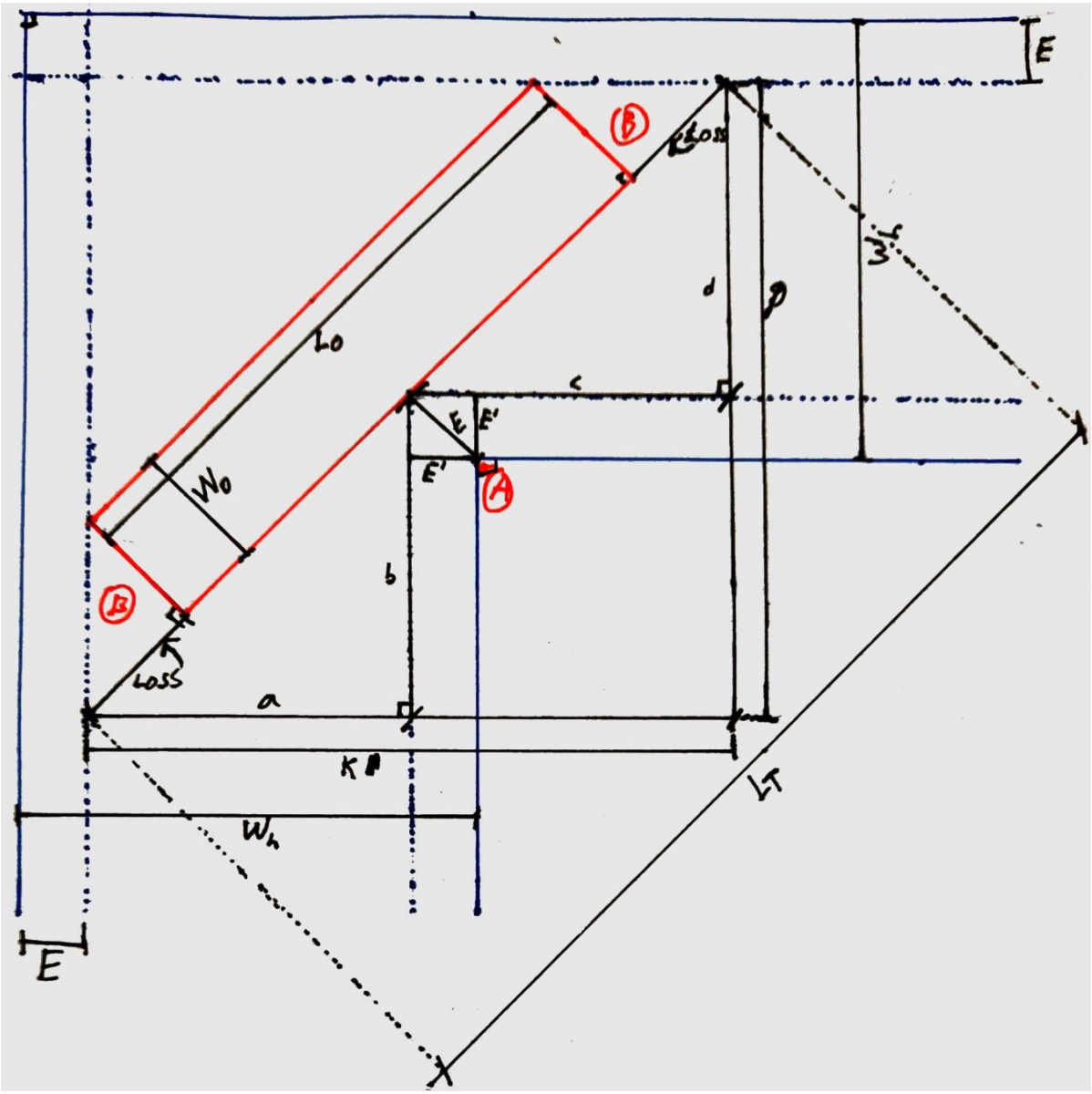


Figure B.1: Equation for error proof

First the variables will be defined.
Known variables:

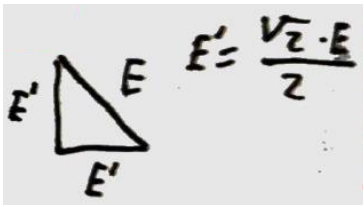
- L_O is the length of the carried object.
- W_O is the width of the carried object.
- W_h is the width of hall way.

Calculation variables:

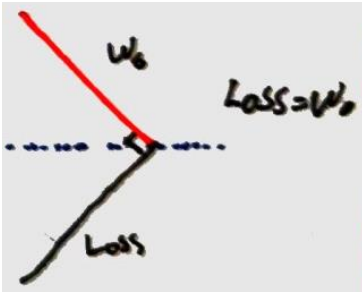
- E′ is a variable related to E. The definition can be seen on Figure B.2a
- Loss is the in object length, caused by the width in the object. The definition can be seen on Figure B.2b
- a,b,c,d,k,p,m variables used to calculate E.
- L_T is the length of the object if there was no Loss.

Output from the calculations:

- E is the error displacement margin.



(a) Relation (A) for E′ and E



(b) Relation (B) for Loss and W_O

Figure B.2: Graphical relation between for (A) & (B)

Because of the symmetry in the scenario, a few variables can be substituted for m.

$$\begin{aligned} W_h - E - E' &= a = b = c = d \\ &= m \end{aligned} \tag{B.1}$$

Now a formula for k and p can be defined.

$$\begin{aligned} k &= a + c = 2 \cdot m \\ p &= d + b = 2 \cdot m \end{aligned} \tag{B.2}$$

From the sketch it can be seen that L_T can be calculated be using Pythagoras. And there after be simplified, as below

$$\begin{aligned} L_T &= \sqrt{k^2 + p^2} \\ &= \sqrt{(2 \cdot m)^2 + (2 \cdot m)^2} \\ &= 2\sqrt{2} \cdot m \end{aligned} \tag{B.3}$$

The length of the object can also from be express simple from the sketch as bellow. Followed

up by substitution, so L_O is only dependent on W_h, E and W_O.

$$\begin{aligned} L_O &= L_T - 2 \cdot Loss \\ &= 2\sqrt{2} \cdot m - 2 \cdot Loss \\ &= 2\sqrt{2} \cdot (W_h - E - E') - 2 \cdot Loss \\ &= 2\sqrt{2} \cdot (W_h - E - E') - 2 \cdot W_O \\ &= 2\sqrt{2} \cdot (W_h - E - (\frac{\sqrt{2} \cdot E}{2})) - 2 \cdot W_O \end{aligned} \tag{B.4}$$

Now E can be isolated.

$$E = \frac{(-L_O - 2 \cdot W_O) \cdot \sqrt{2} + 4 \cdot W_h}{4 + 2 \cdot \sqrt{2}} \tag{B.5}$$

Appendix C

Prove for speed limit equations

In this Appendix, the functions describing the Cobots movement going towards a 90° turn will be derived.
The Cobot has to adjust its speed according to the user walking in front of the long object. This is needed in order to maintain the distance of the object to the user in front. When the user and the Cobot are traveling in the same direction, the Cobot should have the same speed as the user. To do this it's maximum speed and acceleration should be picked according to the user.
But to choose the maximum speed and maximum acceleration, another scenario has to be analysed.
This is because the speed changes when the user takes a turn, and the worst case scenario is on a 90° turn.
To find the relationship between the users speed and the Cobots around a 90° turn. The users position is defined as a vector function of time and the position of the Cobot as the intersect with the Y axis for a circle with the length of the object as radius. In this first scenario the Cobots acceleration and speed is not limited. From these the velocity and acceleration for the Cobot around a 90° corner can be derived. The user makes the turn to $\mathrm{t} = 0$.
To simplify these calculations, only the users path in the x-direction is taken into account, since his y-position does not change for $\mathrm{t} > 0$. The path of the user for $\mathrm{t} > 0$ is described as the vector from P2 to P3 on Figure I.1. The vector from P1 to P2 is the users path for $\mathrm{t} < 0$, and thereby where the Cobot should travel.

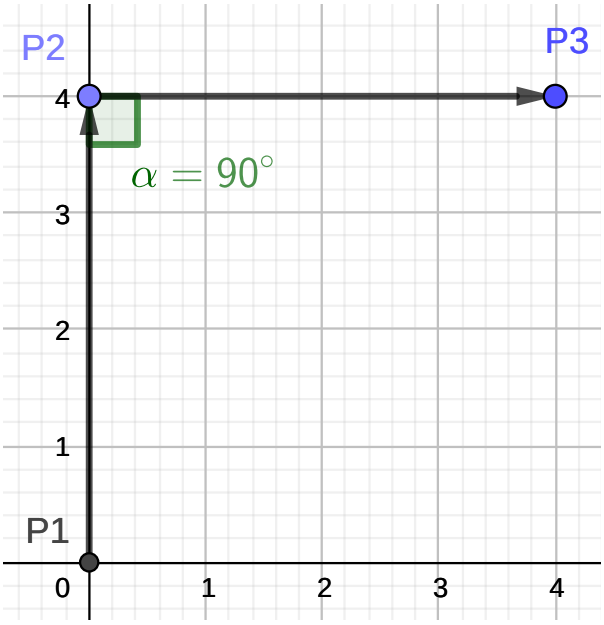


Figure C.1: Describing the users path with vectors.

The users start position is defined as user_0. And is [0,4] because the user is at P2 at $\mathrm{t} = 0$.

$$\mathrm{user}_0 = \begin{bmatrix} 0 \\ 4 \end{bmatrix} \tag{C.1}$$

The direction for the users path for $\mathrm{t} > 0$ is defined with the direction vector, **dir**.

$$\mathrm{dir} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{C.2}$$

The users x position can now be as in Equation C.3. The only unknown here is the variable speed. This is set to the average speed of a person from the age 20 to 70 which is 1.37 $[\frac{\mathrm{m}}{\mathrm{s}}]$ [19].

$$\mathrm{user}_\mathrm{x}(\mathrm{t}) = \mathrm{user}_0[1] + \mathrm{t} \cdot \mathrm{dir}[1] \cdot \mathrm{speed}; \tag{C.3}$$

Since $\mathrm{user}_0[1]$ is 0 and dir[1] is 1 in this scenario, $\mathrm{user}_\mathrm{x}(\mathrm{t})$ can be simplified to Equation C.4.

$$\mathrm{user}_{\mathrm{x,sim}}(\mathrm{t}) = \mathrm{t} \cdot \mathrm{speed}; \tag{C.4}$$

By using pythagoras the Y-position of the intersect can be calculated. This is done in Equation C.5, where L_obj is the length of the object. If nothing else said $\mathrm{L}_\mathrm{obj} = 4[\mathrm{m}]$.

$$\mathrm{int}_\mathrm{y}(\mathrm{t}) = \sqrt{\mathrm{L}_\mathrm{obj}^2 - \mathrm{user}_{\mathrm{x,sim}}(\mathrm{t})^2} \tag{C.5}$$

The speed for the Cobot is known by differentiating $\mathrm{int}_\mathrm{y}(\mathrm{t})$ as in Equation C.6.

$$v_{int,y}(t) = \frac{d(int_y(t))}{dt} = \frac{speed^2 \cdot t}{\sqrt{L_{obj}^2 - speed^2 \cdot t^2}} \tag{C.6}$$

The acceleration can be expressed by double differentiating $int_y(t)$ as in Equation C.7.

$$a_{int,y}(t) = \frac{speed^2}{\sqrt{L_{obj}^2 - speed^2 \cdot t^2}} + \frac{speed^4 \cdot t^2}{(L_{obj}^2 - speed^2 \cdot t^2)^{\frac{3}{2}}} \tag{C.7}$$

By looking at the denuminator from $v_{int,y}$ it can be seen that there are two first order poles at

$$t_{poles} = \pm \frac{L_{obj}}{speed} \tag{C.8}$$

The same goes for poles for the acceleration. From this it can be seen at the Cobot will have to reach infinite speed and infinite acceleration if the user takes a 90° turn and keeps a constant speed.

Appendix D

Analysis of steering techniques

In this section, various methods for steering a robot in environments like a construction site will be analysed. To make it able to turn around sharp corners, it should be possible to make a sharp turn with a minimal deviation of its position. Additionally, it should be able to handle heavy loads.

D.0.1 Differential drive

The differential drive is a simple and commonly seen way of controlling robots. They are typically using two separately motor-controlled wheels in the back and a castor wheel in the front. The front castor wheel is for maintaining equilibrium and is simply following the direction determined by the two rear wheels. These two rear wheels can control the direction of the robot, by letting the motors set a specific velocity difference between the two wheels, making it turn [37].
The advantage of using this method of steering is the simplicity of the design and control algorithm, since there is only the need for specifying the velocity for each of the motors according to the turning angle. The disadvantage of this will be that it has a reduced precision, meaning that it can be problematic to follow a straight line. This lack of precision is due to the different number of rotations of each of the turning wheels. To accommodate this problem, a correction factor can be implemented, although imprecision may still occur.

D.0.2 Tricycle drive

The tricycle drive is composed of three wheels, of which one is in the front and two in the back. With this method, the front wheel acts as a steering wheel, while the two rear wheels are attached to a fixed axle, meaning that these can only go forwards or backwards [37]. This type of steering is the same as seen in four-wheeled automobiles, except this one is using one steering wheel in the front instead of two, at the price of reduced stability.
The advantage hereof will be the simple implementation, as the two rear wheels only have to go forwards or backwards, while the front wheel will be for steering.
The disadvantage of tricycle drive steering is that the robot cannot perform 90 degree turns without driving a bit forward at the same time.

D.0.3 Independent drive

The independent drive is a method where four wheels are used, that each can be controlled individually. By having the possibility to both steer and drive with each of the four wheels, the robot will be able to make turns without moving out of place [37].
The advantages of this method will be the ability to turn in place, making all kinds of turns possible. Moreover, this method is superior to the other methods when meeting uneven terrains, as the robot will be able to maneuver around even if one wheel gets stuck by an object. The disadvantages of this will be the largely increased complexity, as each wheel needs to be steered independently, which are demanding complex algorithms to solve. This

complexity also means that since each wheel will need a steering control, the development of this steering will require more materials, thereby increasing the cost.

Appendix E

Wireless Communication

To be able to control the robot remotely, a type of wireless communication needs to be chosen. There are many different possible frequencies and protocols that can be used, where each one has its own strengths and weaknesses. In this section, a selected few will be researched, and their pros and cons will be compared, to figure out which technology is best suited for this use case. There isn't a need for high data rates, as the only data that will be sent are small control messages, whereas more emphasis will be put on the power usage and the range of the technology.

