CHALMERS
UNIVERSITY OF TECHNOLOGY

Electrical Autonomous Shopping Cart

Development of an Electrically Powered Autonomous Shopping Cart

Bachelor's thesis at the Department of Electrical Engineering

Jacob Carlback Fredrik Eriksson Jacob Gästrin
Martin Hjorth Jonatan Holmström Oskar Thompson

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

EENX15-20-27

Electrical Autonomous Shopping Cart

Development of an Electrically Powered Autonomous Shopping Cart

Jacob Carlback Fredrik Eriksson Jacob Gästrin
Martin Hjorth Jonatan Holmström Oskar Thompsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Gothenburg, 2020

Electrical Autonomous Shopping Cart
Development of an Electrically Powered Autonomous Shopping Cart
EENX15-20-27

Supervisor: Karinne Ramirez-Amaro, Department of Electrical Engineering
Examiner: Yiannis Karayiannidis, Department of Electrical Engineering

Bachelor's Thesis 2020
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg

Cover: Picture of the prototype that has been developed during the project.

Gothenburg, Sweden 2020

Abstract

The purpose of this report is to present the development of an electrical autonomous shopping cart prototype that can follow a customer in a grocery store, facilitating daily shopping. After a literature study, the development of such a cart was divided into three main work areas; the first focusing on locating the customer, the second on avoiding collisions, and the third on designing and assembling a prototype cart. To locate a customer, ultrasonic sensors were mounted on an aluminium frame to triangulate the location of a handheld device. A camera based system capable of image processing was implemented to detect and avoid collisions. The camera was also mounted on the frame, as well as a drive train consisting of components for propulsion, steering, and control. In its finished state, the prototype cart is able to slowly follow a person carrying a handheld device. It also stops when encountering obstacles. These results prove that the systems could be adapted to a full scale system after further development.

Acknowledgements

We would like to thank Karinne Ramirez-Amaro for all the support, as well as Rikard Karlsson and the CASE association for the help with equipment needed for assembly.

Special thanks to Axel Karlsson for providing the shopping basket.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

- CA - Collision Avoidance
- CAD - Computer Aided Design
- CD - Collision Detectance
- CU - Customer Unit
- EMF - Electromotive Force
- FTC - Follow The Customer
- PID - Proportional–Integral–Derivative
- PWM - Pulse Width Modulation
- RPM - Rotations Per Minute
- USS - Ultra Sonic Sensor

1 Introduction

The fast development in electrification and autonomous systems is expanding into more fields, for instance in daily shopping. While shopping for groceries, the shopping cart can often become quite heavy and hard to manoeuvre. To a number of customers an electrically assisted cart could therefore be helpful in steering and manoeuvring an otherwise heavy cart. To implement such an aid, there are many possibilities. One idea is facilitating the propulsion by somehow reducing the force a customer needs to apply on a cart in order to move and steer it. Another idea is to completely eliminate the physical effort behind manoeuvring a cart, by having an autonomous cart that follows the customer in the store while he or she is shopping. The latter described implementation was chosen to be the subject of this bachelor thesis.

1.1 Aim

The aim of this project is to design and assemble a prototype of an autonomous electrical shopping cart. This prototype will be a smaller scale model which represents a full sized shopping cart, that is able to autonomously drive, make turns, follow a customer in a grocery store and avoid collision with objects. The purpose of the prototype is to develop and test collision detection, collision avoidance, and following a customer autonomously.

In the following list some more specific goals for the project are listed. These are standalone goals set to help achieve the aim. The goals for the prototype cart are:

- Distance measuring from the cart to a customer, both sideways direction and forward direction.
- Modular design to facilitate in its testing purposes. This means assembling and disassembling are quick, and there are space available for easily adding and relocating components.
- Detect objects in the direct path between the cart and a customer.
- Be able to drive around any detected object in its path towards a customer.

1.2 Problem Description

This subsection identifies the problems in the work towards fulfilling the aim and goals of the project.

- A challenge regarding the design process is to construct some kind of basis, where all the required components can be attached. Since this will be the foundation of the prototype cart, and will carry the load of all the other components, it is necessary that a durable material is chosen.
- How the framework is assembled is also of importance. The construction needs to be solid and keep the components locked in place.
- To enable the cart to drive towards a customer, an electric drive train has to be designed, implemented, and controlled with a control system.
- The intended environment for the cart is a grocery store. This means that there is a risk for objects blocking the path between the cart and the customer. A system for avoiding these collisions will therefore have to be created.
- The cart needs to keep track of the customers position, as the customer moves around.

Regardless of how solutions to the above problems are solved, there is also a problem of hardware and software interaction. Software implementations are limited to the hardware and its design, and especially limited to hardware provided data. Hence, the hardware solutions should satisfy the software requirements and vice versa.

1.3 Limitations

Due to budget and time constraint, some limitations was set to the project. The budget was 5000 SEK and the time frame was 20 weeks. This entails constraints on the level of complexity of the possible solutions.

- The size of the prototype was limited to carry an ordinary shopping basket.
- With a smaller prototype size a dedicated system for braking was not considered necessary as this prototype have a relatively low weight.
- No automatic charging system was considered. Battery charging must therefore be done manually.
- In calculating the torque required for the motors, an evenly distributed load was assumed.
- The operating environment for the cart is assumed consist of a flat monochrome surface with an inclination of less than one degree.
- The lighting inside a grocery store is assumed to be good, thus only problems caused by slight changes in lighting will be handled. Problems such as soft shadows.

1.4 Proposed Solutions

With some limitations set, a search for possible solutions to the problems described in the previous section could begin. The following list presents these solutions.

- Building the basis of the prototype out of square steel tubes was at first considered. The idea was to weld the tubes together into a rectangular frame. A steel tubes is heavy by itself and would contribute to a high overall load, and welded joints would make the prototype less modular compared with screw joints.
- Setting up multiple cameras on the ceiling could be used to set up a grid that the cart and customer can be placed in and then use simple XY-coordinates to follow the customer. Due to the amount of cameras needed, this solution was deemed too expensive and extensive, therefore it was not chosen.
- Ultrasonic sensors could be used to measure the distance and angle of the customer relative to the cart. Ultrasonic sensors are very cheap and work well in the intended range. The same method could also be implemented using infrared diodes. Due to the simplicity of ultrasonic sensors, this sensor type was chosen.
- A solution for detecting obstacles could involve a camera and computer for image processing.

1.5 Related Work to Similar Projects

There are other similar project to this one. A company that has been developing this type of product idea is LG Corporation[1]. There are also people who have been looking into human following robots, like the engineers B. Heckendorn and J. Robinson [2], who built an autonomous luggage robot. There have also been some peer reviewed papers written on this subject. For example [3], where B. V. Pradeep, E. S. Rahul, and R. R. Bhavani propose using a smartphone to track the movement of the customer and sending the customers positional data to a cart via Bluetooth, or Q. K. Dang and Y. S. Suh[4] who evaluate the usage of an infrared camera in a Wii remote to communicate position and change in position between a robot and a person. They also speculate that a possible use for them is implementing it in shopping carts. There are multiple other examples of similar ideas, like Amazons smart shopping cart [5], that shows the interest in products like this. Further related work for following a customer and avoiding a customer is presented in section 3.1 and 5.1 respectively.

1.6 Report Outline

In this section the contents of each chapter is presented in the list below.

- Chapter 2 - the structure of the project is given, and is presented as four project phases; plan, develop, assemble and assess.
- Chapter 3 - gives an in depth description of the development process of an ultrasonic based triangulation system.
- Chapter 4 - describes how the cart moves towards the customer.
- Chapter 5 - focuses on development of a camera based system for detecting and avoiding obstacles.
- Chapter 6 - describes the design process of the prototype, explaining and visualising the design choices.
- Chapter 7 - describes how the components are assembled, and how the FTC and CA systems are integrated into a complete system.
- Chapter 8 - presents the results a prototype cart, in terms of its movement and some performance data.
- Chapter 9 - includes a discussion about the results, both regarding fulfilment of the goals as well as environmental and ethical aspects.
- Chapter 10 - includes a conclusion and discussion about potential further development and future work.

2 Project Structure

In this chapter, an overview of the how the project was structured is given. The structure include four project phases: planning, development, assembly, and assessment, and can be seen in figure 2.1 below.

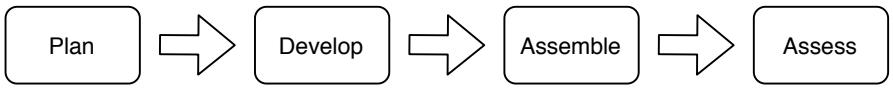


Figure 2.1: Project structure.

In the following subsections a more detailed description of what the different project phase means is given.

2.1 Project Planning

The first phase was project planning, where the goals previously presented in chapter 1 of the project were decided. A work plan for reaching the goals was established, where the project was divided into three work areas. Two of the work ares was the development of two software systems, one for following a customer, and one for avoiding collisions. The third work area was designing the prototype cart. Organising the project this way increased time efficiency, as different work tasks could be completed simultaneously. An overview of the projects work areas can be seen in figure 2.2 below.

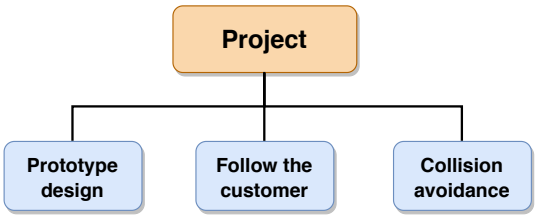


Figure 2.2: Project work areas.

A literature study was presented in the planning stage [6]. In this study, several solutions related to the specified problems were mentioned. The solutions considered viable were selected for further investigation in the development stage.

2.2 Development of the Sub-Systems

The second project phase was designing and developing solutions for solving the identified problems as previously described in section 1.2. The development emanated from the proposed solutions given in section 1.4. The development was also split up between group members according to the three work areas described in the previous planning phase. This resulted in the development and design of three sub-systems; a Follow the Customer (FTC) system, a Collision Avoidance (CA) system, and a physical prototype.

2.3 Assembly of the Sub-Systems

The third phase was to assemble the individual solutions in the development phase into a complete system. The hardware assembly was a continuous process. When parts had arrived and were ready, they could be mounted. The software was developed separately for each subsystem and integrated after the hardware assembly was completed.

2.4 Assessment of the Completed System

The last phase of the project was to evaluate the complete system, by testing how the prototype behaves in operation, and assess if the aim was properly achieved. The evaluation phase included evaluation of how the different systems operate both individually and as a single unit. With the subsystems merged into one system, it was possible to begin testing if the prototype performs as expected. A complete assembly also facilitated for further adjusting and refining. Another aspect of the evaluation was to conclude, and discuss what potential there is for further development.

3 Follow the Customer System

One vital part of making the prototype cart follow a customer, is the communication of customer location relative to the prototype cart. In this chapter, the development of a FTC-system, as introduced in section 2.2, will be presented. The solutions to this will be described, as well as how the solutions changed from the planning phase.

3.1 Related Work

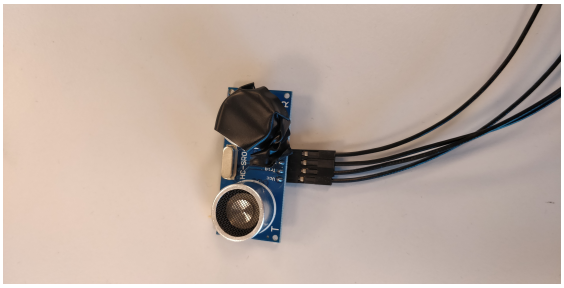
After searching for solutions regarding position tracking, two relevant options were found. These options were to use ultrasonic sensors to triangulate the position of a person as seen in [2], and to use positional data from a persons mobile phone via Bluetooth, as seen in [3]. The Bluetooth option would require the cart to create and follow a path based on the positional data from the phone, while the option with ultrasonic sensors only required two microcontrollers, one on the cart and one in a in the customers hand, in a customer unit, to get sufficient data to keep track of a person. When reviewing these options, it was argued that the Bluetooth option was more difficult, and would thus require more time compared to the ultrasonic option, which led to the ultrasonic option being chosen, and the Bluetooth option was discarded.

3.2 System Requirements

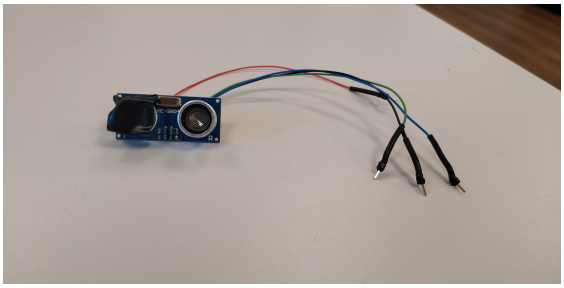
The FTC-system has two required outputs, the angle of the customer relative to the cart, and the distance between the cart and customer. The measured distance requires an accuracy of ± 10 cm, and the angle $\pm 10^{\circ}$. These requirements were based on the intended use of the prototype, as it was argued that ± 10 cm does not make a difference when a customer is reaching for the cart. An additional requirement was that the angle and distance could be updated at least once per second.

3.3 Hardware Description

For the FTC-system, two types of sensors were chosen, ultrasonic transceivers and radio communication modules. For the ultrasonic sensors, SparkFun HC-SR04 [7] was chosen. This sensor has a transmitter and a receiver, and can be seen in figures 3.1a and 3.1b below. The microcontrollers used with the sensors are an Arduino Nano for the customer unit, and an Arduino Mega on the cart. The customer unit is a handheld device which can communicate with the cart.



(a) The ultrasonic transmitter used as part of the customer unit.

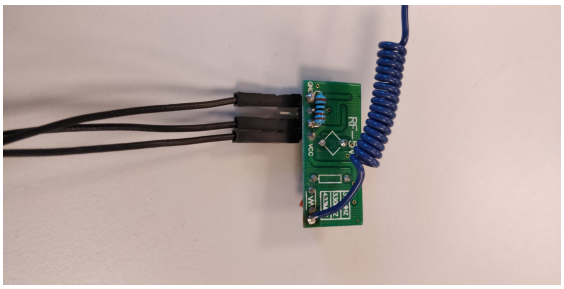


(b) One of the the ultrasonic receivers used on the cart.

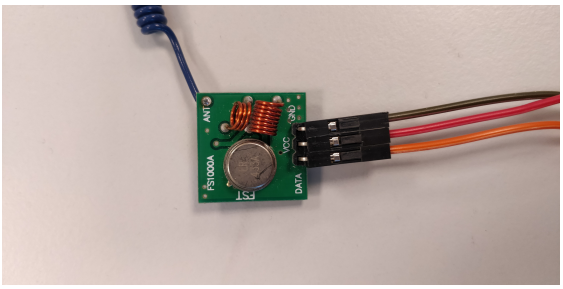
Figure 3.1: The ultrasonic transceivers used in the project, with a modification to only send or receive.

One pin is for +5V, and one for ground. The remaining two pins are called "echo" and "trig". When a digital 1 is written to the trig pin, a sequence of sound waves at 40 000 Hz is emitted. This is called a "ping". When a ping is emitted, the echo pin writes a digital 1 until the receiver receives a ping [8]. The length of which the echo pin writes a 1 can be used to calculate the distance from the sensor to a reflective object.

The other sensor in the FTC-system is two radio communication modules, a transmitter called FS1000A [9] and a receiver called XY-MK-5V [9]. The modules can be seen in figures 3.2b and 3.2a below.



(a) The radio receiver used as part of the customer unit.



(b) The radio transmitter used on the cart.

Figure 3.2: The radio transmitter and receiver setup used in the FTC-system.

The modules have limited reach without antennas, thus a pair of 433 MHz coil loaded antennas has been built to the specifications of the modules [10], as shown in figure 3.3.

3.4 First Iteration of the FTC System

During the planning stage, the functionality of the sensors were not know in detail. Therefore, the first solution did not work as intended. However, the theory is still applicable and therefore presented in this section.

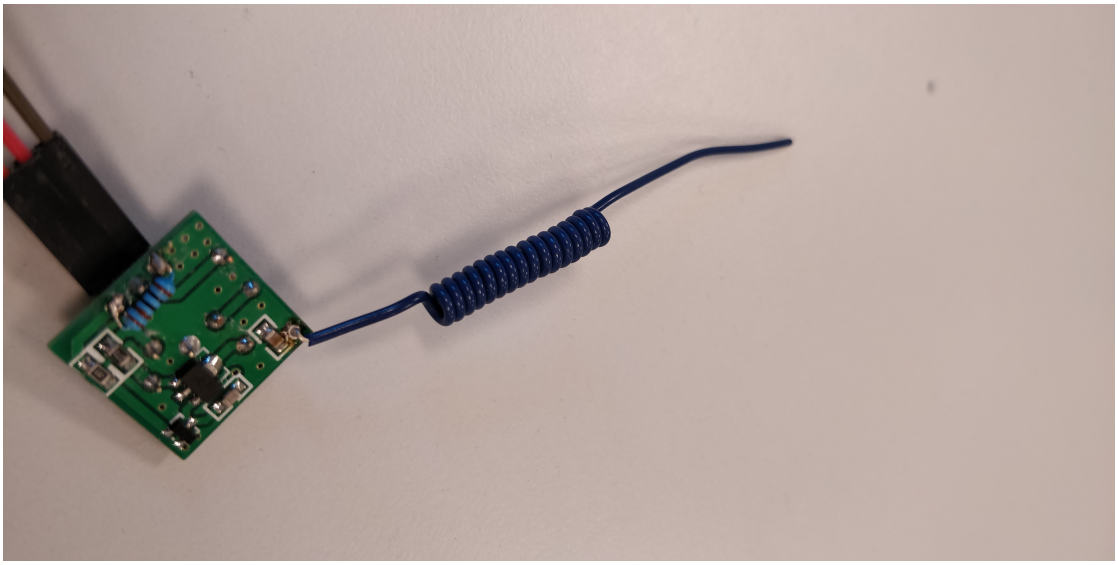


Figure 3.3: One of the coil loaded antennas built to amplify the range of the radio transmitter and receiver.

3.4.1 Theory

The plan for the sensor setup was to have three receivers and one transmitter on the cart, and one transceiver on the customer unit (CU). With these sensors the distances noted as D_1, D_2 and D_3 in figure 3.4 can be calculated from the times measured.

To measure the distances between the sensor on the CU and each of the sensors on the cart, the Arduino function pulseIn [11] was used. PulseIn measures time in microseconds, and the speed of sound was rounded to 340 m/s, which equals to 0.034 m/μs. The result d is in meters and time t is in microseconds, and the formula is shown in equation (3.1).

$$d = 0.034 * t \tag{3.1}$$

These measurements can then be put into equation (3.2), the cosine rule,

$$D_2^2 = D_1^2 + D^2/4 - D_1 \cdot D \cdot \cos\beta \tag{3.2}$$

and the solve it for the requested β in equation (3.3).

$$\beta = \arccos\left(\frac{D_1}{D} + \frac{D}{4 \cdot D_1} - \frac{D_2^2}{D_1 \cdot D}\right) \tag{3.3}$$

Then θ is easily solved in equation (3.4). The intention was to use θ and D1 as parameters to the control system.

$$\theta = 90^\circ - \beta \tag{3.4}$$

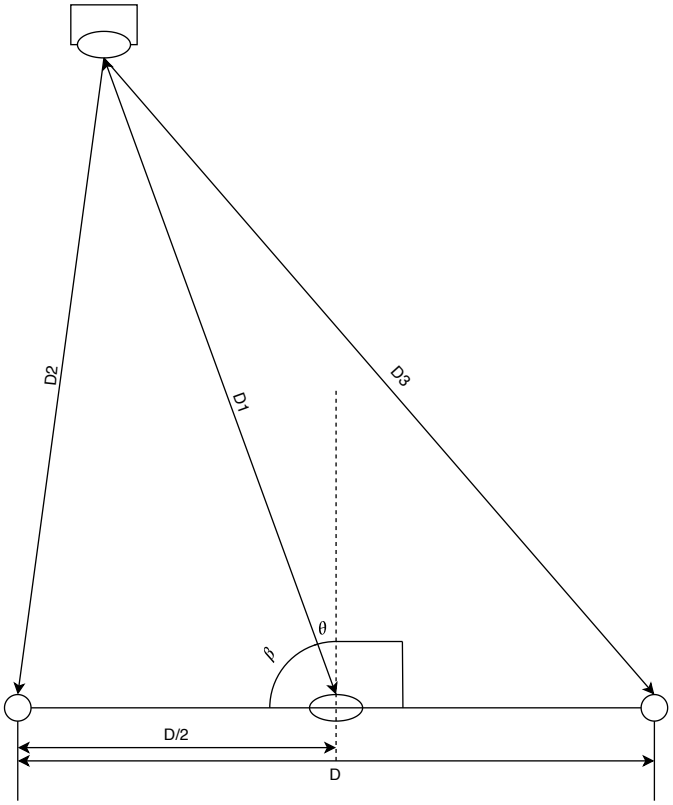


Figure 3.4: Schematic of how the sensors are supposed to be set up. The circles are receivers while the ellipses are transceivers. The rectangle with an ellipse on it, is the customer module. The distance D is the width of the cart.

3.4.2 Testing

The first attempt to solve the FTC-system involved using three of the USS on the cart, with one being used as both receiver and transmitter and two only as receivers. The CU would, using a transceiver, receive a ping from the transmitter on the cart, and send it back. This ping would then be received by the three receivers on the cart, as seen in figure 3.4. By measuring the time between the initial ping and the final received ping, the angle and distance could be calculated as described in section 3.4.1. Before the CU was implemented, the three sensors were tested against a wall.

To ensure the functionality of the sensors, each transceiver was tested separately. This was done by writing a digital 1 to the trig pin, and then listen for the echo using pulseIn. Using equation (3.1) the distance was calculated successfully. The next step was to use all three sensors simultaneously, as seen in the algorithm 1 below.

Algorithm 1: Pseudocode for the first attempt to use all sensors simultaneously.

```
trig middleSensor;
timeMid = pulseIn(middleSensor);
timeLeft = pulseIn(leftSensor);
timeRight = pulseIn(rightSensor);
```

During testing, problems occurred. Upon debugging, it was found out that the USS needed to send a ping in order to be able to receive one back. The code was changed accordingly, however it still did not work. Further research was done regarding the the pulseIn [11] function, and it was found out that it is a so called "blocking code", which means that no other code can run until the blocking code is complete. This can be seen in the source code for pulseIn A.1 that runs the assembler code seen in Appendix A.1.3, which has several loops and does not exit until a specific value is reached. This entails that the processor will run this code and no other until it is complete. As a result of this, measurements for each sensor had to be done separately in a sequence, as seen below in algorithm 2.

Algorithm 2: Pseudocode for the sensors working sequentially.

```
trig middleSensor;
timeMid = pulseIn(middleSensor);
trig middleSensor;
trig leftSensor;
timeLeft = pulseIn(leftSensor);
trig middleSensor;
trig rightSensor;
timeRight = pulseIn(rightSensor);
```

This was tested with each sensor receiving the echo of the initial ping, and the results were successful; an accurate calculation of the distance from the wall to each sensor. The next step was to include the CU. Since it had been discovered that a USS had to transmit in order to receive, the CU had to have two USS, one with a blocked transmitter that only listens continuously, and one which transmits when a ping has been received. This led to the decision of establishing another form of communication between the cart and the CU. Due to availability, the radio modules mentioned in section 3.3 was chosen. When radio communication was established, the code was modified accordingly, as seen in algorithm 4, and the transmitter on the middle sensor was blocked.

The radio communication setup consists of a transmitter that sends out a message when requested, and a receiver that always is looking for a message. If it can not sense anything, it will keep amplifying the input signal until it can read something, even if it is only noise. Therefore it is important to ping regularly. When a message is received, it is converted to a string and compared with a predefined string. If they are the same, a ping is sent out and the string is reset, and this can be seen in the pseudocode presented in algorithm 3 below.