E.0.1 ISM (Industrial, Scientific and Medical) - 2.4 GHz

The Industrial, Scientific and Medical band, also called ISM band is a range of frequencies that are reserved for internationally use for any purpose under the formerly named industries. This means that it is allowed to make a product that uses these frequencies all over the world, without breaking any laws, and that they are free to use. The ISM band includes a wide variety of frequency band that are free to use, but for this project we are focusing on the 2.4 GHz band (2400 - 2500 MHz), as this is a commonly used bandwidth and a fast growing use for this band have been for short range, low power wireless communications systems [38] [39].

Bluetooth Low Energy (BLE)

Bluetooth is a type of wireless communication that operates in the ISM band. Bluetooth was originally designed for continuous, streaming data applications. That means that it can exchange a lot of data at a close range. For this reason, it is commonly used for consumer product such as phones, as it enables the ability to transmit large amounts of data over a relative short range. With the introduction of Bluetooth 4.0, Bluetooth Low Energy (BLE) was introduced. It sacrifices the amount of data that can be sent, for a smaller power draw, which extends the battery life by months to years depending on the usage. With version 5.0, the low power features were increased to allow a small sacrifice of range to increase data transfer, or increase the range at the cost of data transfer. In a free view, with no obstacles the range of Bluetooth 5.0 can be up to 400m, depending on the data transfer required and design of the chip/board [40].

LR-WPAN (IEEE 802.15.4)

IEEE standard 802.15.4 intends to offer the fundamental lower network layers of a type of wireless personal area network (WPAN) which focuses on low-cost, low-speed ubiquitous communication between devices. It can be contrasted with WiFi, which offers more bandwidth and require more power. Also, IEEE 802.15.4 is a technical standard which defines the operation of low-rate wireless personal area networks (LR-WPANs). It specifies the physical layer and media access control for LR-WPANs. [41]

nRF24L01+

The NRF24LO1+ is a chip that also uses the 2.4 GHz spectrum, and is very similar to Bluetooth. It is commonly used by hobby enthusiast for wireless projects, mostly because of its low cost and high data rate/range. It is possible to acquire for as little as 20 DKK [42]. It can be configured for up to 2Mbps transfer rates, and with a range of up to 800m in free space depending on the data rate. It uses Gaussian frequency-shift keying (GFSK) to transmitting data, which reduces sideband power and interference from neighboring channels. It also uses Enhanced ShockBurst Protocol which allows for variable payloads, payment ID and package acknowledgement for a more reliable connection. [43]. Which makes it a very reliable means of transmitting data, for a cheap price.

ANT+

ANT represents ultra-low-power, short-range wireless technology designed for sensor networks and similar applications, and uses the 2.4 GHz ISM band. ANT can be configured to spend long periods in a low-power standby mode, wake up briefly to communicate (when consumption rises to a peak of 22mA (at -5dB) during reception and 13.5mA (at -5 dB) during transmission) and return to sleep mode. Thus, each ANT channel consists of one or more transmitting nodes and one or more receiving nodes, depending on the network topology. Any node can transmit or receive, so the channels are bi-directional. ANT+ is a relatively new addition to ANT. This software function which can be added to the base ANT protocol provides interoperability in a managed network. It facilitates the collection, automatic transfer, and tracking of sensor data for monitoring all involved devices and nodes. It is also designed and maintained by the ANT+ Alliance which is managed by ANT Wireless, a division of Dynastream Innovations owned by Garmin. However, most of the products that are using ANT+ or ANT are related to the sport industry, so it is not suited to be used to this project. Moreover, most of the products cannot communicate for a long range, but only 5-10 meters. [44] [45].

E.0.2 433 MHz radio communication

The 433 MHz spectrum (433.05 - 434.79 MHz) is also a part of the ISM band, but is only freely available in Europe, Africa and Russia. But as the product is not viewed as being marketed outside of any of those regions, it would not become a problem to use these frequencies [39]. For the 433 MHz spectrum there isn't a set way to transfer data, all you need is a receiver and a transmitter. They way you encode the data is up to you, but a commonly used way is by using ASK (Amplitude Shift Keying), which makes it simple to transmit data by using the change in amplitude to transfer the data. The good thing about this is that it is usually cheaper to make, as it reqires less bandwidth, but the con is that it is susceptible to noise which comes from other devices or the background. Another way is to use FSK (Frequency Shift Keying) which changes the frequency of a signal to send data. The advantage of this is much lower interferrance from noise, but as it changes the frequency it is usually more costly [46].

E.0.3 Conclusion on wireless transfer

As the Cobot is meant to only use wireless transmission to receive direction from a remote control, the amount of data that will be sent is not of importance, as every technology can send more than enough data to transmit a direction. Regarding transmission range, as the idea is to remote control it, the controller would be in looking distance of the robot, so there is no need for 100+ meters of range. But to make sure that interference and physical blocking (could be from other people, or objects in the way) do not create an unstable transmission, the minimum range of the sender should be 10+ meters. As far as battery life goes, the fewer times the batteries need to be changed the better, as the remote is a stand alone product, and need to be recharged before it can be used.
To better help compare the different technologies, a table has been created which summarises the different specifications of the technologies. It can be seen in Table E.1

Technology	Bluetooth Low Energy (BLE)	LP-WPAN	nRF24L01+	ANT+	433 MHz
Product	BL651	DIGI XBEE 3 802.15.4	nRF24L01+	D52QD2M4IA	ALPHA-TX433S
Range	<100 m	<1.2 Km	<800 M	<5-10 M	<300 M
TX Power Usage*	6.9-23 mW	283.5-486 mW	7-11.3 mW	15.9-49.8mW	40-75mW
Standby Power Usage	4.95 µW	4.2-7.2 µW	26 µA	Unknown	9.9 µW
Data Speed	<2 Mbit/s	<250 Kbit/s	<2 Mbit/s	<60Kbit/s	<115Kbit/s
Cost	32.13 DKK	103.17 DKK	20 DKK	98.39 DKK	47,62 DKK

Table E.1: Information about the different wireless technologies [47],[48],[49], [50], [51],[52]
* Depends on range

Bluetooth Low Energy (BLE) has been chosen as the preferred technology to wirelessly transfer data. It has been chosen because of its low power usage, great range, and it is an already established and commonly used technology. This gives the possibility to remote control it from a phone or other remote devices, and also update the firmware of the robot by the use of an app. This makes for a simple interface to deploy potential updates. For this prototype a phone has been chosen as the communication device, as this removes the need to design a dedicated remote controller.

In the next section, the Bluetooth standard will be analyzed to get a better understanding, so it can be implemented.

Appendix F

Bluetooth Low Energy

In this appendix some of the core specifications for BLE will be described.

F.1 BLE Architecture

BLE is built in layers and has an architecture as seen on Figure F.1. Since the whole core specification of Bluetooth BLE 4.2 is a 191 pages long paper, only the Physical layer will be investigated and explained [53].

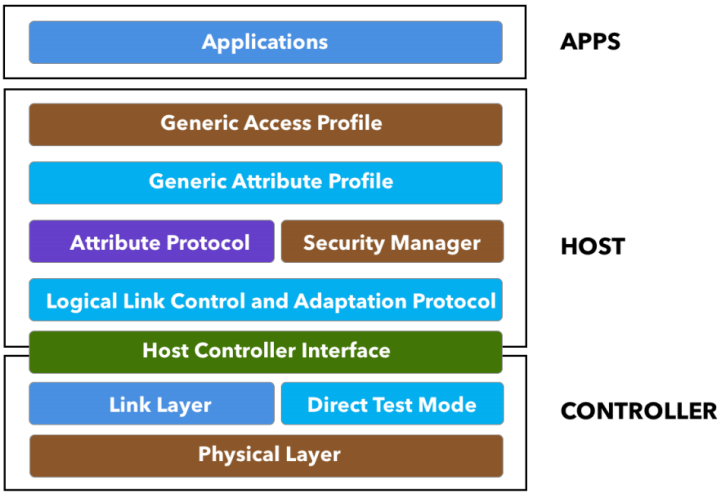


Figure F.1: Architecture of BLE [54].

F.1.1 Physical layer

BLE operates in the frequency spectrum of 2.400-2.4835 GHz (ISM band) and consist of 40 channels [54].
The channel base frequency is defined by Equation F.1 [53].

$$\mathrm{Ft} = 2402\mathrm{MHz} + \mathrm{k} \cdot 2 \tag{F.1}$$

Where:

Ft	The channel base frequency	[Hz]
k	Is the channel number and goes in the interval 0-39	[-]

BLE uses 2-level Gausin Frequency Shift Keying (GFSK) to modulate the data and has a data-rate of 1 Mbps [53] [54].
GFSK encodes data as a series of frequency changes around the base frequency Ft. Since the data is sent as frequency changes it is relatively immune to noise, as noise usually change the amplitude of a signal. [55].

The lower and upper transmitter frequency can be determined by Ft+fd and Ft-fd. Where the variable "fd" is the frequency deviation. A digital "1" will be modulated as $\mathrm{Ft} + \mathrm{fd}$ and a digital "0" will be modulated as $\mathrm{Ft} - \mathrm{fd}$. The formula for fd can be seen in Equation F.2. According to the BLE standard, the modulation index "h" shall be between $0.45 - 0.55$ [53]. A large h gives a bigger bandwidth, than a small h. Where a large or a small h is good depends on the environment where the device is set. As a large bandwidth will interfere more with neighboring channels, but there will be less internal interference (between the two frequencies), and vice versa.

$$\mathrm{fd} = \frac{\mathrm{h} \cdot \mathrm{Ft}}{2} \tag{F.2}$$

Where:

fd	The frequency deviation	[Hz]
Ft	The channel base frequency	[Hz]
H	The modulation index	[-]

On Figure F.2, the definition of GFSK modulation parameters are defined.
On the figure it can be seen that the change from Ft+fd to Ft-fd is a smooth transition. It is done by applying a Gaussian filter to the shape of the radio pulse. This Gaussian filter ensures that the change in frequency does not happen instantaneously. This is the Gaussian part of G-FSK. The smooth transition gives a narrow spectral band. Which is preferred as it reduce interference with neighboring channels [55].

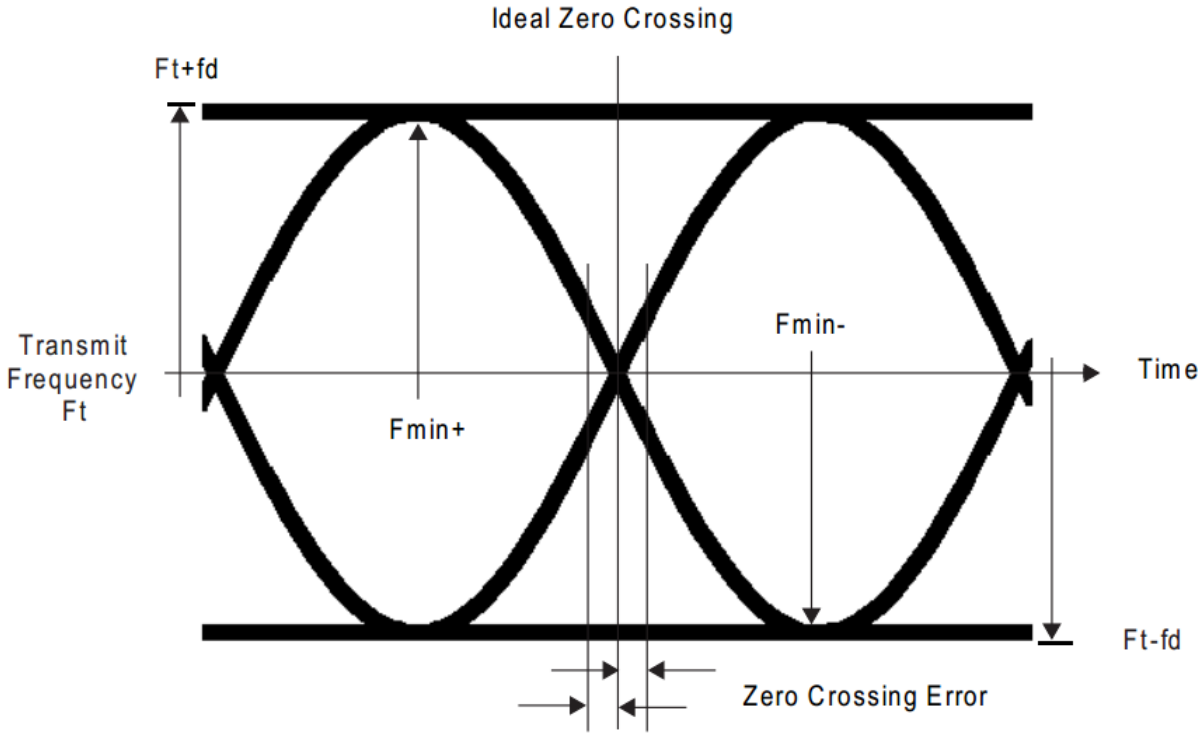


Figure F.2: GFSK parameters definition [53].

To improve reliability, BLE uses Frequency Hopping Spread Spectrum (FHSS). This is done so bit errors are spread more randomly. The reason why this is done becomes clear when Figure F.3 is investigated. The figure shows the signal strength at a receiver antenna according to distance to the transmitter antenna. The deep negative spikes in signal strength comes from reflections from the earth being 180 degrees out of phase, and thereby canceling the signal [56].
In the scenario where the distance between the user and the Cobot slowly changing, a lot of bits will be lost at these negative spikes, if it was not for FHSS. By constantly changing the channel, these errors will be spread more evenly and it is possible to correct them.

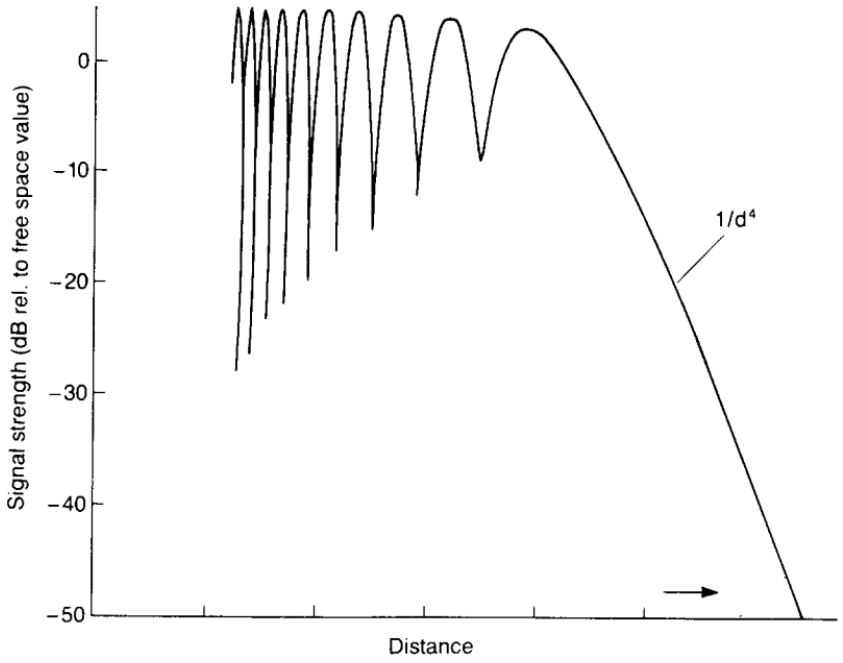


Figure F.3: Variation of signal strength with distance in the presence of a specular reflection [56].

Appendix G

Technologies for angle measurements

The next few sections, will go through a few different technologies to measure angles or change in angles. This done select the best possible technology to each of the two angles (θ_O, θ_C).

G.1 Tachometers/rotary encoders

Tachometers/rotary-encoders are electromechanical modules that are able to measure the change in an angle by generating and observing a wave [57].
There are two main categories for rotary encoders, which are incremental encoders and absolute encoders. This short general comparison between incremental and absolute encoders will be made with the requirements:

- They have the same resolution.
- They are able to determine direction.

Absolute encoders has a unique point for each angle, which always allows it to know which angle it has, even after a restart. The trade-off for this is increased complexity and cost [57].
Incremental encoders counts the number of rising or falling edges of the square wave to determine the change in angle. As the name suggests, it keeps track of angular change by incrementing a counter. This makes it simpler and thereby cheaper to make than the absolute encoder. The incremental encoder has the disadvantage that it needs a "zero point". This zero point needs to be fixed point, from where the controller can know the encoder's exact position [57].

Both incremental and absolute encoders can be made in a few different ways. Three widely used ways are optical, magnetic and contact encoders.
Optical and magnetic are non-contact types which therefore are not so susceptible to wear and tear. On the other hand, optical encoders can get faulty readings from dust, whereas magnetic encoders can get interference from nearby magnetic fields. The third widely used option is a contact encoder. Here a brush or pin sensor is employed on a disc, thus generating a signal. This type is limited by bridging in disc segments and wear of the contacts [57].

All of the encoders from above has the disadvantage that their resolution is finite. This is not the case for the potentiometer as an encoder.

G.2 Potentiometer

A potentiometer is a possible solution for measuring the angle between the robot and the object. A potentiometer is an adjustable resistor meaning that the resistance is variable[58]. With a rising and falling resistance, the voltage would be changed as well. Knowing the voltage change caused by the potentiometer could be converted to a degree and thereby a possible angle. The potentiometer has an infinite resolution because of the never ending

decimals from the analog signal and therefore no fixed points, which make the potentiometer more accurate than other discrete. Even though this idea seems reasonable, there may be some flaws, especially in the long term. The potentiometer is a mechanical component that over time will be worn. Furthermore, dust and water may get in contact with the component due to the surroundings and therefore shorten the lifetime and accuracy of the Cobot.

G.3 Gyroscope

A gyroscope is a device that uses the gravity of the earth for orientation [59]. The gyroscope measures the rotational velocity of the sensor by measuring the rate of change of the angular position over time along the X, Y and Z-Axis [60].

G.4 Accelerometer

The accelerometer on the other hand measures the static and dynamic force of the acceleration. The static force is Earth's gravitational force, and the dynamic force is force caused by things such as vibration, movement and others [61]. The accelerometer is often used for tilt measurement. It can either measure tilt by using one, two or three sensitive axes X, Y and Z. When measuring the acceleration with two or more sensitivity axes the accuracy is equally divided on all angles [62] [63]. The sum of the acceleration on the accelerometer is 1g in a steady state as shown in Equation G.1.That is when the Z-axis is pointing upward/downward, which means the gravitational force is perpendicular to the Z-axis and it has no effect. The "g" in this case stands for the gravitational force which is $9.82\frac{\mathrm{m}}{\mathrm{s}^2}$ [62].

$$\sqrt{\mathrm{Ax}^2 + \mathrm{Ay}^2 + \mathrm{Az}^2} = 1\mathrm{g} \tag{G.1}$$

By using sensor fusion the accelerometer can be combined with a gyroscope. This combination improves the accuracy of the orientation on the plane ground because the values of rotation, acceleration, tilting and angle are taken into account [64].

G.5 Digital compass (magnetometer)

A magnetometer is able to measure strength and direction of magnetic fields. From this a true absolute direction can be derived, and thereby a digital compass. These are very susceptible of interference from other magnetic sources. This is because the strength from the earth's magnetic field is roughly 0.6 Gauss in contrary to a typical refrigerator magnet which is around 50 Gauss [65] [66]. Because of this, the design and placement of such a device is very important to take into account, for it to be useful.

G.6 Conclusion on angle measurements

The measurement of the two angles (θ_C and θ_O) are two different scenarios. Because in the scenario for θ_O the two elements that should be measured (Cobot and Object) are not moving relative to each-other, but they are for θ_C, as the Cobot is moving on the plane. Because of this there are picked two different technologies for the two scenarios.

G.6.1 Measurement of angle θ_C

Since the Cobot is moving on the plane, a fixed angle measurement is undesirable. This makes an potentiometer or a encoder undesirable.
As final version of Cobot is targeted at construction sites, it has to be robust against multiple interference, such as magnetic interference from large metal objects, nearby heavy electrical machinery and also dust. Because of this, the optimal solution for angle measurements and thereby navigation might not be a single technology.
With that said, a device such as the magnetometer is not optimal, since it is very difficult to design in a way so it is not disturbed by surrounding magnetic interference. The Cobot does simply not need its exact direction according to earth, and is thereby discarded.
This leaves the use of a gyroscope and an accelerometer. This is practical as many modules combine both technologies because they both have a tendency to drift. Such a module is GY-87, this module has an on-board DMP that outputs an angle from both the accelerometer and the gyroscope [23] [36]. Because of this, the GY-87 is chosen to measure the angle θ_C.

G.6.2 Measurement of angle θ_O

The turning axis of the object is following the axis of the Cobot and thereby abilitating a fixed measurement.
For stability reasons, the angle between the Cobot and the object which is carried, should be measured by either an encoder or a potentiometer. As the potentiometer wears out a lot faster over time than encoders, it leaves the encoder as the best option.
To decide between the absolute and the incremental encoder, the question of accuracy versus precision becomes apparent.
Since the technical demand for measuring this angle is a precision margin of "< 1.877°" the encoder needs to have a high precision. Because of this a incremental encoder is picked. The picked encoder is one from Sparkfun with 1024 pulses per revolution (P/R) [22]. The 1024 P/R gives a precision margin of 0.35°.

Appendix H

Software

This appendix will document different parts of the software that has been left out of the main report, but is still essential for the Cobot.

H.1 Gyro

The class Gyro is made to simplify the methods that interact with MPU6050 (hereafter MPU) and get the angle θ_C which is the angle of the Cobot.
Using the sub-library MPU6050 from I2Cdevlib, a class is available to communicate with the MPU6050, and therethrough the MPU [24]. The class available from the library is called MPU6050 (hereafter MPU6050).

H.1.1 UML diagram

A UML diagram of the class Gyro and the clases it is dependent on, can be seen on Figure H.1.

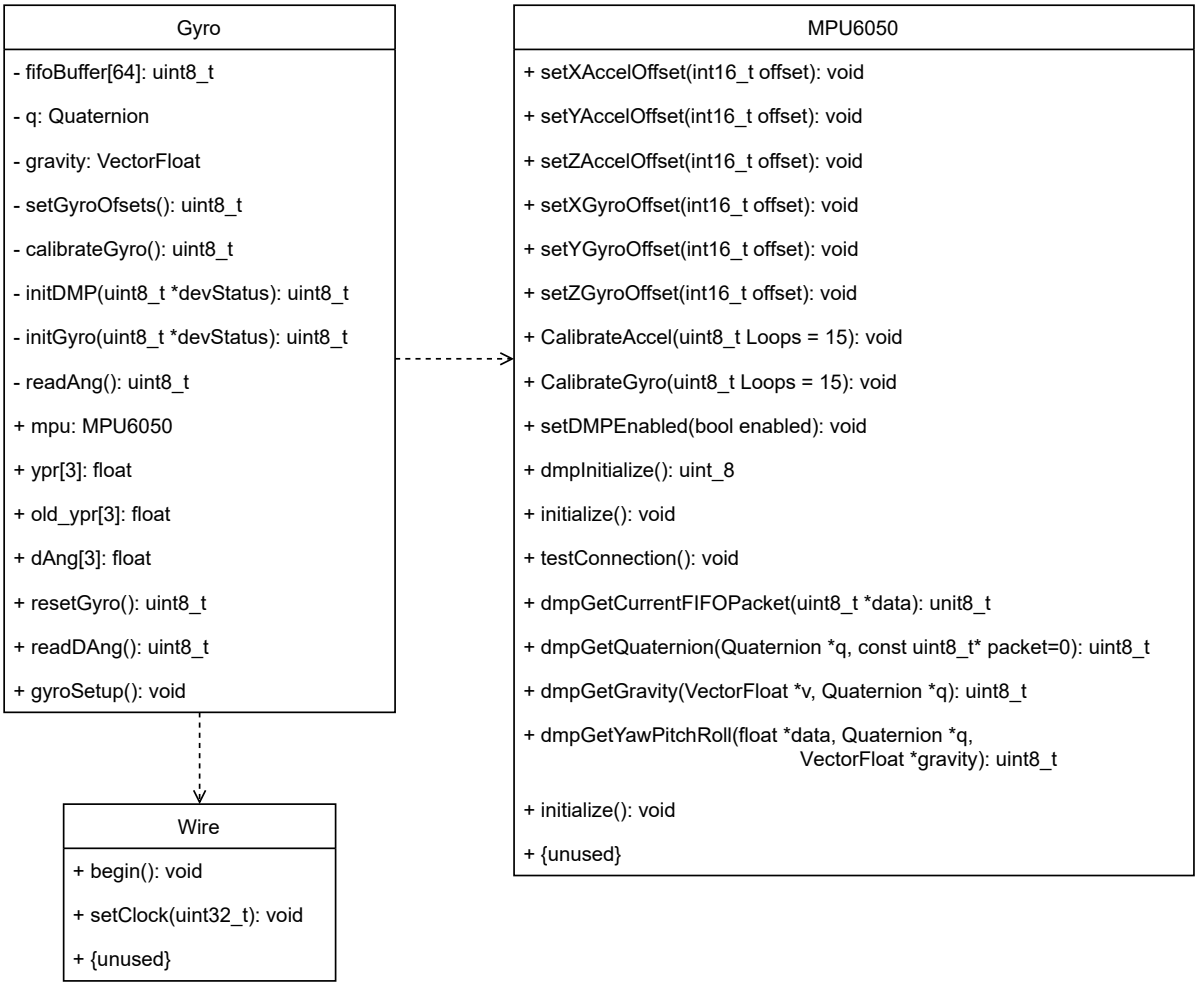


Figure H.1: Class diagram of Gyro.

Gyro has three public methods made to interact with the MPU. These are gyroSetup(), readDAng() and resetGyro(). Which will be explained in the following subsections. Below the explanations, a flowchart of each method is shown. Besides the public methods the private method readAng() will also be explained here.

H.1.2 gyroSetup()

This method is used to initially setup I2C communication to the MPU.
This method uses methods from Wire to initiate I2C communication.
Afterwards it initializes the MPU and the onboard DMP, using methods from MPU6050 class.
The flowchart for this method can be seen on Figure H.2a.

H.1.3 readAng()

This method updates the class variable ypr with new data from the MPU, if there is any. The flowchart for this method can be seen on Figure H.2b.
To do this it first checks if there is a new packet from the MPU. If there is, then it saves it. In

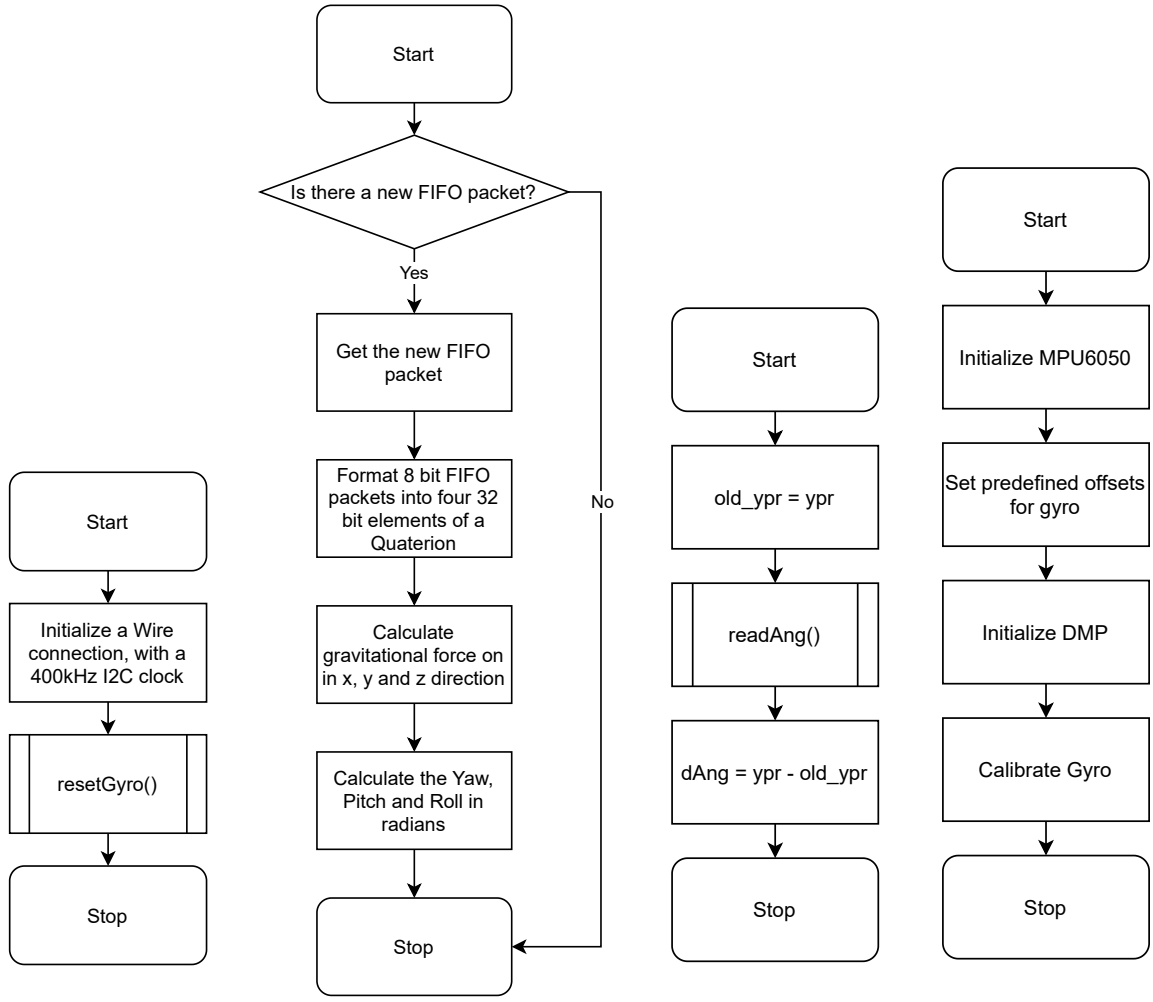
order to use the packet it has to be formatted first into Quarterion koordinates, and then the gravitational forces have to be factored in. At last the rotation around the module's z, x and y axis (yaw pitch and roll axis) is calcualted in radians, and saved into ypr.