Algorithm 3: Pseudocode for the radio receiver.

```
incMessage
String receive
for(incMessage.length)
    receive += (char)incMessage[i]
if(receive==correct message)
    send Ping
receive=""
```

Algorithm 4: Pseudocode for the sensors working sequentially with radio connection.

```
request ping;
trig middleSensor;
timeMid = pulseIn(middleSensor);
request ping;
trig leftSensor;
timeLeft = pulseIn(leftSensor);
request ping;
trig rightSensor;
timeRight = pulseIn(rightSensor);
```

This solution worked as expected, however while monitoring the results it was discovered that the angle varied between -20 and +20 degrees when the CU was at 0 degrees relative to the cart. This variance did not change when the CU was moved. Furthermore, the calculated distance had a variance of about 40 cm. Both angle and distance had variance above the specified requirements for the prototype cart. It was discovered that the processing time for the radio messages varied substantially. This was due to the radio transmitter blocking the code until the message was sent and the radio receiver blocking the code on the CU until the message was received in full. This led to a significant change in the system requirements being made. This will be discussed in section 3.5.

3.5 Second Iteration of the FTC System

Due to the limitations described in 3.3 of how the USS work, the setup changed during the project, and is described in the following sections 3.5.1-3.5.2

3.5.1 Updated Theory

The new setup only has 2 receivers to measure distances, and one transmitter on the CU, as seen in figure 3.5. The setup now requires another way to communicate when to send out a ping. This was implemented using the radio modules. The cart sends a radio transmission to the CU requesting a sonic ping, which then is used to measure the distances D1 and D2 in figure 3.5. Instead of angle and distance, the

parameters for the control system were changed to the distance difference between D1 and D2, and an approximate distance to the customer l, which was calculated based on D1 and D2 as seen in equation (3.5).

$$l \approx \frac{D1 + D2}{2} \tag{3.5}$$

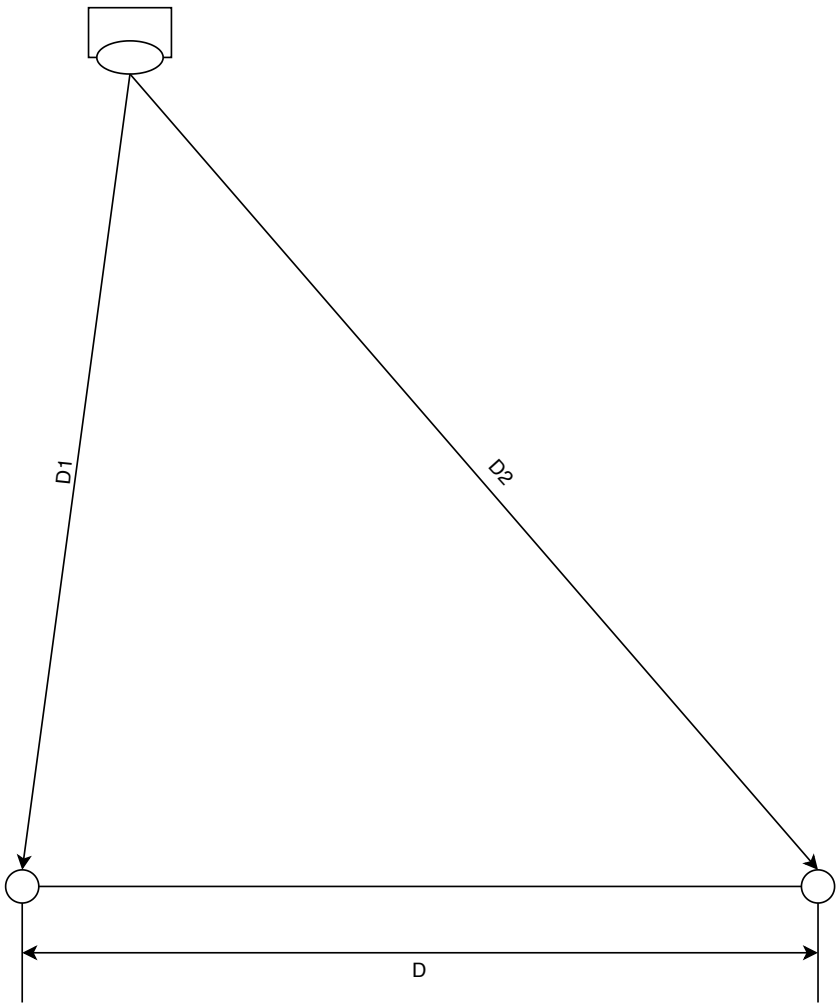


Figure 3.5: Schematic of how the sonic sensors are set up.

3.5.2 Realising the Updated Theory

To counter the variance described in section 3.4, it was found that both sensors on the cart had to receive the same ping from the CU in order to achieve sufficient accuracy. The idea was to attach interrupts [12] to the echo pin on each sensor on the cart. These interrupts would trigger on a change of value, and using the Arduino standard function micros [13], the time which the echo pin was high could be measured. Micros keeps track of the number of microseconds which has passed since the microprocessor was initialised. When the echo pin goes from 0 to 1 on a given sensor, an interrupt is triggered which saves the current time using micros. When the pin goes from 1 to 0 the same interrupt is triggered and the time is

checked again. These values are then subtracted, thus resulting in the time which the pin was 1, which corresponds to the distance to the customer. Using equation (3.1), D1 and D2 in figure 3.5 could be calculated, and thus the difference could be calculated. To get an approximate distance to the customer the mean of D1 and D2 was calculated.

The delay mentioned in section 3.4 was still present, however since both sensors listened to the same ping, this error of D1 and D2 were the same for a given ping, but varied from ping to ping. To counter this, it was decided to use the mean value of the last ten recorded values as the parameter for the control system, both regarding distance to the customer and distance difference. To achieve this each recorded value was saved in an array of length ten. When this array was full, the next recorded value would take the first place in the array. This led to the variance going from $\pm$ 5 cm to $\pm$ 0.5 cm, which was more accurate than the requirement. The trade-off was that the system responded to change with a delay of about two seconds, however this was judged to be acceptable. The distance to the customer had an accuracy of a few centimetres due to it being approximated based on D1 and D2. If the delay was too long, it was made possible to calculate the mean values based on fewer values, which would reduce the delay but increase the variance.

This FTC-system could determine how far away the customer is with a few centimetres accuracy.

4 Propulsion, Steering and Control System

An essential feature of the cart is to be able to drive. This entails several problems; What steering should be chosen? What motors are required? How are the motors controlled? How is the speed of the motors measured? These problems will be discussed in this chapter.

The aim for the project is to develop systems which can be applied to a full scale shopping cart. The propulsion, steering and control system were considered the least important since they were specifically developed for the prototype, not for a full scale system. Therefore less research was done in this area, and no related work will appear in this chapter. The solutions used were chosen based on cost, effectiveness and simplicity.

4.1 System Requirements

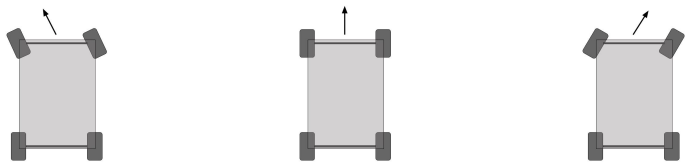
Motors will be chosen based on the following requirement; the system shall accelerate from 0 to 1.5 m/s in less than 2 s, with a maximum total weight of 15 kg. The speed of the motors will be controlled through feedback control, therefore the speed must be measurable.

4.2 Steering

Two types of steering were considered, Ackermann and Differential. These were chosen based on complexity and familiarity.

4.2.1 Ackermann Steering

Ackermann steering is most commonly used in cars [14] and works by having the front or back wheels turn. Using this steering method, the engine does not determine the turning speed. The angle which the wheels are turned will determine the rate which the vehicle turns, as shown in 4.1. This method of steering requires several moving parts, and a method of measuring the turning rate. This was deemed too complicated for the project which led to this option being discarded.

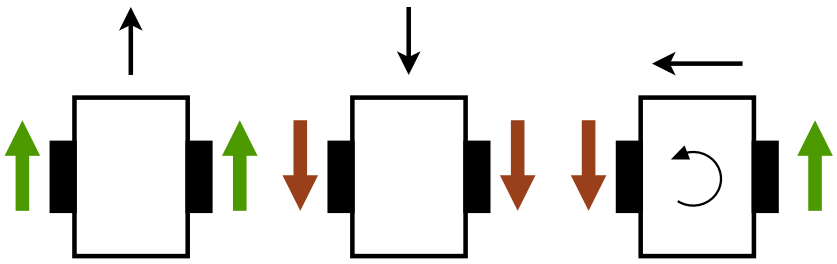


(a) Turning left (b) Driving straight (c) Turning right

Figure 4.1: Ackerman steering basics.

4.2.2 Differential Steering

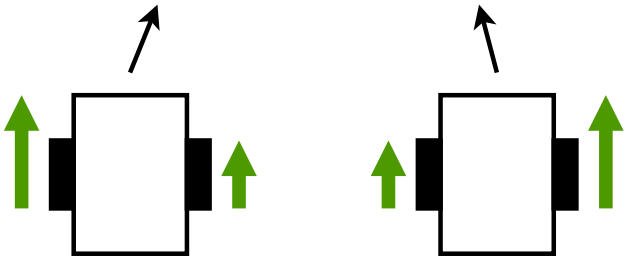
Differential steering uses two independently controlled motors to control both turning and velocity. If both motors run at the same speed and direction, the cart will go straight forward, or straight backwards. If the motors go in the same speed but in different directions, the cart rotates around its central point. See figure 4.2.



(a) Forward drive (b) Backwards (c) Rotate

Figure 4.2: Differential steering.

If the motors are driven in the same direction but at different speeds the cart turns around a radii, the larger the difference between the two, the sharper the turn and the smaller the turning radii. See figure 4.3.



(a) Turn right (b) Turn left

Figure 4.3: Turning with differential steering, vectors indicate velocity.

4.3 Hardware Description

There are three components needed in order to drive the cart, while being able to control its speed. Motors, motor controllers and a battery. The most important choice is what kind of motors will be used, the other components are then selected based on what motors were chosen.

4.3.1 Motors

While several different motor types exist that could be used for this application, it was decided that regular brushed DC motors was the best option for this prototype. This was due to their low cost, high power output, and ease of regulation through voltage and current control [15].

In order to get an estimation of what motors would be suited for the prototype, a simple representation of the theoretical system was made in MATLAB, and this is shown in figure 4.4 below.

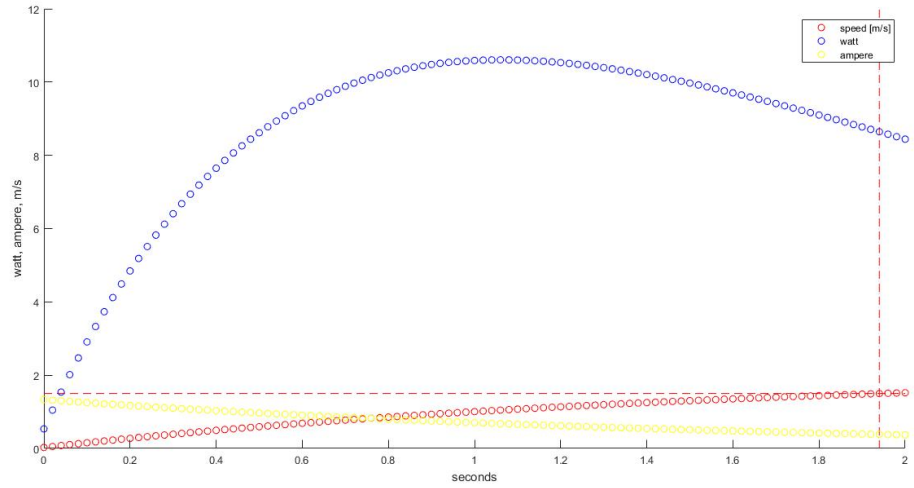


Figure 4.4: A MATLAB plot of the minimum motor requirements.

Different motor parameters was tested to see if the system requirements were met, and evaluation of their effect on acceleration, power and current drawn could be made.

Given two motors and a total mass of $m = 15$ kg it was concluded that 11 W motors were a minimum in order to reach the system requirements which were to accelerate from a stand still $v = 0$ to $v = 1.5$ m/s or typical walking pace in less than 2 s. As assembly of the system had not started at this stage, no losses were included in the calculations due to the uncertainties involved. This resulted in a lower power requirement for the motors than a calculation including friction. The calculations were also done by approximating the motor torque output to be linear in relation to the motor RPM [15].