H.1.4 readDAng()

This method is used for for updating the change i angle turn around its own axes, since last method call.
The first four processes from the flowchart on Figure H.2b are methods from the MPU class. The first method is the only communication with the MPU. The received data is then formatted into four 32 bit elements, which each represents a Quaternion coordinate [36]. These Quaternion coordinates are used to calculate the gravitational force in the x, y and z direction. At last this is recalculated into the turn around x, y and z axis are found in radians.

H.1.5 resetGyro()

This method resets the MPU, this means that it zeroes out all measured data. This is done by a method almost identical to the gyroSetup, with the difference that the I2C communication is already setup. This flowchart can be seen on Figure H.2d.



(a) Flowchart for gyroSetup().

(b) Flowchart for readAng().

(c) Flowchart for readDAng().

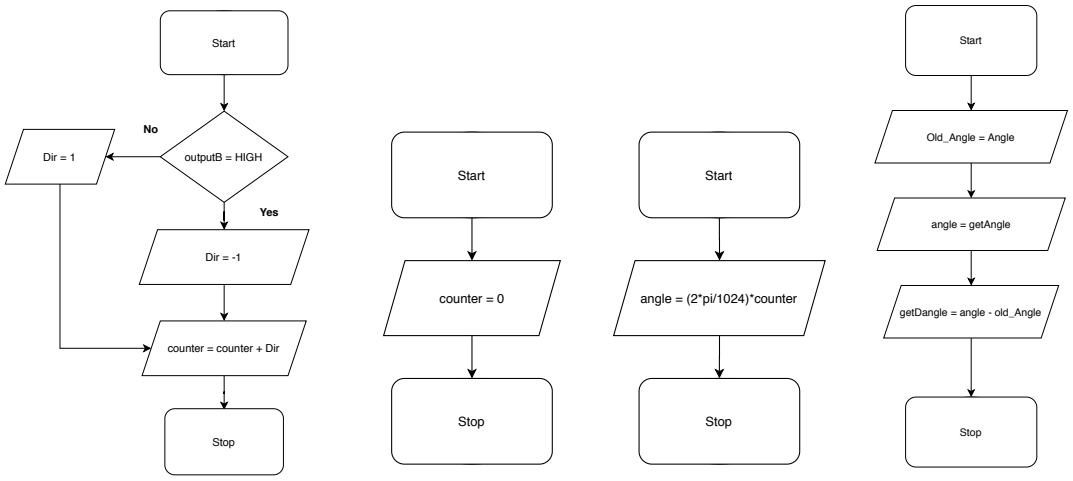
(d) Flowchart for resetGyro().

Figure H.2: Flowcharts concerning Gyro.

H.2 Cobot tachometer

To measure the turning angle between the object and the Cobot, a tachometer has been used in this project. The tachometer is incremental encoder with 1024 pulses in full circle. This incremental encoder has been used to measure angle by dividing 360° by 1024 and multiply by the number of pulses. In this section the program that has been used is reviewed through flowcharts. The program is using four functions in the programming pulseA(), pulseZ(), getObjectAngle() and getDiffObjectAng(). Functions such as pulseA() and pulseZ() are both attached to an interrupt pin and they only run when the interrupt pins goes HIGH. The purpose of pulseA() is to count the number of pulses from the tachometer when it turns and pulseZ() reset the counted number. For the two remaining functions getObjectAng() calculates the current angle and getDiffObjectAng() calculates the difference between current

angle and last angle. The entire program consist of the flowcharts shown on Figure H.3.



(a) Flowchart for pulseA().

(b) Flowchart for pulseZ().

(c) Flowchart for getAngle().

(d) Flowchart for getDAngle().

Figure H.3: Flowcharts concerning tachometer.

H.3 Motor Controller

To give the ability to drive the motors forward and get the velocity of the motors, two functions has been made called get_vq, and drivemotors(). In this appendix it will be explained what drivemotors() does, and how it works. get_vq is explained in Section 5.2.1.

Drivemotors

Drivemotors(angle,velocity,dir) is the function that controls the speed and direction of the two motors. It does this by receiving an angle direction change, a velocity and the direction the motors should turn, and from this calculates an individual speed for the left and right motor. The flowchart can be separated into four parts, the part that sets the direction of the motors (A), the part that calculates the velocity for each motor (B), the part that converts the velocity to a PWM signal (C), and the part that checks if the PWM signal is too large and sets the speed of the motors (D). The flowchart for this function can be seen on Figure H.4. Dir is the direction the motor wheels should turn, angle is the angle the motors should turn, velocity is total forward speed at which the motors should turn, Lwheel is the distance between the two wheels, wheel_radius is the radius of the wheels and motorveltopwm is a constant that determines how much to multiply the velocity with to get a PWM signal. The function returns a true or false statement depending on whether a direction has been chosen for the motors.

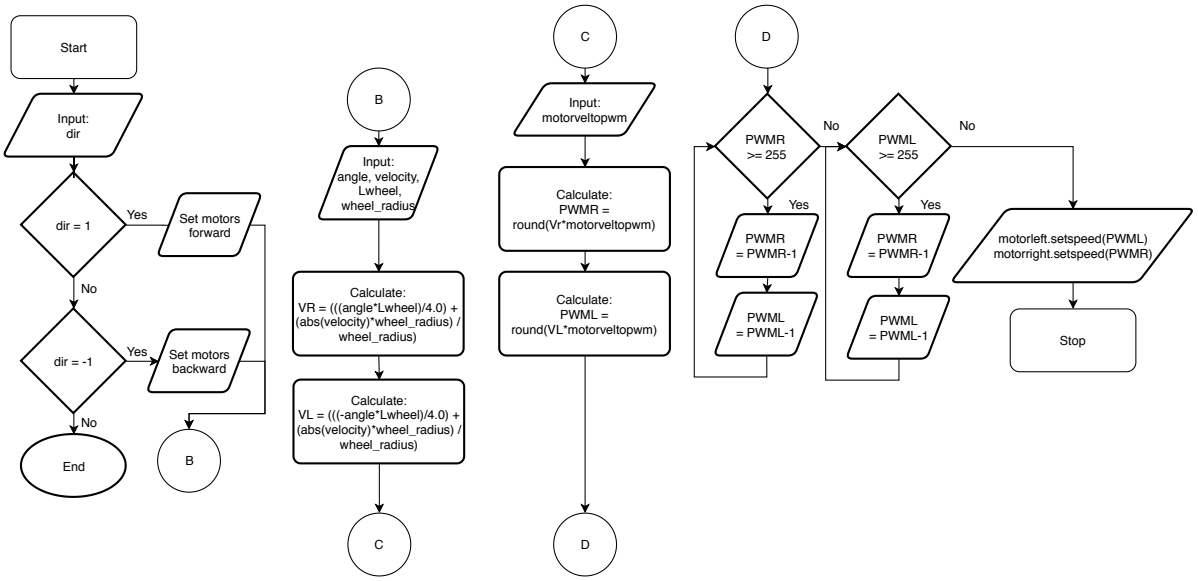


Figure H.4: Flowchart for drivemotors.

H.4 Main code flowcharts

H.4.1 First drive

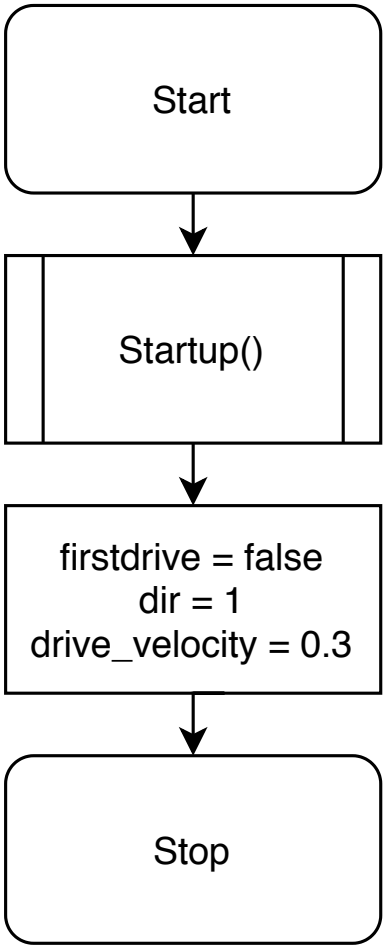


Figure H.5: Flowchart for first drive

H.4.2 Auto drive

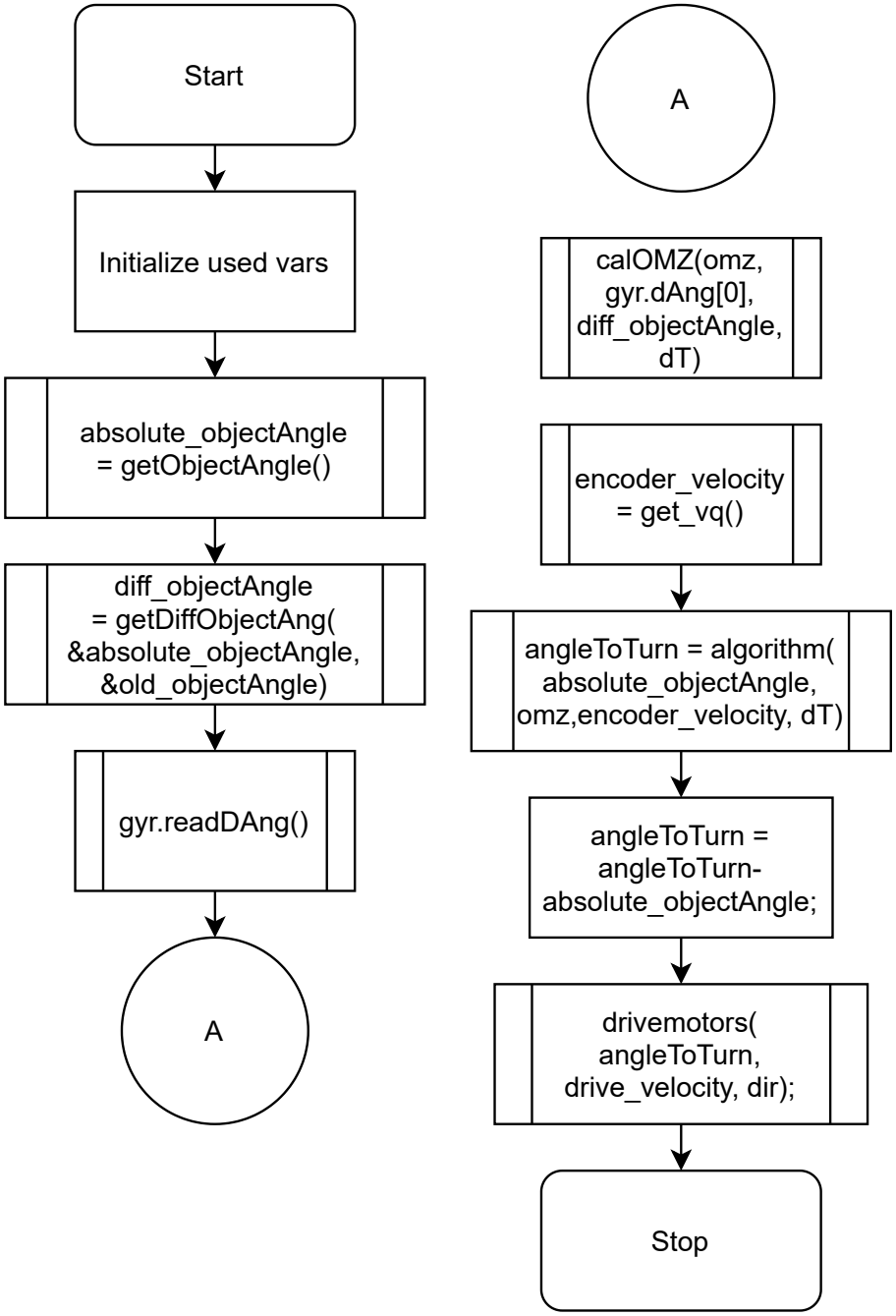


Figure H.6: Flowchart for Auto drive.