As specified above, the required velocity is $v = 1.5$ m/s while the diameter of the wheels are assumed to be $d_{\text{wheel}} = 0.1$ m. From these values, the required revolutions per minute (RPM) can be calculated, as seen in equation (4.1)

$$\text{RPM} = \frac{v \cdot 60}{d_{\text{wheel}}} \tag{4.1}$$

With the required wheel RPM, the torque equation of a DC-motor [15], can be seen in equation (4.2)

$$M_{\text{motor}} = M_{\text{Stall}} - \text{RPM}_{\text{current}} \frac{M_{\text{Stall}}}{\text{RPM}_{\text{max}}} \tag{4.2}$$

was used to calculate the motor torque (M) for different RPM values. This simplification is based on the steady-state relationship between voltage across the rotor and the motor's torque output. The developed torque of an permanent magnet brushed DC motor is proportional thought a constant λ to the current flowing through the rotor windings. This current can in an steady state condition be calculated through Ohms law and is described in the following equation (4.3) where U_{feed} refers to the voltage across the motor, R is the rotor resistant. I is the current through the rotor and U_{EMF} is the counter-electromotive force, and counteracts the voltage when the motor rotates [15].

$$I = \frac{U_{\text{feed}} - U_{\text{EMF}}}{R} \tag{4.3}$$

The EMF is proportional to the RPM of the motor. This entails a proportional relationship between the current and the torque. The torque can then be put into newtons second law, as seen in equation (4.4), to calculate the acceleration

$$a = \frac{F}{m} = \frac{2 \cdot M}{d_{\text{wheel}} \cdot m} \tag{4.4}$$

which is used with all the other values to calculate the power P in equation (4.5).

$$\text{P} = F \cdot v = v \cdot m \cdot a \tag{4.5}$$

As no losses were included in the calculations, it was decided to err on the side of caution when selecting the motors; the limited selection of motors that met the requirements within the budgetary constraints facilitated the choice of motors. The motors chosen were a pair of RS PRO brushed DC Geared motors with an maximum output speed of 366 RPM and an maximum power output of 50 watts seen in figure 4.5. This was deemed more than sufficient for the prototype.

Figure 4.5: RS PRO - Brushed DC Geared Motor, Output Speed 366 RPM, 12 V dc.

4.3.2 Battery and Motor Driver

In order to move without external power, the system needs some sort of battery. There exists many different battery chemistries, including ones based on elements such as lithium or nickel. The lithium chemistry is lightweight and energy dense, however, this type of battery was considered too expensive. Instead, it was decided to use a 12 V lead-acid battery due to its low cost, ruggedness, ease of charging and availability. Although this choice resulted in a heavier system than otherwise, this was not considered a drawback since the cart is designed to carry a fully loaded shopping basket in its normal operational state.

As the motors require more power than the Arduino micro processor can supply, a motor controller is needed between the Arduino and the motors. A motor controller takes digital signals from the micro processor and regulate the motor speed accordingly. For this, two MIKROE-3613 DC motor controller from Mikroelektronika [16] was chosen, one for each motor. It has a maximum output current of 6 A which reaches the motor requirement. With this motor controller it is possible to control the rotational direction of the motor. When using differential steering, the motors can rotate in opposite directions which can be used to reduce the turn radius.

Furthermore, this motor controller has a pulse width modulation (PWM) signal input. This give an easy way of controlling it with the Arduino, as the Arduino have digital outputs with PWM support. An Arduino library function called analogWrite can be used to set a PWM value between 0-255, where 0 is 0 % PWM duty cycle and 255 is 100 % duty cycle. In PWM, the duty cycle percentage is the active duration in one period of a pulsed signal. In reality, the PWM value will correspond to a certain motor speed and consequently wheel RPM.

4.3.3 Rotary Encoders

There were only one requirement when choosing encoder; to be able to handle the same speed as the motors. However, due to issues with the supplier, the encoders originally chosen never arrived. Because of this delay, another model was used. This was more expensive, and caused delays in the project.

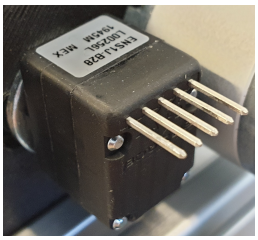


Figure 4.6: The rotary encoder used for the prototype, Bourns ENS1J-B28-L00256L - 256 CPR 5V 3000rpm.

4.4 Feedback Control

While the motor speed can be controlled via the motor controllers, a set PWM value results in different speeds under different conditions. This means that an empty cart will not behave in the same manner as a fully loaded cart. In order to get consistent behaviour, feedback control will be used. See figure 4.7.

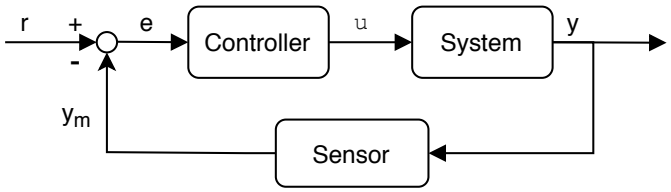


Figure 4.7: Visualisation of a basic feedback loop.

With feedback control a target speed is set. By comparing the target speed with the measured speed, an error is produced. This error is then fed into a controller which adjusts the PWM value sent to the motors. This ensures that the motor speed will follow the target speed, even if the conditions change.

4.4.1 Measuring the Motor Speed

To use feedback control, the speed must first be measurable. To achieve this incremental rotary encoders will be used. An incremental rotary encoder converts the movement of its shaft into a digital output. The output has two channels, which consists of two square waves out of phase, see fig 4.8. With this output the direction and the speed of the motors can be measured using software.

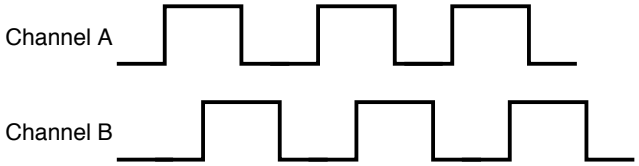
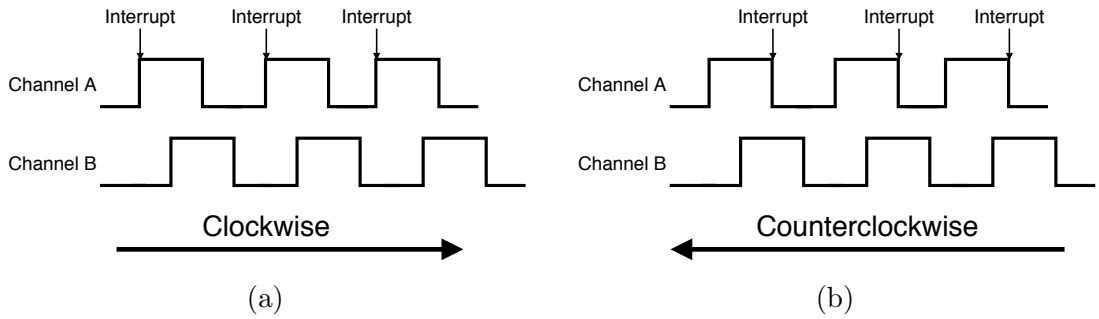


Figure 4.8: Rotary encoder channels.

To measure the speed using the encoders, the output needs to be sent to a microcontroller. Channel A of the encoder is connected to a interrupt pin, and channel B to a digital pin, see fig 7.11 for the complete circuit diagram. The interrupt pin is then configured to trigger a interrupt on a rising edge, as can be seen in figure 4.9 below.



(a) (b)

Figure 4.9: Visualisation of when interrupts are triggered.

Each interrupt runs a function which increments or decrements a variable depending on which direction the shaft is moving in. The function determines the direction by reading the channel B pin, if it is low when the interrupt triggered, it means that the shaft is moving in a clockwise direction. The psueudocode for this is presented in algorithm 5 below.

Algorithm 5: Pseudocode that runs when a interrupt triggers on the right encoder.

```
if(Channel B == LOW)
    encoderValRight++
else
    encoderValRight−−
```

Since the encoders will be mounted on opposite sides, the clockwise direction for the left encoder will be mirrored. Which means that the code will also have to change to reflect this, see algorithm 6.

Algorithm 6: Pseudocode that runs when a interrupt triggers, altered for left encoder.

```
if(Channel B == HIGH)
    encoderValLeft++
else
    encoderValLeft−−
```

With the functions set up, the amount of pulses generated by the encoders, n, can be counted. By sampling the encoders over a set period of time, t, the speed can be calculated. This is done by converting the amount of pulses to revolutions per minute (RPM), see equation (4.6) below, where N are the pulses per revolution.

$$\text{RPM} = 60 \cdot \frac{n}{t \cdot N} \tag{4.6}$$

On the microcontroller the function for this calculation looks slightly different, due to the sampleTime being defined in milliseconds, and the pulses per revolution

being defined as encoderRes, see algorithm 7.

Algorithm 7: Pseudocode to calculate revolutions per second.

```
void updateRPM()
    if (time since last RPM calculation > sampleTime)
        rpmRight = 60 * encoderValRight / (encoderRes * 0.001 * sampleTime)
        rpmLeft = 60 * encoderValLeft / (encoderRes * 0.001 * sampleTime)
        encoderLVal = 0
        encoderRVal = 0
        prevTime = curTime
```

4.4.2 PID Control

With the speed measurable, the error can be fed into the controller. A commonly used controller is the PID controller [17]. A PID controller consists of three terms, the proportional, the integral and the derivative term. Each term is based on the error and calculated independently. The terms are then added together to produce the control signal, see figure 4.10.

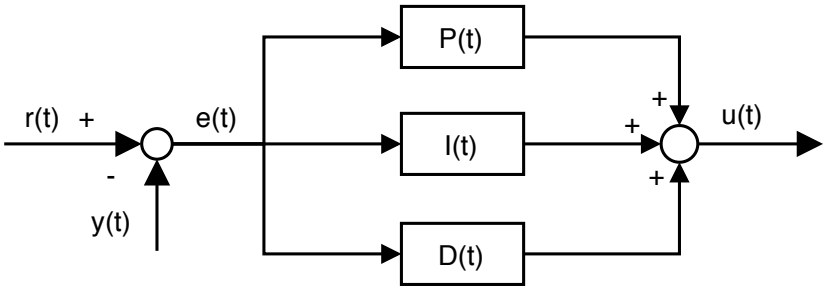


Figure 4.10: Block diagram PID controller.

The proportional term is set by multiplying the error with the proportional gain, K_p, as shown in equation (4.7).

$$P(t) = K_p e(t) \tag{4.7}$$

The integral term not only reacts to the current error, but also accounts for how long the error has persisted. It is calculated by multiplying the accumulated error with the integral gain, K_i. See equation (4.8).

$$I(t) = K_i \int_0^t e(t)dt \tag{4.8}$$

The derivative term is proportional to the rate of change of the error. It is calculated by multiplying the derivative of the error with the derivative gain, K_d. See equation (4.9).

$$D(t) = K_d \frac{de(t)}{dt} \tag{4.9}$$

Adding the terms together produces the control signal, as shown in equation (4.10) below.

$$u(t) = P(t) + I(t) + D(t) \tag{4.10}$$

The gain values K_p, K_i and K_d presented in the equations above, dictates the behaviour of the PID controller and therefore the system. The challenge in using a PID controller lies in configuring these values to get the desired behaviour. Determining these variables can be done by using a mathematical model of the system. However, creating a mathematical model of the system is a time consuming process, since it requires parameters that are unknown. Another approach is to manually tune one gain value at a time in a process of trial and error. This experimental method was chosen to allow for more time in other areas of the project.

The actual PID calculations will be done on a microcontroller. Since a microcontroller runs at a set frequency, the error can not be read continuously. It will instead be read at a predetermined sample time, h. Because of this the function for each regulator term will need to be discretized.

The P-term only depends on the current error, and therefore the discrete version can be translated directly, see equation (4.11)

$$P(n) = K_p e(n) \tag{4.11}$$

Since the I-term includes the integral of the error, it needs to be approximated, this is done by the euler-forward method, shown in equation (4.12).

$$K_i e(n) = \frac{I(n+1) - I(n)}{h} \Rightarrow I(n+1) = I(n) + K_i h e(t) \tag{4.12}$$

The D-term includes the derivative of the error, therefore it also needs to be approximated, this is done with the euler-backwards method, shown in equation (4.13)

$$D(n) = K_d \frac{e(n) - e(n-1)}{h} \tag{4.13}$$

With the regulator discretized, it can now be implemented on the microcontroller. The pseudocode for the controller is shown in algorithm 8.