H.4.3 Manual drive

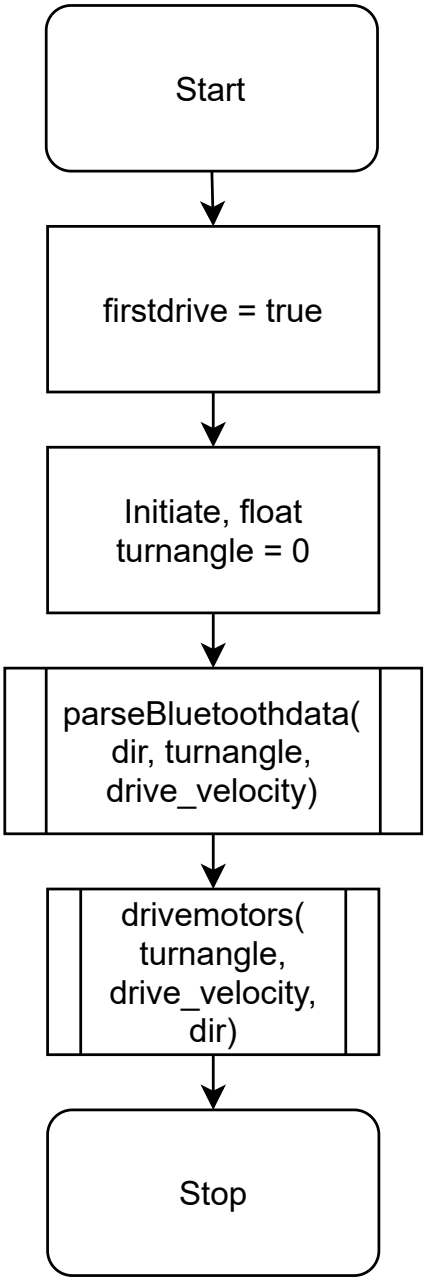


Figure H.7: Flowchart for manual drive.

Appendix I

Wiring diagram

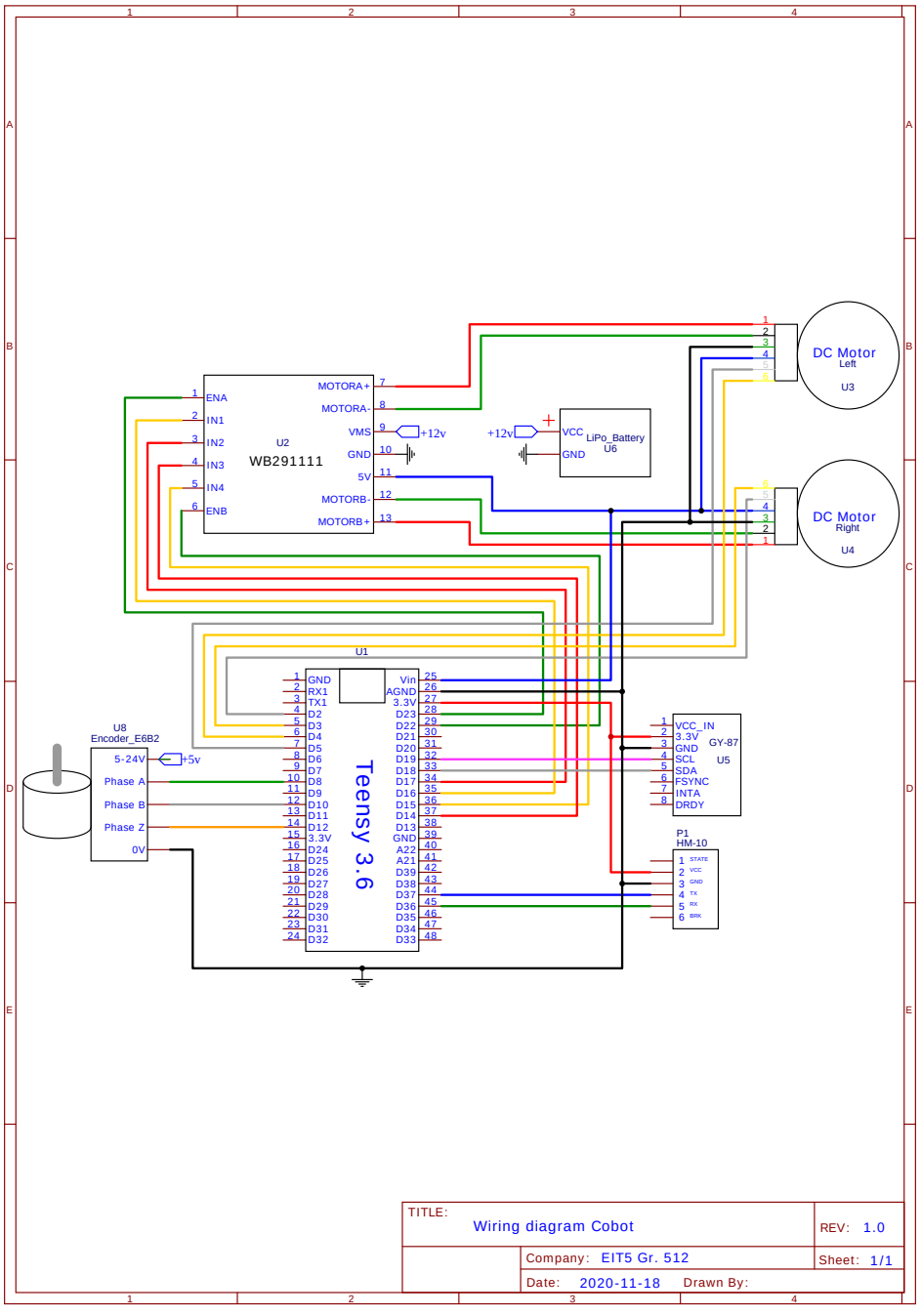


Figure I.1: Wiring diagram of the Cobot.

Appendix J

Measurement report - Motor variables

This measurement report documents the measurement of the variables mentioned below. All the measurements are done on the motor RK-370SD-22140, a picture of which can be seen on Figure J.1.

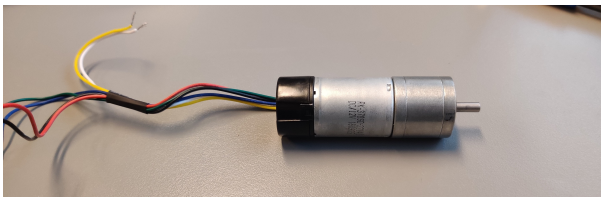


Figure J.1: Picture of the motor used for the measurements

The following measurements will be done on the motor:

- i_a - The current added to the motor
- $\ddot{\Theta}_m$ - The motors angular acceleration
- $\dot{\Theta}_m$ - The motors angular velocity
- B_m - The friction coefficient of the motor

Measurement instruments

For the measurements, the equipment shown in Table J.1 have been used.

Instrument	*Identification No.*	*Manufacturer, type, etc*
Multimeter	AUC-No. 77056	FLUKE 289
Multimeter	AAU-No. 77057	Fluke 289
Oscillioscope	AUC-No. 64572	DS06034A
Power Supply	AUC-No. 77077	EA-PS 7032-100
Tachometer	AUC-No. 08246	DT-205

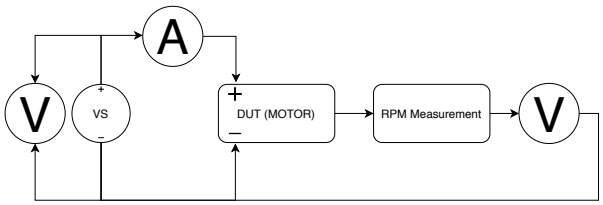
Table J.1: Measurement equipment used for the measurement of the motor.

Room and date

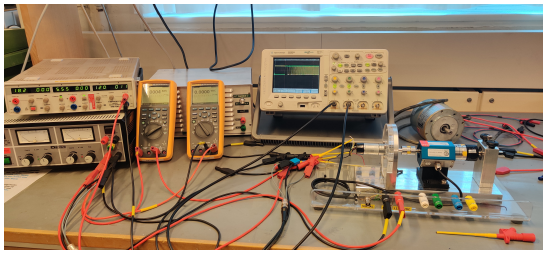
All measurement in this report are done on Fredrik Bajersvej 7, room C3-103, the 29 November 2020

Test setup

The test setup, as can be seen on Figure J.2a, show an illustration of how the motor and the test/measurement equipment are connected, and Figure J.2b shows a picture of the setup.



(a) Illustration of the test setup for measuring the motor.



(b) Picture of the motor setup.

Figure J.2: Two figures of the test setup.

J.1 Measurement of the motors angular velocity $\dot{\Theta}_m$

This measurement will run the motor at a fixed velocity, then measure how fast the motor is spinning to determine the velocity.

Measurement procedure

1. Connect the motor, power supply and meters as shown on Figure J.2.
2. Set the power supply to +12V and turn it on
3. Let it stabilise at a constant velocity
4. Measure the output from the tachometer and convert it to rad/sek.

Measurement result

In Table J.2 the RPM from the tachometer can be seen, as well as the RPM calculated to angular velocity.

Measurement	Value
RPM	130
$\dot{\Theta}_m$	13.61 Rad/s

Table J.2: Measurement results for measurements of $\dot{\Theta}_m$

J.2 Measurement of the motors angular acceleration $\ddot{\Theta}_m$

This measurement will start the motor from a velocity of 0, connect the motor to power, and measure how long it takes to get to max velocity. On Figure J.3 an illustration of the setup can be seen.

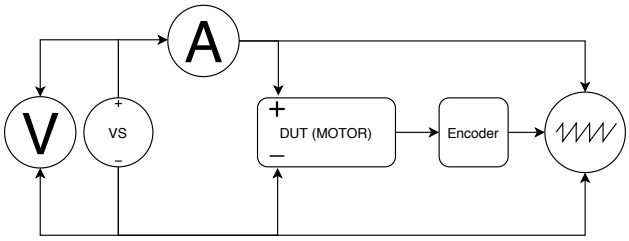


Figure J.3: Motor test setup with encoder

Measurement procedure

1. Connect the motor, power supply and oscilloscope as shown on Figure J.3.
2. Disconnect the positive power wire and set the power supply to +12V and turn it on
3. Connect the yellow wire from the tachometer on the motor to the oscilloscope ch1, and the +12V from the power supply to ch2
4. Connect the positive wire to the power supply
5. Let the motor spin up to full speed, and measure the time between pulses.
6. Disconnect the power wire to the motor, and reconnect it. Measure the time between CH2 going high, and the first time the time between the pulses are the same as with full speed.

Measurement result

In Table J.3 the measured times can be seen, where also the time until 90% of full speed is reached.

Measurement	Value
Time between pulses at full speed	268 µs
Time until first full speed pulse	43.53 ms
Time until first 90% speed pulse	32 ms

Table J.3: Measurement results for measurements of $\ddot{\Theta}_{\mathrm{m}}$

As we already know the angular velocity when the motor is going at full speed, as was measured in J.1 we can use that to calculate an acceleration.

$$\ddot{\Theta}_{\mathrm{m}} = \frac{\omega \mathrm{f} - \omega \mathrm{i}}{\Delta \mathrm{t}} = \frac{13.61 - 0}{0.04353} = 312.65[\frac{\mathrm{rad}}{\mathrm{s}^2}] \tag{J.1}$$

J.3 Measurement of the motors friction coefficient B_{m}

This measurement will measure the current and velocity in a number of steady states. From this the motor friction will be calculated.

Measurement procedure

1. Connect the motor, power supply and meters as shown on Figure J.2.
2. Set the power supply to +5V and turn it on
3. Note the velocity of the motor and the current running into it.
4. Set the power supply to +8V
5. Note the velocity of the motor and the current running into it.
6. Set the power supply to +12V
7. Note the velocity of the motor and the current running into it.
8. From the three steady states calculate the motors friction

Measurement result

In Table J.4 the measurement results can be seen.

Measurement of B_m	Meas. 1	Meas. 2	Meas. 3
Voltage	5.04 V	8.02 V	12.0 V
Current	0.145 A	0.172 A	0.220A
RPM	53	86	130
$\dot{\Theta}_m$	5.55 rad/s	9.00 rad/s	13.61 rad/s

Table J.4: Measurement results for measurements of B_m

With a known K_t and I_a the friction coefficient may be found at different angular velocities using the equation seen in Equation J.2. K_t has been found in Section 5.3.2, and is set to 0.297.

$$B_m = \frac{K_t \cdot I_a}{\dot{\Theta}_m} \tag{J.2}$$

Where:

K_t	The motor constant	[-]
τ	The torque	[$N \cdot m$]
B_m	The motor friction coefficient	[-]

This gives three different friction coefficients and an average friction coefficient that can be seen in Table J.5.

Measurement	Result
Bm_1	0.0077
Bm_2	0.0056
Bm_3	0.0048
$\mathrm{Bm}_{\mathrm{avg}}$	0.0060

Table J.5: Calculation of $\mathrm{B_m}$.

Appendix K

Measurement report - Code time

This measurement report documents the measurements of how how long the functions mentioned below takes to run. All measurements are done on the Teensy 3.6, with all interrupt pins set in a steady state as to not activate.
The following functions will be measured:

- getDiffObjectAng() - The angle from the Cobot encoder
- gyr.readDAng() - Updates the angle from the gyroscope
- calOMZ - Calculates the value of omz
- get_vq() - The velocity the motors are driving with
- drivemotors() - Function that drives the motors

Measurement instruments

For the measurements, the equipment shown in Table K.1 have been used.

Instrument	*Identification No.*	*Manufacturer, type, etc*
PC	-	Intel(R) Core(TM) i7-2670QM
Teensy	-	Teensy 3.6
USB cable	-	2 M

Table K.1: Measurement equipment used for the measurement of code time.

Room and date

All measurements in this report are performed on Fredrik Bajersvej 7, room B2-203, the 1. December 2020.

K.1 Time measurement of the mentioned functions

This measurement will start a timer at the start of the first function called, save a copy of how long it has run after each function has finished running, until all the mentioned functions has run once. Consequently, the total time will be solved by adding all the values of time together.

Test setup

The test setup, as can be seen on Figure K.1, shows an illustration of how the Teensy and the test equipment are connected.