Algorithm 8: Pseudocode for PID calculations.

```
computePID()
    if(time since last PID calcualation >  sampleTime)
        error = targetSpeed - rpm
        integralTerm = integralTerm + ki * error * sampleTime
        integralTerm = constrain(integralTerm, 0, 255)
        dRpm = rpm - lastRpm
        outputPWM = kp * error 0 integralTerm + kd * dRpm
        lastRpm = rpm
```

4.4.3 Calculating Setpoints for the Motors

As the prototype cart uses differential steering, the two motors does not only control the speed of the system, but also the steering. In order to tune the behaviour of the system, these are calculated independently, and combined into an individual setpoints, one for the left motor, and one for the right.

To get a more reactive behaviour from the cart, the target velocity will be controlled by the customers position. The distance from the customer, l, controls the average speed of the motors by using the following function. See equation (4.14) below. β and γ are variables that can be tuned in order to get the desired behaviour of the cart. β controls the maximum reachable speed and γ controls how steep the ascent towards that maximum value is.

$$V_{\text{avg}}(l) = \frac{\beta(l-\gamma)}{l} \tag{4.14}$$

The angle to the customer, θ, determines the difference in speed between the two motors. See equation (4.15). The scaling variable α is is used to tune the magnitude of the difference.

$$\Delta V(\theta) = \alpha \cdot \theta \tag{4.15}$$

V_{avg} and ΔV are then combined in to the setpoints for the left and the right motor. See equation (4.16) and (4.17).

$$V_{\text{L}} = V_{\text{avg}} + \Delta V \tag{4.16}$$

$$V_{\text{R}} = V_{\text{avg}} - \Delta V \tag{4.17}$$

In figure 4.11 below, the complete process is illustrated.

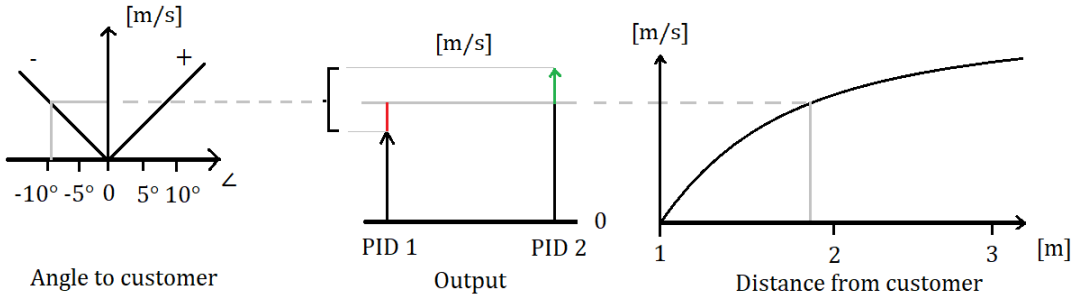


Figure 4.11: Illustration of how the motor speed is set depending on the distance and angle to the customer.

5 Collision Avoidance

With working systems for triangulation of the CU, the prototype cart is still essentially blind, meaning that any objects in its path will cause a collision. To make sure that this does not happen, a system that can detect obstacles and avoid collision is needed. However, there are two variants of this.

The first variant is passive, where avoiding a collision means that the cart can detect obstacles in its path. If the cart stops as soon as an obstacle is found, the collision has effectively been avoided.

The second variant is active. Here, the system uses the obstacle data from the first variant to guide the cart around the obstacle. In this chapter solutions for both variants will be explored, therefore CA is divided into two parts; 5.3 obstacle detection and 5.4 active collision avoidance (ACA).

5.1 Related Work

There are several methods for detecting obstacles in real-time, many of the methods found were based on LIDAR[18], [19], [20]. The technologies that uses lasers to measure distance based on the time it takes for the laser to reflect back and hit the sensor. Doing this many times a second in different directions and storing the data, a 3D-depth map can be created of the surrounding area.

Another method uses the Microsofts Kinect [21] which has built in cameras, an IR transmitter, and an IR depth sensor. A video output with corresponding depth information can be captured. The depth sensor works by having the IR sensor transmitting many individual IR dots in the room. These dots are then captured with two sets of IR cameras with a set distance between each other. A computer later uses triangulation on each dot to determine the distance of each dot. However, the relatively high price of these components and its complexity makes these solutions unsuitable for our project.

A third option is to use cameras. There are many advanced methods to detect obstacles using a camera [22], [23], [24], [25], however the methods found were developed for more complex applications and environments than a grocery store. Because of this, rather than trying to implement one of these complex methods, a simpler method will be developed.

5.2 System Requirements

The CA-systems requirement is to detect any obstacles in the path of the cart and avoid a collision with them. Since the cart is mobile, the system should be able to process the data in real-time, with a refresh rate of at least 10 frames per second. This rate is considered enough for the system to respond to suddenly appearing obstacles.

In order to detect all relevant obstacles there are requirements for the camera placement. It must be able to see at least 40 cm in front of the cart, and the view must at least be as wide as the cart.

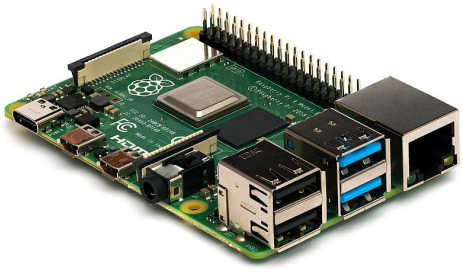
5.3 Obstacle Detection

For the system to avoid a collision it first needs detect obstacles, and since the cart is only intended to move forward, the area of interest is in front of the cart. By mounting a camera on top of the cart, facing down, this area can be captured. The data captured by the camera can be streamed to a computer which processes the images. The processed images are then analysed and a decision can be made as to whether any obstacles were found. These three basic steps for obstacle detection are represented in figure 5.1.

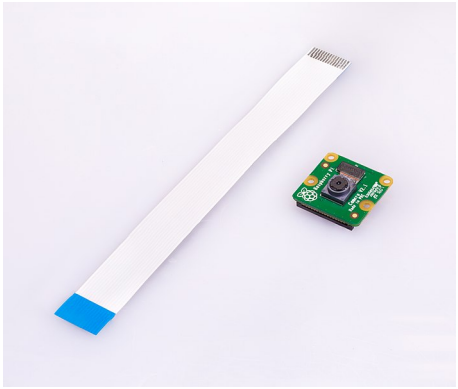
Figure 5.1: Basic flow for obstacle detection.

5.3.1 Hardware

Three components are needed for this system. The two main components used for the steps shown in figure 5.1 is a camera module for capturing the images, and a computer to process and analyse them. The computer used is a Raspberry Pi 4 Model B 2GB, and the camera a Raspberry Pi Camera V2. The third component is the Arduino Mega, which is the microcontroller the collision data is sent to.



(a) Raspberry Pi 4 Model B - Side, from [26], Michael Henzler / Wikimedia Commons / CC BY-SA 4.0.



(b) Raspberry Pi Camera Module V2, from [27], CC BY-SA 4.0.

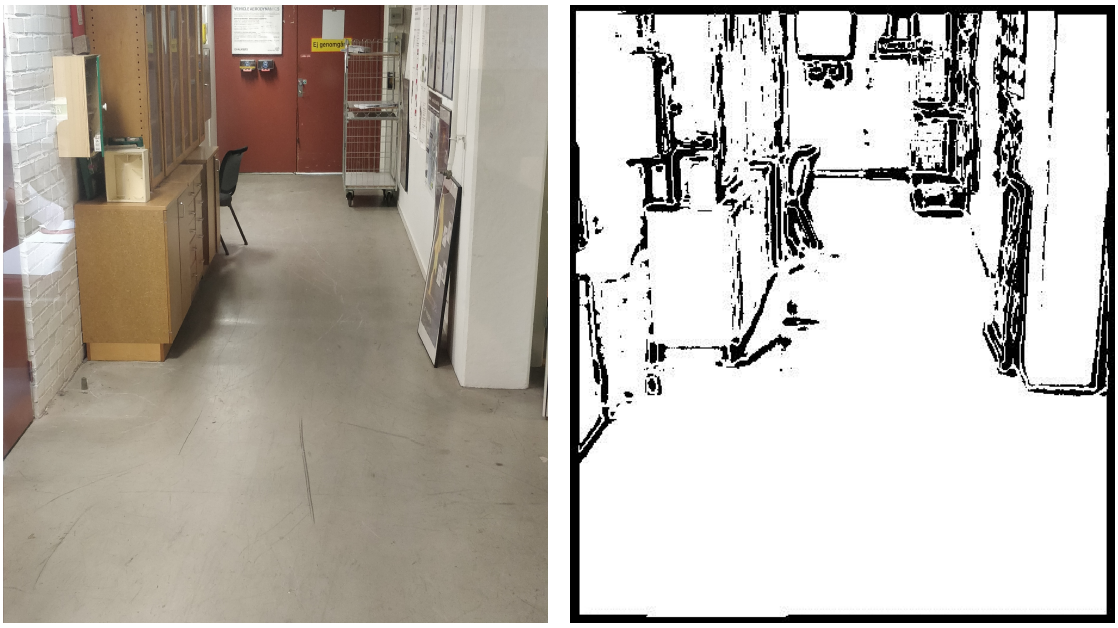
Figure 5.2: Main hardware components.

5.3.2 First Iteration for Obstacle Detection

Before receiving any of the components, a self-made algorithm was constructed in Java. However, when this algorithm was implemented on the Raspberry Pi, processing one frame took several seconds. Since performance would have had to increase by several orders of magnitude to reach the system requirements, it was decided to start over, rather than try to optimise this code.

Even though the algorithm was deemed to have insufficient performance for real-time processing, the algorithm described below, still produced the expected result, as can be seen in figure 5.3 below.

1. Downsizing the image to improve performance.
2. Making the image grey scale of the original by averaging the colour data.
3. Blurring the image to get rid of textures and small irregularities in flooring.
4. Finding contours by checking if the grey-scale data of the single pixel being examined differs from average grey-scale data of the chosen surrounding pixels over a set value. Also if either the surrounding pixels or the pixel being examined is excessively bright the contour is perceived as glare and ignored.
5. Return the altered image.



(a) An example of a image used to test the java obstacle detection method.

(b) The resulting image using the java obstacle detection method where the black pixels represents obstacles.

Figure 5.3: Test of Java image processing algorithm.

5.3.3 Second Iteration for Obstacle Detection

As the results in figures 5.3a and 5.3b show, the method works, however the performance was deemed too bad to continue by improving it. To solve the performance problem, instead, research into more efficient methods that produced the same, or similar, results were made. During that time the OpenCV library was discovered. OpenCV stands for Open Source Computer Vision, and is a library of open source programming functions for real-time computer vision applications. One function in this library, is called Canny Edge Detection. Canny edge is an algorithm that detects edges of objects in an image. Initial testing of the canny edge algorithm showed that it could produce similar results to those in figure 5.3b, but with much greater performance.

The Canny edge method is a multi-step algorithm, the steps are illustrated in figure 5.4 below. Each of the steps are explained in detail in the OpenCV documentation[28].

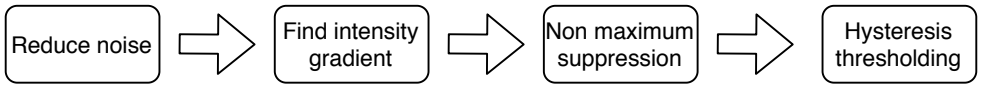


Figure 5.4: The canny edge method.

In the list below, a brief summary of the steps in the Canny edge documentation can be seen[28].

1. Reduce Noise
The image is smoothed by using a 5x5 Gaussian blur filter. This step removes textures and unwanted noise in the image.

2. Gradient Calculation
This step calculates the gradient of the image in order to detect the edge intensity and direction. The magnitude of the gradient gives the edge intensity, and the angle gives the edge direction. The result is a picture with only edges. However, these edges are not consistent, meaning the edge thickness varies.