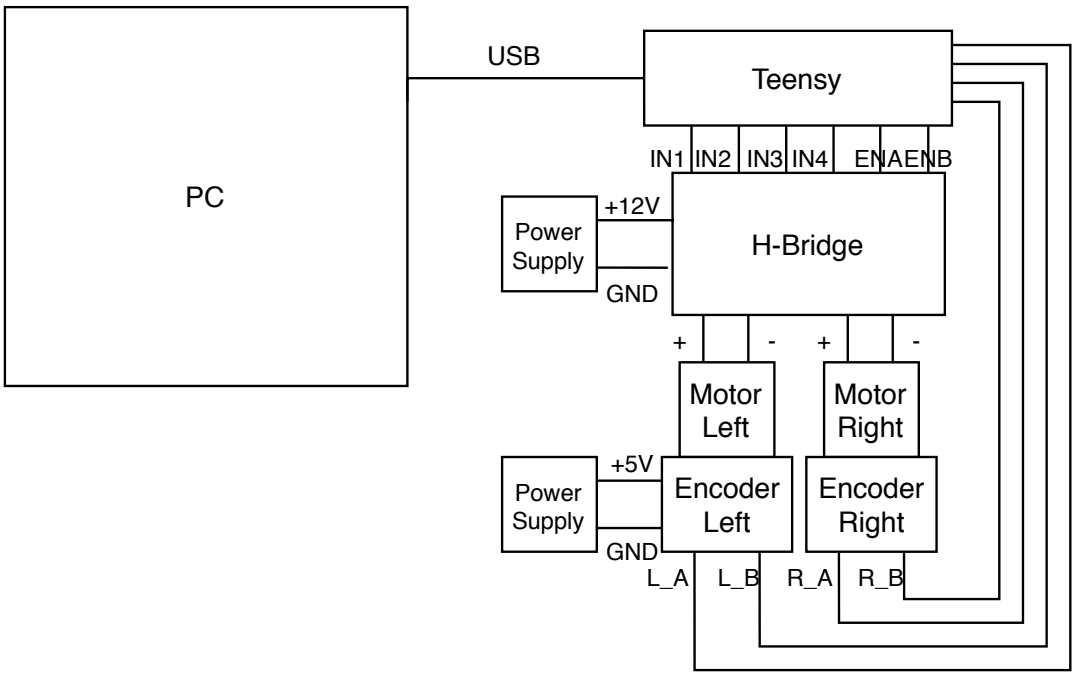


Figure K.1: Illustration of the test setup for measurement of the code.

Code

The code for this test can be seen on the Github repository under [20]:
master/Measurement_reports/_code_time/

Measurement procedure

1. Connect the Teensy as shown on Figure K.1.
2. Upload and run the _code_time.ino code
3. Note all the individual times, and the total time for all the functions.

Measurement results

In Table K.2, the time each function takes to run can be seen. The function that takes the longest to run is the Gyroscope, which takes a total of 135 µs, and the total time it takes to run all the functions is 160 µs.

Measurement	Result(µs)
Encoder	1
Gyroscope	135
Omz	6
Velocity	3
Drive	15
Total	160

Table K.2: Calculation of measurement time.

Uncertainties

As the way the timing for each function is measured is by saving the local time to a variable, the time it takes to save that variable will also be added to the total time for that function. But as this is seen as negligible, it is not calculated into the total time.

Appendix L

Measurement report - Interrupt timing

This measurement report documents the measurements of how long the interrupts functions mentioned below takes to run. All measurements are done on the Teensy 3.6, with all other interrupts disabled.
The following interrupt functions will be measured:

- Motor encoder pulse - Interrupt that counts the amount of encoder pulses
- Cobot tachometer pulse - Interrupt that counts the amount of encoder pulses
- Bluetooth data received - Interrupt that receives the data from the Bluetooth module

Measurement instruments

For the measurements, the equipment shown in Table L.1 have been used.

Instrument	*Identification No.*	*Manufacturer, type, etc*
PC	-	DELL XPS 15 9570
Teensy	-	Teensy 3.6
USB cable x 2	-	2 M
Arduino	-	Arduino Uno
Drill	-	Dewalt Drill 12V
Motor x2	-	rk-370 sd-22140
H-bridge	-	WB291111

Table L.1: Measurement equipment used for the measurement for timing of interrupts.

Room and date

All measurement in this report are performed on Fredrik Bajersvej 7, room B2-203, the 8. December 2020

L.1 Measurement of motor encoder pulse

As the way this interrupt routine is implemented, any change to that code would also change the way it is compiled, which would make the function take longer to run. Instead of measuring the interrupt directly, the algorithm would be run first without any interrupts enabled, and afterwards again with the motor encoder interrupt enabled, with the motors spinning at maximum velocity. The time difference between these two runs would be seen as the time the interrupt takes.

Test setup

The test setup, as can be seen on Figure L.1, shows an illustration of how the Teensy and the test equipment are connected.

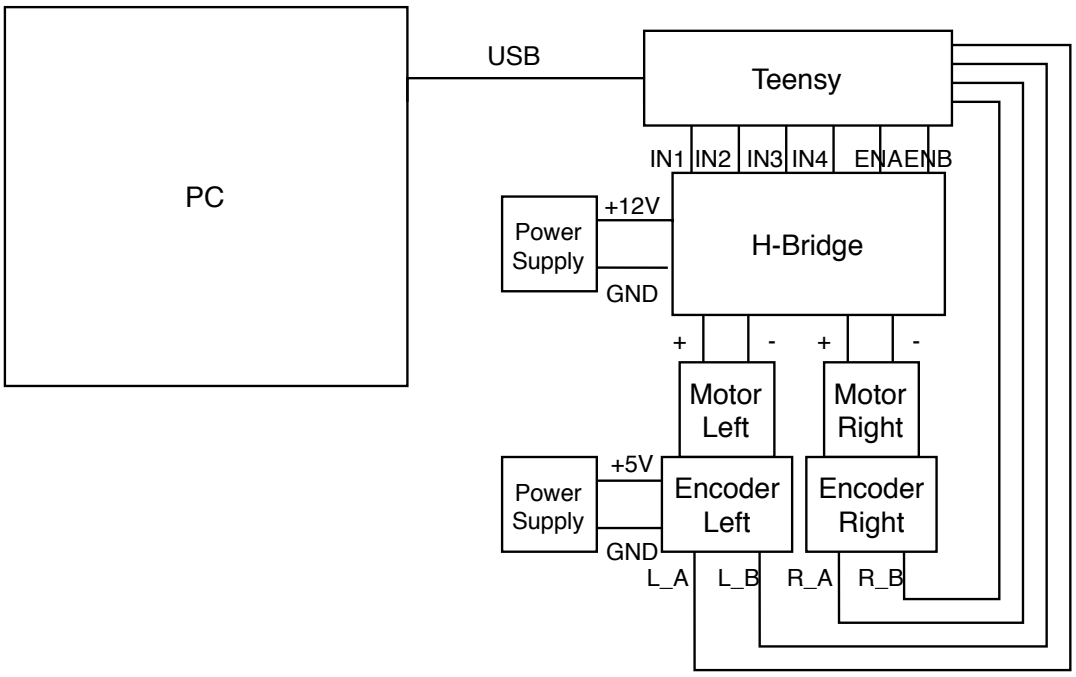


Figure L.1: Illustration of the test setup for the motor encoder interrupt.

Code

The code for this test can be seen on the Github repository under [20]:
master/Measurement_reports/interrupt_timing/encoder_pulse

Measurement procedure

1. Connect the Teensy as shown on Figure L.1.
2. Upload and run the encoder_pulse.ino code as, and note the time it takes to run
3. Comment out line 103 and 104, and run the code again. Note the time it takes to run.

Measurement result

	Result
With interrupt	2811 µ
Without interrupt	2778 µ
Difference	33 µ

Table L.2: Result of measurement of motor encoder timing

L.2 Measurement of tachometer for object angle

Instead of measuring the interrupt directly, the algorithm would be initially run without any interrupts enabled, and afterwards again with the Cobot tachometer interrupt enabled. As it

is hard to define how fast the object encoder rotates when it is in use, it will be set to rotate at a constant velocity that is deemed larger than what it would rotate within a real scenario. The time difference between these two runs would be seen as the time the interrupt takes.

Test setup

The test setup, as can be seen on Figure L.2, shows an illustration of how the Teensy and the test equipment are connected.

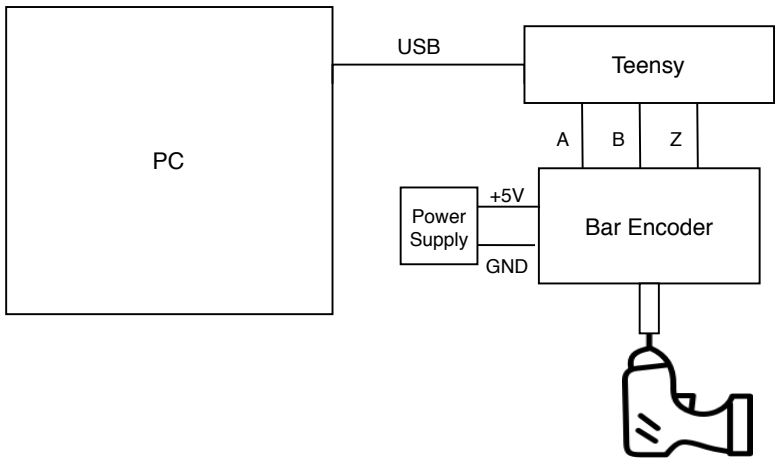


Figure L.2: Illustration of the test setup for the object encoder interrupt.

Code

The code for this test can be seen on the Github repository under [20]:
master/Measurement_reports/interrupt_timing/obj_ang/

Measurement procedure

1. Connect the Arduino as shown on Figure L.2.
2. Comment out line 103 and 104 from file obj_ang.ino, and run the code again. Note the time it takes to run. Note the time it takes to run.
3. Connect the drill to the tachometer, and set it to slow speed, and turn it on.
4. Upload and run the code with line 103 and 104 uncommented, and note the time it takes to run

Measurement results

	Result
With interrupt	1142 µ
Without interrupt	1140 µ
Difference	2 µ

Table L.3: Result of measurement of object encoder timing

L.3 Measurement of Bluetooth

Instead of measuring the interrupt directly, the algorithm would be run first without any interrupts enabled, and afterwards again with the Bluetooth interrupt enabled. To simulate a worst case scenario of the Bluetooth module, another Arduino will be used to continuously send Serial data to the serial port the Bluetooth module is connected to. The time difference between these two runs would be seen as the time the interrupt takes.

Test setup

The test setup, as can be seen on Figure L.3, shows an illustration how the Teensy and the test equipment are connected.

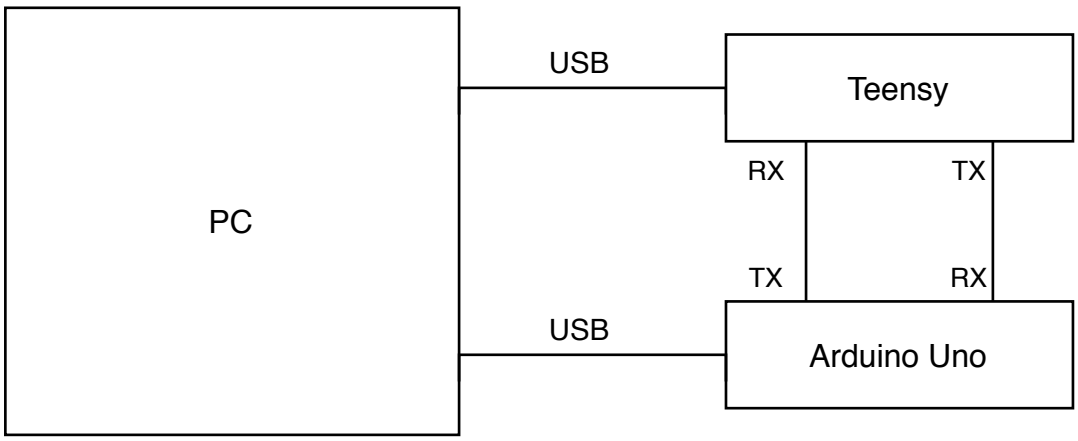


Figure L.3: Illustration of the test setup for Bluetooth interrupt.

Code - Teensy

The main code for this test can be seen on the Github repository under [20]:
master/Measurement_reports/interrupt_timing/bluetooth/teenzy

Code - Arduino

The main code for this test can be seen on the Github repository under [20]:
master/Measurement_reports/interrupt_timing/bluetooth/arduino/arduino.ino

Measurement procedure

1. Connect the Teensy and Arduino Uno as shown on Figure L.3.
2. Comment out line 78 and 100 from teensy.ino, and run the code. Note the time it takes to run.
3. Upload arduino.ino to the Arduino Uno.
4. Upload and run the code to the Teensy with line 78 and 100 uncommented, and note the time it takes to run

Measurement results

	Result
With interrupt	1138 µ
Without interrupt	1138 µ
Difference	0 µ

Table L.4: Result of measurement of Bluetooth timing.

Appendix M

Measurement report - Size of the ring buffer

This measurement report documents the measurement of the variables mentioned below. All the measurement was measured by using Arduino Teensy and Arduino program.
The following measurements will be done.

- t - The time which is taken for implemeting the algorithm function
- rbs - The the ring buffer size

Measurement instruments

For the measurements, the equipment shown in Table M.1 have been used.

Instrument	*Identification No.*	*Manufacturer, type, etc*
PC	-	DELL XPS 15 9570
Teensy	-	Teensy 3.6
USB cable x 2	-	2 M
Motor x2	-	rk-370 sd-22140
H-bridge	-	WB291111

Table M.1: Measurement equipment used for the measurement for timing of interrupts.

Room and date

All measurement in this report are done on Frederik Bajersvej 7, room B2-203, the 28 November 2020

Test setup

The test setup, as can be seen on Figure M.1, shows an illustration of how the Teensy and the test equipment are connected.

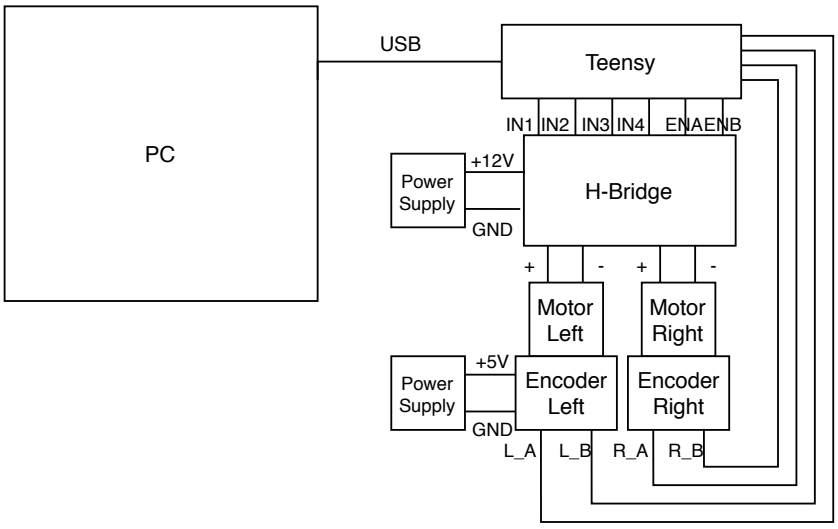


Figure M.1: Illustration of the test setup for the motor encoder interrupt.

The buffer size is sat to 1429 as calculated in Section 5.1.6. For checking the time, this code has been used.

Code

The main code for this test can be seen on the Github repository under [20]:
master/Measurement_reports/ring_buff_size

Measurement procedure

1. Connect the Teensy as shown on Figure M.1.
2. Upload and run the code ring_buff_size.ino, and note the time it takes to run.