3. Non-maximum Suppression
After the edges has been found with step 2, this step refines the edges. This is done by checking every pixel in the image. If the pixel is not a local maximum in the direction of the gradient, it is removed. The result is similar to second step, but all the edges are now thinner.

4. Hysteresis Thresholding
In this step, two threshold values, minVal and maxVal, decides which edges which edges are considered to be real edges. The edges intensity gradient is compared to the minVal and maxVal thresholds. If it is above maxVal, it is considered to be a real edge and kept as such. If it is below the minVal threshold, it is not, and

thus removed. Edges which are neither above maxVal or below minVal are kept depending on whether they are connected to a real edge or not. See figure 5.5 below.

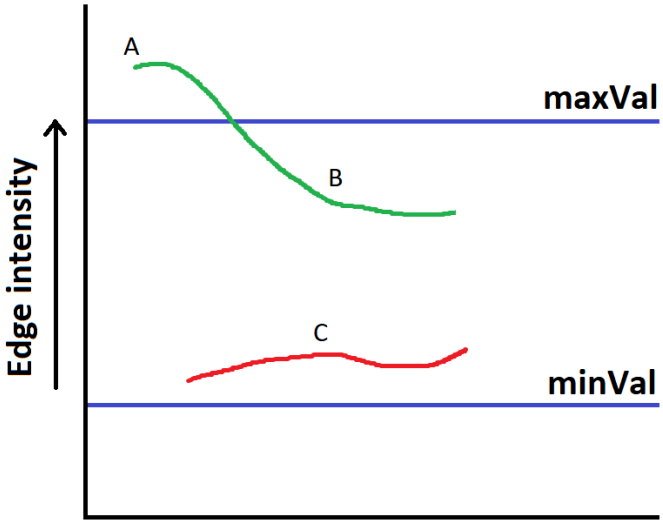
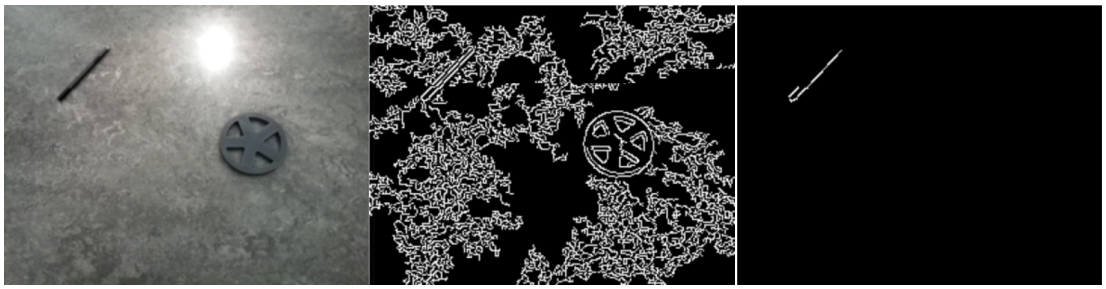


Figure 5.5: Canny edge thresholding, A and B are kept, C is discarded.

Edge A has a higher intensity than the maxVal so it is considered as a real edge. Edge B is below the maxVal, but since it is connected to A, it is also considered as a valid edge. Edge C is in the middle, even though it is above the minVal, it does not have a connection to a real edge, and is therefore removed.

In the OpenCV library all of these steps are done by a single function, cv.Canny(). The function has three inputs. In first one, the Canny edge algorithm is applied to the image. The second and third argument are for the minVal and maxVal values. The challenge with these variables are to calibrate them to remove any false positives from glare or textures in the floor while making sure that objects are not filtered out by mistake. Initially, different combinations of these thresholds were tested in order produce the desired result, shown in figure 5.6.

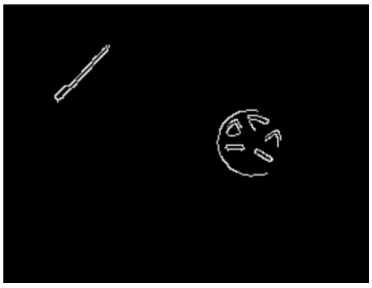


(a) Original image. (b) Canny edge with false positives. (c) Canny edge object missed.

Figure 5.6: Canny edge with bad threshold values.

However, when testing the system in different lighting conditions, see figure 5.7, the

parameters once again had to be tweaked. Because of this, a way to automatically configure the parameters based on image brightness was necessary.



(a) Brighter environment.

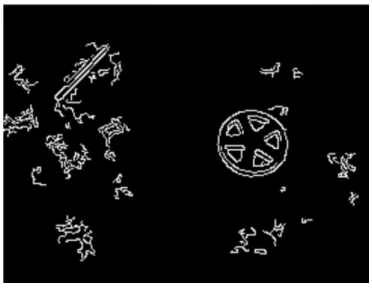
(b) Darker environment.

Figure 5.7: Same threshold values, but different lighting.

On programcreek.com[29], a site with many open-source code examples, author *dockerizeme* defines a method for automatically setting the thresholds. This works by calculating the median of the image, using the median, lower and upper threshold are calculated. The size of these thresholds are weighted by the variable σ, a higher value of σ increases the span. This can be seen in equations (5.1) and (5.2) below.

$$\text{minVal} = (1 - \sigma) \cdot \text{median} \tag{5.1}$$

$$\text{maxVal} = (1 + \sigma) \cdot \text{median} \tag{5.2}$$



(a) Brighter environment.

(b) Darker environment.

Figure 5.8: Canny thresholds set by the auto-calibration function, $\sigma = 0.33$.

While this made the results somewhat consistent under differing light conditions, as shown in figure 5.8, too much noise was still picked up. Experimenting with different values of σ only slight reduced the amount of noise picked up. By adding a manual calibration value to minVal and maxVal the noise could be eliminated, while still keeping the results somewhat consistent, as can be seen in figure 5.9.

$$\text{minVal} = (1 - \sigma) \cdot \text{median} + \text{minCalibration} \tag{5.3}$$

$$\text{maxVal} = (1 + \sigma) \cdot \text{median} + \text{maxCalibration} \tag{5.4}$$

(a) Brighter environment. (b) Darker environment.

Figure 5.9: Auto-calibration function with manual adjustments.

After the image has been processed with the Canny edge method, any objects within the range are detected with an algorithm. The algorithm is quite simple, if any white pixels are detected within the Canny edge image it is considered as a potential collision, see algorithm 9.

Algorithm 9: Pseudocode to detect potential collisions.

```
bool detectCollision(cannyImage):
    for(every pixel in cannyImage):
        if(pixel is white):
            return True
    return false
```

However, while most noise in the Canny image are eliminated by the threshold values, sometimes random noise still appear over one or two frames. The potential collision therefore needs to persist over at least three frames before it is considered an actual collision. This is done to minimize the effects of the random noise. Finally, when the collision has been confirmed, a signal to stop is sent to the arduino via USB. This can be seen in algorithm 10 below.

Algorithm 10: Pseudocode to confirm any collisions.

```
for(every frame in cameraStream):
    cannyImage=cv2.canny(frame,minVal, maxVal)
    blocked=detectCollision(cannyImage)
    if(blocked):
        collision++
    else:
        collision=0
    if(collision > 2):
        send stop signal to Arduino
```

5.4 Active Collision Avoidance (ACA)

The solution using Canny edge enables the prototype cart to detect any objects in its path, and avoid collision with them by stopping. However, this solution is not ideal, since it requires the object to move out of the carts path in order for the system to start again. This means that any static objects in the path, will require manual intervention, else the cart will be stuck.

In order to not only stop at the sight of an obstacle and instead avoid it in the first place, another steering algorithm is required between the output from the follow the customer subsystem and and the prototype. While a number of existing technologies exists, as previously mentioned in section 5.1, they were deemed to complex and compute-intensive to be implemented as a subsystem. Due to this, and the uncertainty of acquiring other sensors during the covid-19 pandemic, a simple and efficient method using the existing hardware was required. As no such methods were found, a new, custom one needed to be developed.

This custom method will use the data from the Raspberry Pi, and the data from the follow the customer subsystem, in order to determine the optimal path for the prototype while actively avoiding obstacles along the way. This is done on the Arduino Mega for simplicity as this only requires a one-way communication from the Raspberry Pi and the Arduino.

To limit the amount of data that needs to be processed in the avoidance system, the images are processed by the canny edge algorithm and then further simplified into a grid. The algorithm that converts the image to a grid first creates the grid by taking the resolution of the original image and down-scaling it by a predetermined factor. This grid is then filled by dividing the original image into grids of the same size. Each grid is then scanned for a white pixel, if one is found, that whole grid is deemed to be blocked and the corresponding pixel in the grid image is set to white. This repeats for the whole image until the grid is filled. See algorithm 11 for pseudocode, and figure 5.10 for results.

Algorithm 11: Pseudocode to convert canny edge image to grid.

```
convertToGrid(cannyImage, gridImage):
    divide cannyImage into a grid of same size as gridImage
    scan each grid for a white pixel:
        if a white pixel is found in a grid:
            set corresponding grid in gridImage to white and move on to the
              next grid
    repeat until the gridImage is filled
```

(a) Canny edge image. (b) Grid image.

Figure 5.10: Conversion of Conny edge image to grid data.

5.4.1 Performance and Optimisation

As figure 5.10 shows, algorithm 11 produces the desired results, however, real-time performance of this algorithm was problematic. Even with a very low resolution, 272x208, the system could only process one to two images per second. Research into why the algorithm slowed down the system indicated that the language used was the reason. Python adds a large amount of overhead when running numerical loops [30], which is why looping over images dramatically decreases performance. In order to fix this problem, a python library named Cython was used. Cython solves the problem by compiling python code directly to C[30]. Because it uses C types, Cython makes it possible to embed numerical loops, running at C speed, directly in Python code [30].

> *Cython is a Python language extension that allows explicit type declarations and is compiled directly to C. As such, it addresses Python's large overhead for numerical loops and the difficulty of efficiently using existing C and Fortran code, which Cython can interact with natively* [30].

Adrian Rosebrocks tutorial [31] about optimising for-loops with Cython goes into detail about how to apply the Cython library to existing python functions. Using the same technique, the functions in the obstacle detection program could be optimised. The performance gained from this was significant, the program went from processing one to two images per second to over 100 images per second, see figure 5.11.

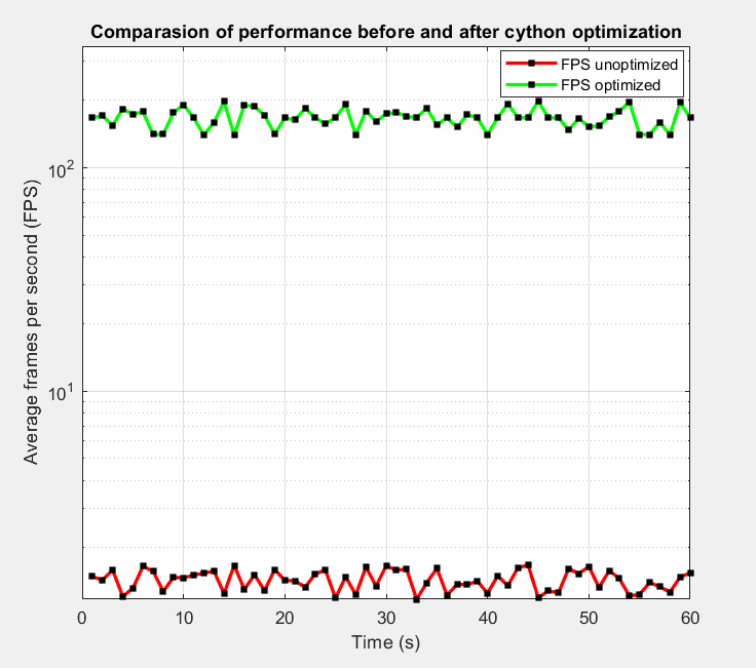


Figure 5.11: Performance of the grid conversion function, before and after optimisation with Cython.

5.4.2 Communication

Before the image grid data can be processed on the Arduino, it first needs to be sent to it. This is done in a unchecked fashion, meaning that the raspberry sends all the image data without knowing if the data has been received or not. Each image sent is preceded by a predetermined start byte and concluded with a end byte. On the receiving end, the Arduino checks for the start byte. Once the start byte is received, the image data is stored in a array on the Arduino. If it then receives the correct end byte, the image is deemed to be uncorrupted and is consequently processed. If the end byte does not match with what was predetermined, the data is deemed corrupt and is discarded.