Measurement result

	Result
Run time	18 ms

Table M.2: Result of measurement of motor encoder timing

Appendix N

Measurement report - Drift in Gyro

This measurement report documents the how much the gyro measurements from the GY-87 drifts over time. It assumes that the offset values for the GY-87 has previously been set. This is done by logging the yaw, pitch and roll measurements from the GY-87 each 10 seconds.

Measurement instruments

For the measurements, the equipment shown in Table N.1 have been used.

Instrument	*Identification No.*	*Manufacturer, type, etc*
PC	-	Dell XPS 15
Micro-controller	-	Teensy 3.6
DUT	-	GY-87
USB cable	-	2 m
Connection Wires	-	10 cm long
Room thermometer	-	Rosenborg Model 66760

Table N.1: Measurement equipment used for the measurement drift from gyro.

Room and date

All measurements in this report are done on Niels Borhs Vej 40, 1. 14, the 11. December 2020

N.1 Measurement of gyro drift.

The code used for the measurements can be seen on the project repository, under 'Measurement_reports/gyro_drift_test' [20].
This will measure the values from the gyro and each second the latest values measured will be written to the serial port. This will be done for 30 minutes. By using the matlab file 'Recieve_data.m' and configuring the com port, matlab will log all the data, and save it in a measurement object.

Test setup

The test setup, as can be seen on Figure N.1, shows an illustration how the micro-controller and the test equipment are connected.

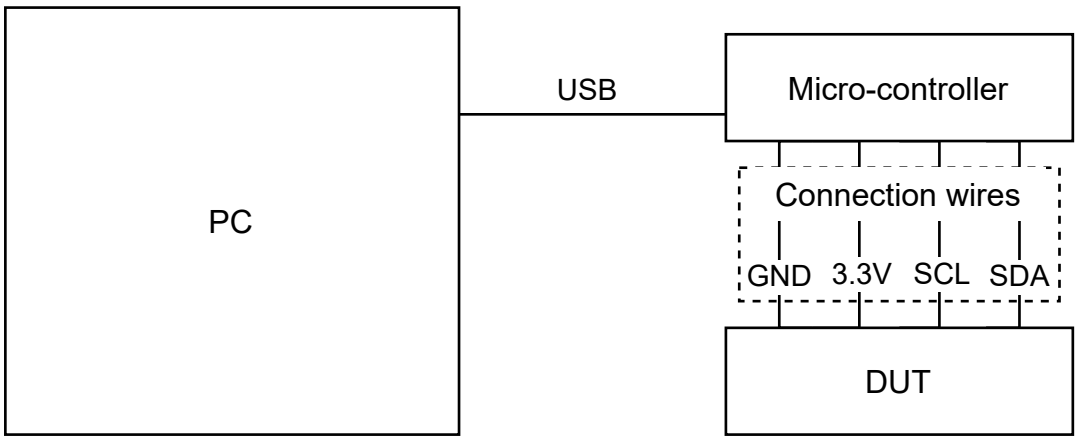


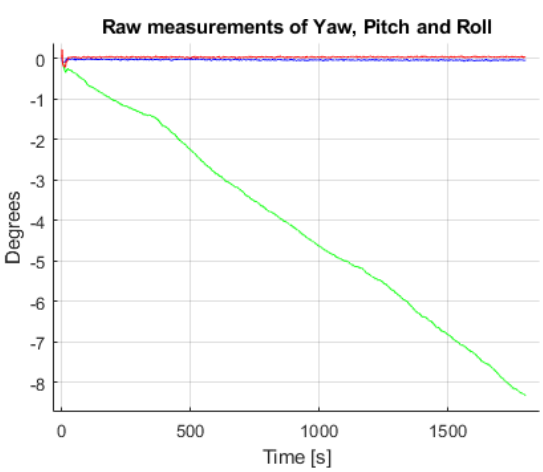
Figure N.1: Illustration of the test setup for measuring of the gyro drift.

Measurement procedure

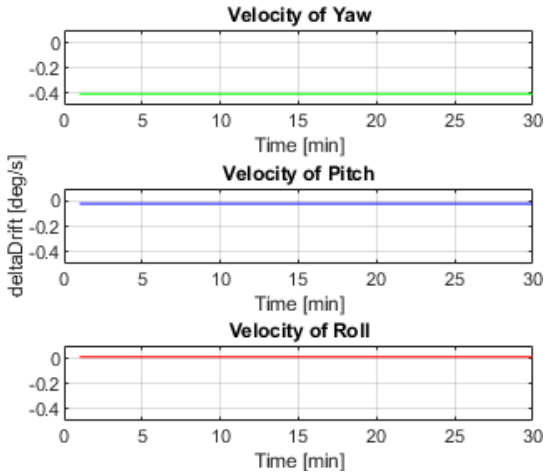
1. Measure the room temperature.
2. Connect micro-controller to PC.
3. Upload gyro_drift_test.ino to micro-controller.
4. Unconnect micro-controller from PC.
5. Connect micro-controller to DUT.
6. Connect micro-controller to PC.
7. Run Recieve_data.m, and wait until it is done.
8. Run analyse_data.m.
9. Save the three output figures.

Measurement result

The room temperature was measured to 22.3°C
On the Figures N.2 the results from the test can be seen.



(a) Drift - Yaw, Pitch and Roll [deg]



(b) Velocity of drift - Yaw, Pitch and Roll [deg/min]

Figure N.2: Results from gyro drift test.

Appendix O

Measurement report - Velocity to PWM

This measurement report documents what velocity the motors are at, at different duty-cycles for the PWM signal. All measurements are done on the Teensy 3.6, and the H-bridge WB291111, where the Cobot does not touch the ground, and the wheels can spin freely.

Measurement instruments

For the measurements, the equipment shown in Table O.1 have been used.

Instrument	*Identification No.*	*Manufacturer, type, etc*
PC	-	DELL XPS 15 9570
Teensy	-	Teensy 3.6
USB cable	-	2 M
H-bridge	-	WB291111

Table O.1: Measurement equipment used for the measurement of velocity to PWM

Room and date

All measurements in this report are performed at Toldstrupsgade 18. 6.3 , the 17. December 2020.

O.1 Measurement of PWM to velocity - Bottom-up

This measurement will send a PWM signal to each motor, going from 70 to 255 in an interval of 5. It starts at 70 as the motors do not turn until 70+. For each interval it will measure the velocity of each motor and output it to the serial port. This should then be logged to the result table.

Test setup

The test setup, that can be seen on Figure O.1, shows an illustration of how the Teensy and the test equipment are connected.

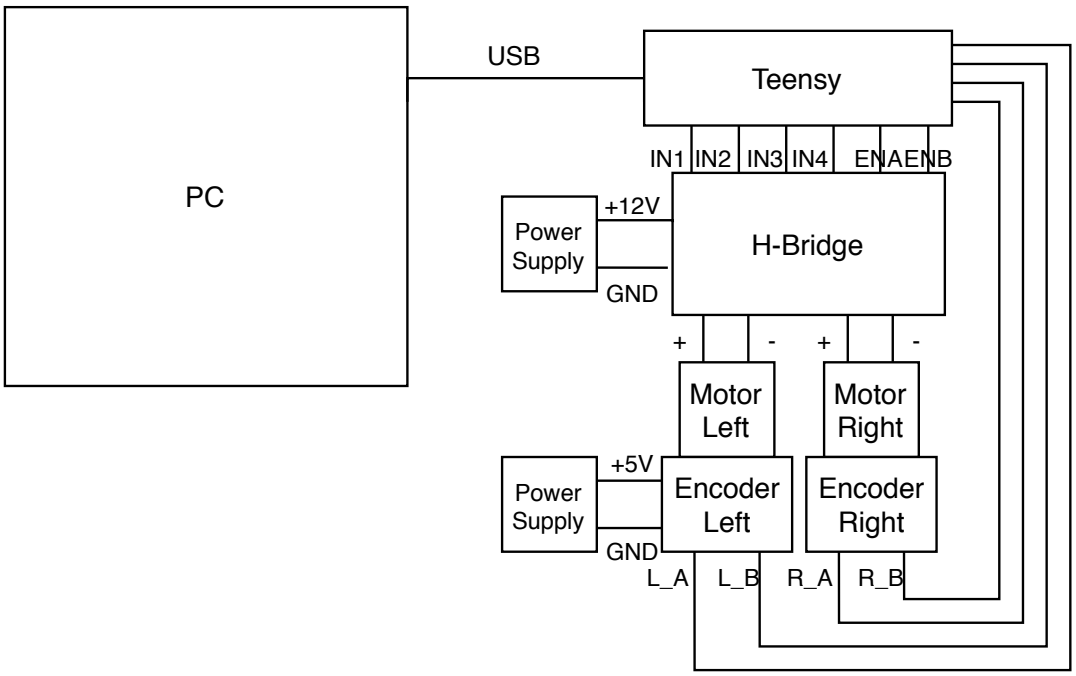


Figure O.1: Illustration of the test setup for test of PWM

Code

The main code for this test can be seen on the Github repository under [20]: master/Measurement_reports/vel_to_pwm/bottom_up

Measurement procedure

1. Connect the Cobot as shown on Figure O.1.
2. Place the Cobot so the wheels does not touch the ground.
3. Upload and run the code in bottom_up.ino.
4. Save all the data outputted to the serial port.

Measurement results

In Table O.2 the measurement result can be seen.

PWM	Velocity	
	L	R
70	0.0	0.0
75	0.0	0.0
80	0.0	0.0
85	0.0	0.383
90	0.438	0.483
95	0.513	0.498
100	0.526	0.510
105	0.536	0.521
110	0.546	0.530
115	0.556	0.540
120	0.564	0.548
125	0.571	0.554
130	0.578	0.562
135	0.584	0.567
140	0.590	0.572
145	0.594	0.577
150	0.599	0.582
155	0.604	0.586
160	0.609	0.590

PWM	Velocity	
	L	R
165	0.613	0.594
170	0.617	0.598
175	0.621	0.601
180	0.624	0.604
185	0.627	0.607
190	0.631	0.609
195	0.633	0.612
200	0.636	0.614
205	0.638	0.616
210	0.639	0.618
215	0.641	0.621
220	0.643	0.623
225	0.644	0.624
230	0.647	0.627
235	0.649	0.629
240	0.650	0.630
245	0.652	0.632
250	0.653	0.633
255	0.663	0.643

Table O.2: Measurement results from measuring PWM to velocity - bottom-up .

O.2 Measurement of PWM to velocity - Top-bottom

This measurement will send a PWM signal to each motor, going from 255 to 0 in an interval of 5. For each interval it will measure the velocity of each motor and output it to the serial port. This should then be logged in the result table.

Test setup

The test setup, that can be seen on Figure O.2, shows an illustration of how the Teensy and the test equipment are connected.

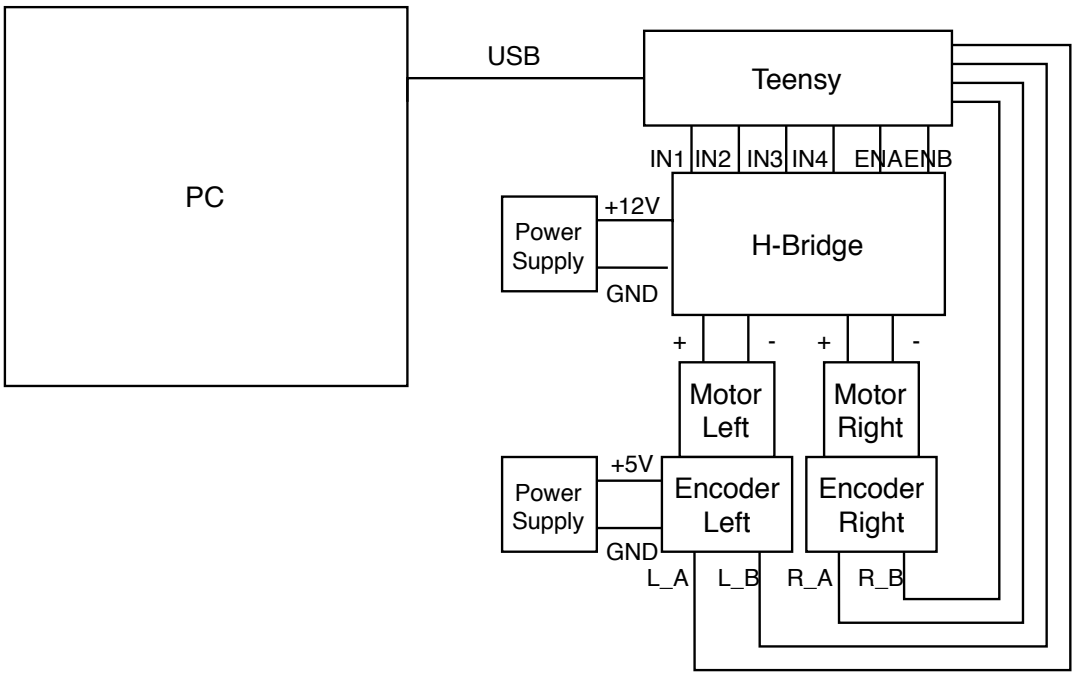


Figure O.2: Illustration of the test setup for test of PWM

Code

The main code for this test can be seen on the Github repository under [20]: master/Measurement_reports/vel_to_pwm/top_bottom

Measurement procedure

1. Connect the Cobot as shown on Figure O.2.
2. Place the Cobot so the wheels does not touch the ground.
3. Upload and run the code in top_bottom.ino.
4. Save all the data outputted to the serial port.

Measurement results

In Table O.3 the measurement result can be seen.

PWM	Velocity	
	L	R
255	0.664100	0.643300
250	0.653500	0.632900
245	0.652900	0.632800
240	0.650200	0.630100
235	0.649200	0.629100
230	0.647200	0.627100
225	0.645800	0.625600
220	0.644200	0.623500
215	0.641600	0.621000
210	0.639900	0.619500
205	0.637700	0.617500
200	0.635700	0.615300
195	0.633500	0.613500
190	0.631400	0.610400
185	0.628500	0.608300
180	0.625800	0.605300
175	0.623100	0.602600
170	0.619900	0.599600
165	0.615900	0.596300
160	0.611500	0.592900
155	0.607300	0.588900
150	0.602800	0.584900
145	0.597600	0.580000
140	0.593400	0.575900
135	0.588400	0.570400
130	0.583200	0.565000

PWM	Velocity	
	L	R
125	0.576800	0.558800
120	0.570300	0.552400
115	0.562900	0.544600
110	0.554500	0.538300
105	0.546600	0.529100
100	0.537500	0.518500
95	0.525100	0.508000
90	0.511300	0.495000
85	0.496500	0.481400
80	0.479800	0.464700
75	0.461300	0.446300
70	0.441600	0.427000
65	0.419100	0.404300
60	0.391400	0.377700
55	0.360900	0.345500
50	0.326400	0.310900
45	0.283000	0.269700
40	0.234000	0.223200
35	0.179700	0.170600
30	0.119100	0.106600
25	0.043800	0.031800
20	0.000000	0.000000
15	0.000000	0.000000
10	0.000000	0.000000
5	0.000000	0.000000
0	0.000000	0.000000

Table O.3: Measurement results from measuring PWM to velocity - top-bottom.

Bibliography

[1] W. E. in Denmark, *Working environment act*, https://at.dk/regler/love-eu-forordninger/arbejdsmiljoe-674/, (Sidst besøgt 16-09-2020), 2020.

[2] Arbejdstilsynet, *Anmeldte arbejdsulykker i tal*, https://at.dk/arbejdsmiljoe-i-tal/analyser-og-publikationer/anmeldte-arbejdsulykker-i-tal/, (Sidst besøgt 09-09-2020).

[3] ——, *Tilsyn i tal*, https://at.dk/arbejdsmiljoe-i-tal/tilsyn-i-tal/, (Sidst besøgt 18-11-2020).

[4] D. W. E. Regulations, *About danish working environment regulations*, https://at.dk/om-os/formaal-strategi-organisation/, (Sidst besøgt 09-09-2020).

[5] Arbejdstilsynet, *Løft, træk og skub*, https://at.dk/regler/at-vejledninger/loeft-traek-skub-d-3-1/, (Sidst besøgt 12-09-2020).

[6] D. ergonomiske konsulen, *Manuel håndtering*, http://denergonomiskekonsulent.dk/Dokumenter/Manuel%20haandtering.pdf, (Sidst besøgt 14-09-2020).

[7] B. for Bygge & Anlæg, *Arbejdsmiljø i bygge og anlæg*, https://www.haandbogen.info/wp-content/uploads/H%C3%A5ndbogen-2020-print.pdf, (Sidst besøgt 14-09-2020).

[8] J. o. M. F. 3F Bygge, *Risiko for akut overbelastning*, https://www2.3f.dk/bjmf/fagforening/klubber-i-bjmf/snedker-toemrernes-brancheklub/st-klubbens-sikkerhedskampagne-saet-foden-ned/2,-d-,-risiko-for-akut-overbelastning, (Sidst besøgt 14-09-2020).

[9] B. Branchefællesskabet for Arbejdsmiljø (BFA) Fællessekretariatet, *Hvordan du bedst ...transporterer lange og tunge varer*, http://hvordandubedst.dk/filer/hvor-arbejder-du/instruktionsark/B4.pdf, (Sidst besøgt 14-09-2020).

[10] S. Equipment, *Warehouse distribution trolley*, http://www.shopequip.co.uk/shopping+baskets+shopping+trolleys/warehouse+distribution+trolley-C55-I2454.html, (Last visited 17-11-2020).

[11] L. det ver nemt, https://www.lomax.dk/lager/vogne-og-transportudstyr/rullevogne/lager-og-transportvogne/lagervogn-120x70-cm-500-kg-massive-hjul-9901030/?pla=1&gclid=EAIaIQobChMI3LnX8e7b6wIVVeDtCh3pkAr9EAQYASABEgJ93fD_BwE, (Last visited 09-09-2020), 2020.

[12] Indiamart, *Toyota forklift*, https://www.indiamart.com/proddetail/toyota-forklift-16621891091.html, (Sidst besøgt 18-11-2020).

[13] Wikipedia, *Forklift*, https://en.wikipedia.org/wiki/Forklift, (Sidst besøgt 15-09-2020).

[14] Welch, *Raymond 9400 side loader long load forklift*, https://welchequipment.com/for-sale/side-loader-long-load-forklift-2/, (Sidst besøgt 18-11-2020).

[15] Baumann, *What is a side loader forklift used for?*, https://baumann-sideloaders.com/what-is-a-side-loader-forklift/, (Sidst besøgt 15-09-2020).

[16] Toyota, *Automatic guided vehicles (agvs)*, https://www.toyotamaterialhandling.com.au/products/automatic-guided-vehicles-agvs/toyota-tae050-hd-autopilot-cart/, (Sidst besøgt 16-09-2020).

[17] ——, https://toyota-forklifts.eu/automation/automated-solutions/#truck, (Last visited 09-09-2020), 2020.

[18] D. Statistik, *Befolkningens løn*, http://www.dst.dk/pukora/epub/upload/19581/befloen.pdf, (Sidst besøgt 17-09-2020), 2013.

[19] E. Cronkleton, *What is the average walking speed of an adult?*, https://www.healthline.com/health/exercise-fitness/average-walking-speed, (Sidst besøgt 16-10-2020), 14-03-2019.

[20] G512, *Eit5 cobot github repository*, https://github.com/NielsDyrberg/EIT5-COBOT.

[21] Elektronik-lavpris, *Motor: Dc; with encoder,with gearbox; hp; 12vdc; 5.6a; 130rpm*, https://elektronik-lavpris.dk/p145876/pololu4846-motor-dc-with-encoderwith-gearbox-hp-12vdc-56a-130rpm/, (Last visited 16-11-2020), 2020.

[22] Let-Elektronik, *Rotary encoder - 1024 p/r (quadrature)*, https://let-elektronik.dk/shop/1720-omskiftere--encoder/11102-rotary-encoder---1024-pr-quadrature/, (Sidst besøgt 01-12-2020).

[23] Conrad, *Gy-87*, https://www.conrad.de/de/p/gy-87-10-dof-3-achsen-gyroskop-accelerometer-mpu6050-mmc5883l-bmp180-sensor-modu-802235245.html, (Sidst besøgt 25-11-2020).

[24] i2cdevlib, *I2cdevlib*, https://github.com/NielsDyrberg/i2cdevlib, (Sidst besøgt 11-12-2020).

[25] J. Mckie, *Electric motor shafts*, https://www.ierservices.com/blog/what-is-the-best-material-for-an-electric-motor-shaft-0, (Last visited 04-11-2020), 2018.

[26] E. edge, *Coefficient of friction equation and table chart*, https://www.engineersedge.com/coeffients_of_friction.htm, (Last visited 04-11-2020), 2020.

[27] NXP, *Kinetis k66 sub-family*, https://www.nxp.com/docs/en/data-sheet/K66P144M180SF5V2.pdf, (Sidst besøgt 10-12-2020).

[28] STMicroelectronics, *L298 - dual full-bridge driver*, http://www.geeetech.com/Documents/L298N%20datasheet.pdf, (Sidst besøgt 14-12-2020).

[29] Amazon, *Oiyagai l298 module l298n dual bridge dc stepper motor driver shield expansion controller board for arduino intelligent car automaton*, https://www.amazon.ca/Willwin-Expansion-Controller-Intelligent-Automaton/dp/B076QDDY4G, (Sidst besøgt 10-12-2020).

[30] homofaciens, *H bridge*, https://www.homofaciens.de/technics-base-circuits-h-bridge_en.htm, (Sidst besøgt 16-12-2020).

[31] T. S. Pedersen, *Pulse width modulation - lecture 13 slides*, (Sidst besøgt 14-12-2020).

[32] mbtechworks.com, *Raspberry pi pulse width modulation (pwm)*, https://www.mbtechworks.com/projects/raspberry-pi-pwm.html, (Sidst besøgt 10-12-2020).

[33] M. Egerstedt, *Control of mobile robots- 2.2 differential drive robots*, https://www.youtube.com/watch?v=aE7RQNhwnPQ, (Sidst besøgt 14-12-2020).

[34] K. Morich, *Serial bluetooth terminal*, https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal, (Sidst besøgt 16-12-2020).

[35] *Hm-10 bluetooth module*, https://components101.com/wireless/hm-10-bluetooth-module, (Sidst besøgt 16-12-2020).

[36] mjwhite8119, *The mpu6050 explained*, https://mjwhite8119.github.io/Robots/mpu6050, (Last visited 16-11-2020), 2019.

[37] RobotPlatform, *Wheel control theory)*, http://www.robotplatform.com/knowledge/Classification_of_Robots/wheel_control_theory.html, (Sidst besøgt 24-09-2020).

[38] Wikipedia, *Ism band*, https://en.wikipedia.org/wiki/ISM_band, (Last visited 29-09-2020), 2020.

[39] E. RF, *Ism frequency bands*, https://www.everythingrf.com/community/ism-frequency-bands, (Last visited 29-09-2020), 2020.

[40] J. G. Sponås, *Things you should know about bluetooth range*, https://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range, (Sidst besøgt 21-09-2020), 2020.

[41] Wikipedia, *Ieee 802.15.4*, https://en.wikipedia.org/wiki/IEEE_802.15.4, (Last visited 23-09-2020), 2020.

[42] Banggood, *Nrf24l01+ si24r1 2.4g wireless power enhanced communication receiver module*, https://www.banggood.com/NRF24L01+-SI24R1-2_4G-Wireless-Power-Enhanced-Communication-Receiver-Module-p-1056647.html?cur_warehouse=CN, (Last visited 23-09-2020), 2020.

[43] LastMinuteEngineers, *How nrf24l01+ wireless module works interface with arduino*, https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/, (Last visited 22-09-2020), 2020.

[44] Wikipedia, *Ant(network)*, https://en.wikipedia.org/wiki/ANT_(network), (Last visited 22-09-2020), 2020.

[45] Electronicdesign, *What's the difference between bluetooth low energy and ant?*, https://www.electronicdesign.com/markets/mobile/article/21796086/whats-the-difference-between-bluetooth-low-energy-and-ant, (Last visited 22-09-2020), 2012.

[46] L. M. ENGINEERS, *How 433mhz rf tx-rx modules work interface with arduino*, https://lastminuteengineers.com/433mhz-rf-wireless-arduino-tutorial/, (Last visited 23-09-2020), 2020.

[47] Digikey, *Bluetooth v5.0 transceiver module bl651*, https://www.digikey.dk/product-detail/en/laird-connectivity-inc/453-00006/453-00006CT-ND/9608593, (Last visited 29-09-2020), 2020.

[48] ——, *Digi xbee 3 802.15.4*, https://www.digi.com/resources/library/data-sheets/ds_xbee3-802-15-4, (Last visited 24-09-2020), 2020.

[49] Digi-key, *Xb3-24arm-j*, https://www.digikey.com/product-detail/en/digi/XB3-24ARM-J/602-2198-ND/8130944?WT.z_cid=ref_neda_dkc_buynow_digiintl&utm_source=ecia&utm_medium=aggregator&utm_campaign=digiintl, (Last visited 24-09-2020), 2020.

[50] RS-online, *Rf solutions alpha-tx433s sender rf-modul*, https://dk.rs-online.com/web/p/rf-moduler/6666744?cm_mmc=DK-PLA-DS3A-_-google-_-CSS_DK_DK_Computere_og_enheder_Whoop-_-(DK:Whoop!)+RF-moduler-_-6666744&matchtype=&aud-827186183886:pla-476229990919&gclid=CjwKCAjw5Kv7BRBSEiwAXGDElWgGpaxkd464adbWjbEv2s-VCiVbcu6XQOL1vvN0r1BXZR_eS2e7hBoCziYQAvD_BwE&gclsrc=aw.ds, (Last visited 29-09-2020), 2020.

[51] Digi-Key, *D52qd2m4ia-tray*, https://www.digikey.dk/product-detail/en/garmin-canada-inc/D52QD2M4IA-TRAY/1094-1023-ND/6149451?cur=EUR&lang=en, (Last visited 29-09-2020), 2020.

[52] Garmin, *D52 ant soc module series*, https://media.digikey.com/pdf/Data%20Sheets/Garmin%20Canada/D52_ANT_SoC_Module_v.2.3_DS.pdf, (Last visited 29-09-2020), 2020.

[53] Bluetooth, *Core system package [low energy controller volume]*, https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=286439, Standard, 2014.

[54] M. Afaneh, *Intro to bluetooth low energy*, https://www.novelbits.io/introduction-to-bluetooth-low-energy-book/, (Sidst besøgt 02-11-2020), 2018.

[55] M. Gast, *802.11 wireless networks: The definitive guide*, 2005.

[56] D. Parsons, *The mobile radio propagation channel*, 2001.

[57] U. of Hawaii, *Encoder primer*, irtfweb.ifa.hawaii.edu/~tcs3/tcs3/0306_conceptual_design/Docs/05_Encoders/encoder_primer.pdf, (Sidst besøgt 24-09-2020).

[58] Circuits$_{\text{t}}$oday, *Potentiometer*, https://www.circuitstoday.com/potentiometer, (Sidst besøgt 05-12-2020).

[59] Farnell, *Gyroscopes*, https://dk.farnell.com/sensor-gyroscope-technology, (Sidst besøgt 02-10-2020).

[60] H. to mechatronicsl, *Arduino and mpu6050 accelerometer and gyroscope tutorial*, https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/, (Sidst besøgt 02-10-2020).

[61] ——, *How to track orientation with arduino and adxl345 accelerometer*, https://howtomechatronics.com/tutorials/arduino/how-to-track-orientation-with-arduino-and-adxl345-accelerometer/, (Sidst besøgt 02-10-2020).

[62] K. Tuck, *Tilt sensing using linear accelerometers*, https://www.thierry-lequeu.fr/data/AN3461.pdf, (Sidst besøgt 02-10-2020), 2007.

[63] L. control ab, *Tilt sensing using linear accelerometers*, http://lagge.se/files/2009/11/TechNote_02_AA.pdf, (Sidst besøgt 02-10-2020), 2005.

[64] Digi-key, *Apply sensor fusion to accelerometers and gyroscopes*, https://www.digikey.com/en/articles/apply-sensor-fusion-to-accelerometers-and-gyroscopes, (Sidst besøgt 02-10-2020), 2018.

[65] Honeywell, *Compass heading using magnetometers*, https://cdn-shop.adafruit.com/datasheets/AN203_Compass_Heading_Using_Magnetometers.pdf, (Sidst besøgt 24-09-2020).

[66] , *Gauss (unit)*, https://en.wikipedia.org/wiki/Gauss_(unit), (Sidst besøgt 024-09-2020).