5.4.3 Method

While a number of methods were considered for navigation, it was quickly concluded that any complex path-finding-algorithm would take to much processing time. The algorithm would have to run every frame due to the constantly varying in-data from the camera. Due to this limitation in computing power in the Arduino, a simpler method was created, as there were no existing methods for solving the problem. The core method consist of a position vector based cost optimisation that runs every frame, where the vector points to where the system wants to end up in the new frame. The vectors length is based on the speed of the system while the angle chosen after what gives the lowest cost possible calculated from three different costs. The three costs are:

1. Cost of erratic movements: Calculated based on a scalar of how severe the new angle is.
2. Cost of new proximity to closest obstacle: Calculated based on the proximity of the new position of the system to the closest obstacle if the angle in question was chosen.
3. Cost of swerving from the FTC path: Calculated based on the difference between the new tested angle and the angle to the customer.

These three costs were chosen to give the system the desired behaviour.

The figure 5.12 below is a visual representation of how this method works, where the theoretical position of the customer is represented by a green circle. It constantly evaluates a set number of vectors that describe the system's possible future movements. The number of vectors can be altered to aid performance. The length of the vectors is represented by the letter r in figure 5.12 and deepens on the current speed of the prototype. It then evaluates the costs for every vector, every frame to determine which vector has the lowest cost and uses the angle of the lowest-cost vector as the output. This is the new direction where the cart should go, and is illustrated by the green vector in figure 5.12.

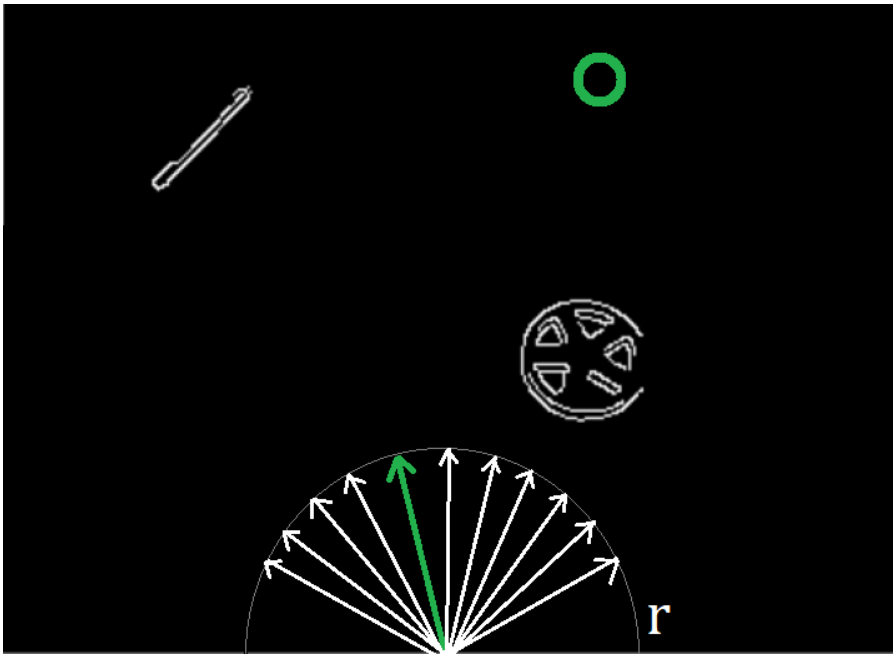


Figure 5.12: Visual representation of of the ACA-method.

The number 1 and 3 costs can easily be calculated using the angle of the currently tested vector, and the angle to the customer. The second cost 2 requires further calculations.

5.4.4 Mapping

In order to use this method effectively, there are two main problems that needs to be solved. The first one being determining the exact coordinate in 2-d space of any select pixel, and the second one being determining which obstacle is the closest. In order to do this some way of mapping pixels to coordinates is needed. As the camera is static, every pixel always corresponds to the same position relative to the cart. Using this information and some data, such as the field of view of the camera, camera angle, and the resolution of image it is possible using trigonometry to map every pixel relative to the system. Figure 5.13, 5.14, and 5.15 serve to illustrate the angles and distances involved in mapping X and Z coordinates to pixels from the camera. 5.13 is a 3D representation, 5.14 is from the top and 5.15 from the side.

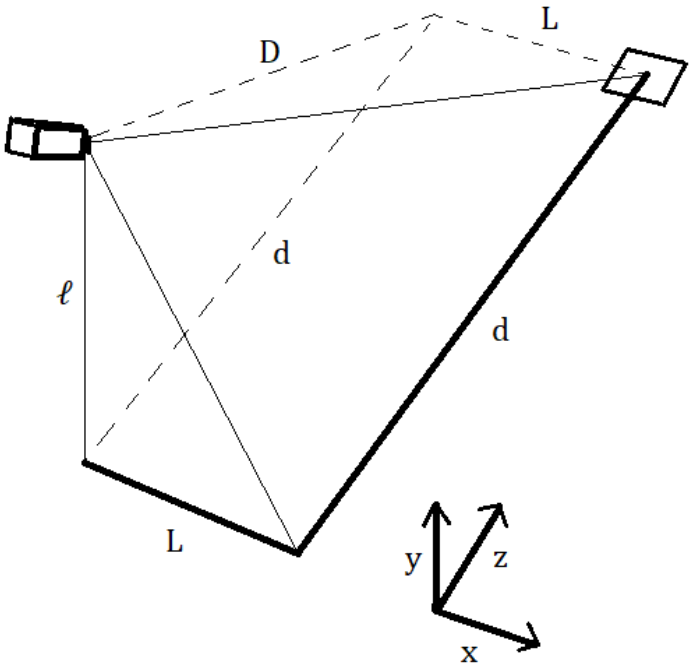


Figure 5.13: Illustrative picture of the distances involved in mapping where the letter L represents the X value and the letter d represents the Z value of the pixel being mapped.

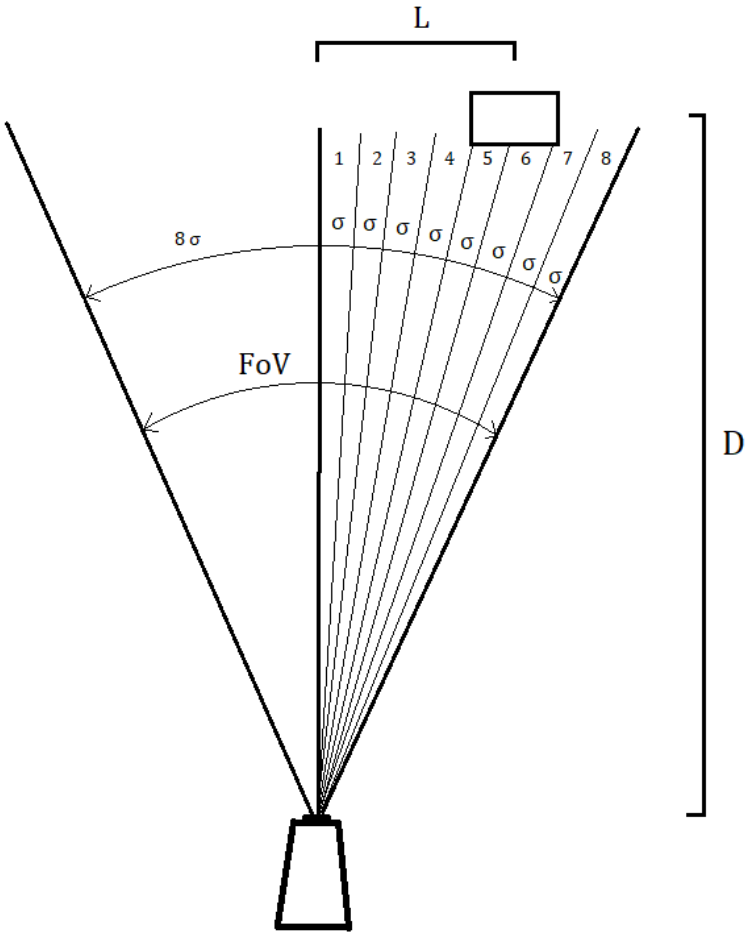


Figure 5.14: Illustrative picture of the angles involved in mapping obstacles viewed from top down where FoV refers to the horizontal field of view seen from this plane (also referred to as $FoV_{horizontal}$), and σ represents the angle between every pixel from the camera in this plane.

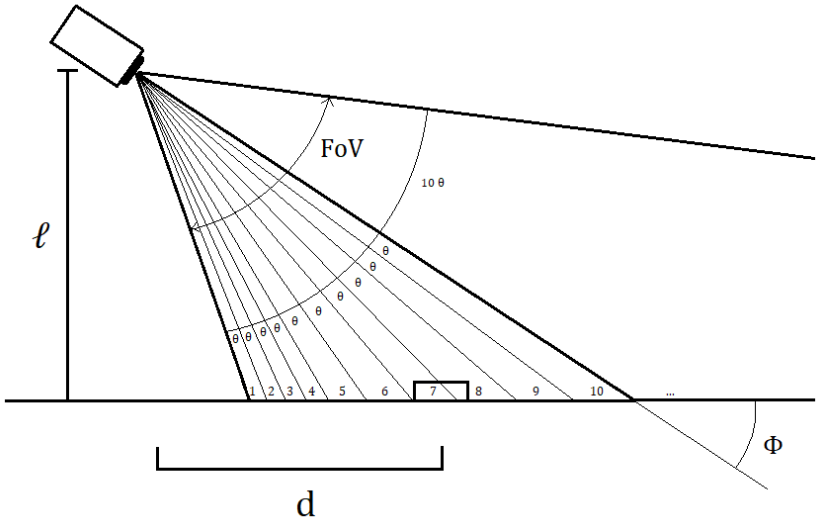


Figure 5.15: Illustrative picture of the angles involved in mapping obstacles viewed from side on where ϕ is angle of the camera, FoV refers to the vertical field of view seen from this plane (also referred to as $FoV_{vertical}$), and θ represents the angle between every pixel from the camera in this plane.

The angle θ, seen in figure 5.15, is calculated by dividing the vertical field of view FoV_{vertical} by the number of vertical pixels n, as seen in equation (5.5).

$$\theta = \frac{FoV_{\text{vertical}}}{n} \tag{5.5}$$

The angle σ, seen in figure 5.14, is calculated by dividing the horizontal field of view FoV_{width} by the number of horizontal pixels m, as seen in equation (5.6)

$$\sigma = \frac{FoV_{\text{width}}}{m} \tag{5.6}$$

In order to calculate the distance d as seen in figure 5.13 and figure 5.15 which represents the distance from the cart to the obstacle in the Z direction the angle from the camera to the pixel being examined seen from side on (here represented by the letter ϵ) is required. This can be determined by the following equation (5.7) where the letter i represents the Y value of the pixel being examined.

$$\epsilon = \phi + FoV_{vertical} - (n - 1 - i) \cdot \theta \tag{5.7}$$

The distance d can now be describes by the equation (5.8) below.

$$d = \frac{\ell}{\tan(\epsilon)} \tag{5.8}$$

In order to calculate L which represents the distance from the cart to the obstacle in the X direction, the distance D must first be calculated. As the previous equations

gave a value for the distance d, and the height of the camera (ℓ) is known, the distance D can be calculated using Pythagoras theorem in equation (5.9) below.

$$D = \sqrt{\ell^2 + d^2} \tag{5.9}$$

Much like with ϵ the angle from the camera to the obstacle from the top down view seen in figure 5.14 is needed in order to calculate the wanted distance L. This is done in the equation below (5.10) where j represents the X value of the pixel being examined.

$$\zeta = \sigma \cdot \left(j - \frac{m}{2}\right) \tag{5.10}$$

The distance L can now be describes by the equation (5.11) below.

$$L = D \cdot \tan(\zeta) \tag{5.11}$$

With these equations it is possible to map every pixel (and therefore obstacle) to a point in 2-d space given the correct input parameters which all are static in this system. This is done in the the setup-phase for every pixel from the grid data and stored in an position map for easier and faster computation.

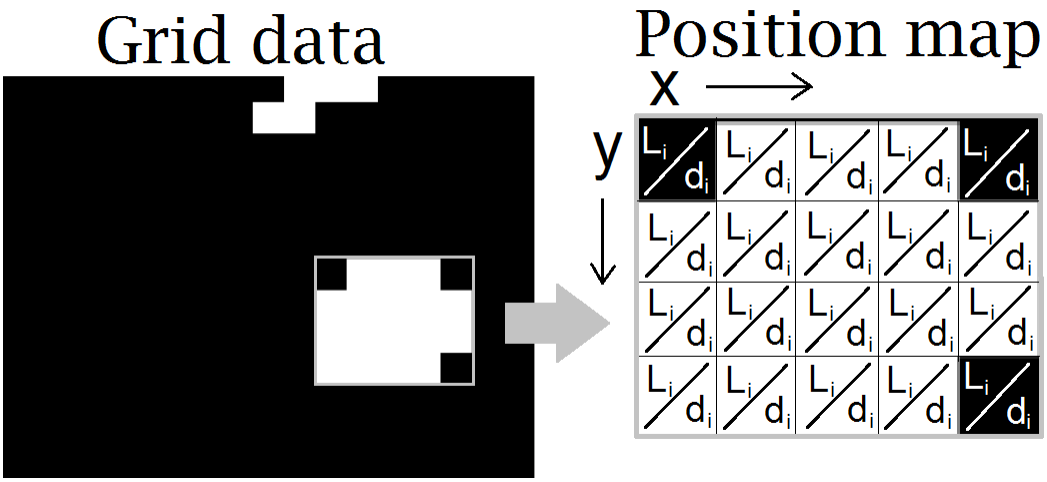


Figure 5.16: Mapping of pixel positions.

In order to easily detect which obstacle is closest the system always checks the closest pixel in succession. In order to do this a sorting algorithm was used to create a list of positions with distances in acceding order. Given a start-position this method creates a list with the index of the pixels which represent the lowest distance to the start-position in acceding order.

1. If it is the first time, set current position to first position in the list.
2. For every pixel, see if there are positions in the list with a higher distance to the start position than the position being tested. If there is, count how many.
3. Shift every position in the current list that has a longer distance to their position +1 starting with the last position in the list.
4. After that is done, write the current position in the correct place.
5. When every pixel has been sorted, return the list.

6 Prototype Design Process

In this chapter, the design process of the prototype will be presented. The process of designing the prototype cart was to draw the complete system as a CAD assembly. Having an overview of the assembly in CAD gave a sense of scale between components, and component designs could be adjusted if necessary. Also various dimensions could be verified, making sure different components did not collide with each other.

The workflow was in a linear fashion, meaning that the components were added one by one, eventually forming the complete assembly, that can be seen in figure 6.1 below.

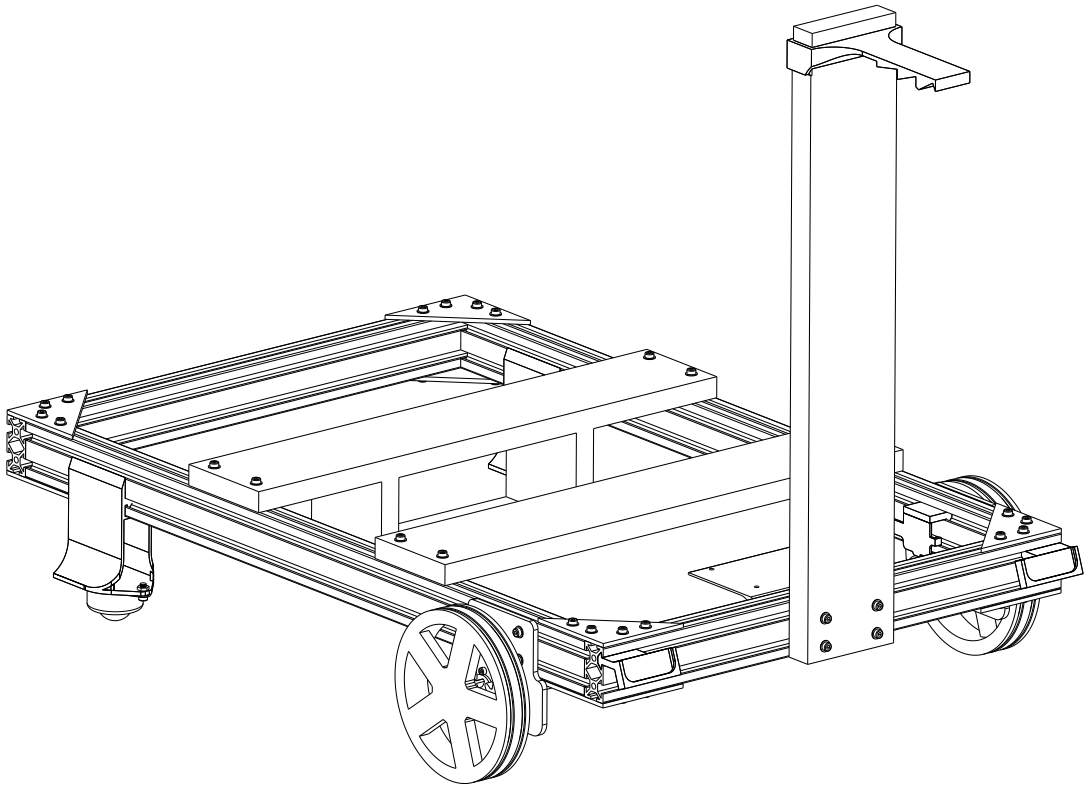


Figure 6.1: Complete CAD assembly.

To physically connect all the components together into a complete unit, while maintaining the desired modularity, an extruded aluminium framework was proposed as a solution, as will be described in the following section.

6.1 Extruded Aluminium Framework

Extruded aluminium struts with slots for custom mounting equipment were available in many different varieties. Struts were appealing as it can be utilised when fitting

components together, and in the end constitute to a modular design. Outlines for an aluminium strut can be seen in figure 6.2 below.

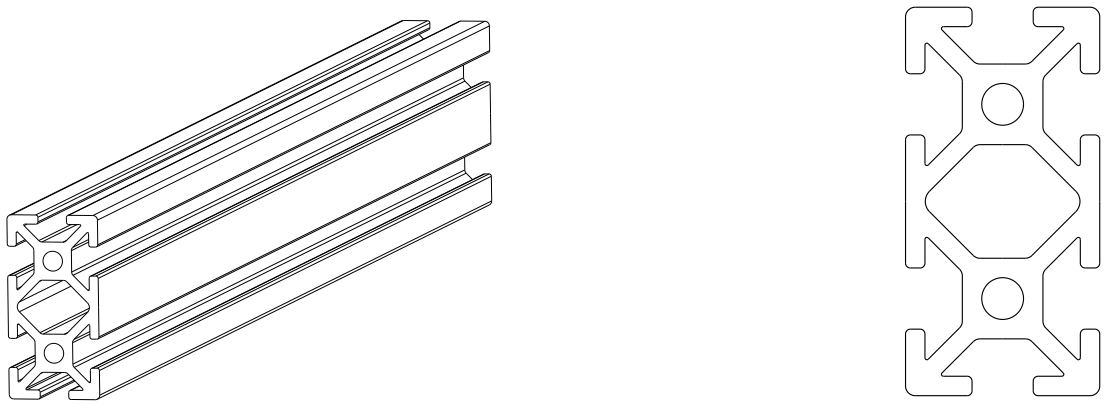


Figure 6.2: Aluminium strut outlines. The left is a side view and the right one is the profile.

A rectangle profile outline was chosen as it will add more material vertically for better handling vertical loads, and also facilitate in making perpendicular connections in the framework.

The first idea for connecting the aluminium struts to each other, was to drill holes through them, use a tap tool to create threads, and fix the framework in place using bolts. This idea was abandoned because the holes had to be drilled with great accuracy to make sure that all the corners of the framework would be perpendicular. Having only two screws for each connection also results in rather weak screw joints that could potentially break during heavy loads.

Instead it was decided to utilise the available slots in the struts. The idea was to fix the corners with plates bolted into nuts that are placed in the slots. For implementing and testing this idea, aluminium sheet was cut with a sheet metal cutter into a triangle shape. Then holes were drilled using a drill press.

The resulting triangle attachments were satisfying, and enabled a simple way of connecting the corners and locking the entire frame in a rectangular shape. The corner connection is shown in figure 6.3 below. The triangle plates can be bolted onto the struts with nuts placed in the strut slots, as shown in figure 6.4.

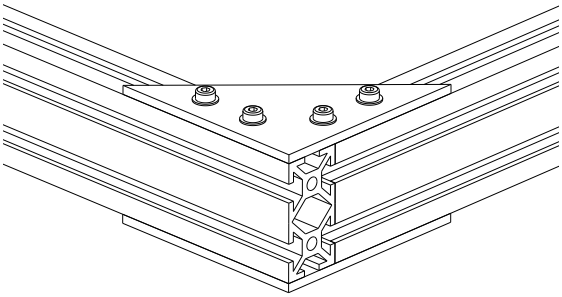


Figure 6.3: Two struts connected with the corner plates.

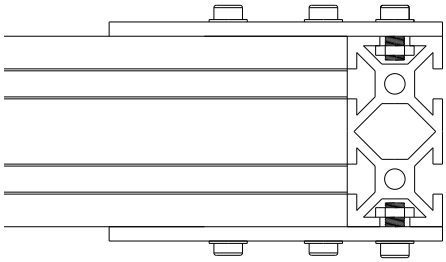


Figure 6.4: Connection from the side showing the nut placement in the slots.

6.2 Motor and Encoder Bracket

With the framework design complete, the next step was to attach the motors and encoders to it. The motor and encoder axles must be connected with a transmission.

A simple way of doing this is to connect the axle tops with an adaptor of some sort. However, this was not possible as it was planned to mount the driving wheels directly to the motor axle.

Instead the axles were placed parallel to each other with some space in between, making room for gears. For this arrangement, a bracket with outlines matching the motor fixing holes and the motor flange was drawn. This can be seen in figures 6.5 below.

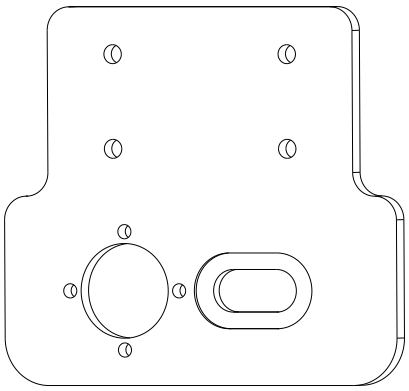


Figure 6.5: Motor and encoder bracket outline.

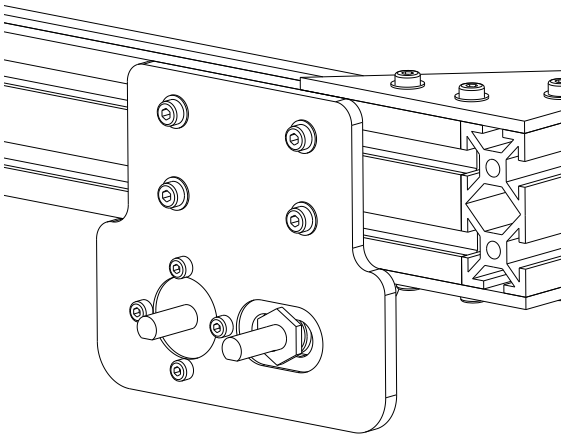


Figure 6.6: Bracket bolted onto the frame with motor and encoder axles visible.

For bolting the bracket to the frame, as seen in 6.6 above, hole outlines matching the strut slots were drawn. As the transmission gears were yet to be designed, a precise centre point for an encoder hole on the bracket was not known. Therefore

a groove outline was drawn instead, enabling lateral adjustments later on. The groove can be seen in figure 6.5 above.

The intention was to manufacture a pair of brackets out of steel or aluminium at the Chalmers prototype lab, similar to how the corner plates were made. However, because of the covid-19 outbreak, the prototype lab closed, which complicated the manufacturing process. For solving this circumstantial problem, the brackets were 3D-printed in plastic instead. Consequently, its durability was reduced as plastic material is weaker compared with a metal equivalent. However, 3D-printing the brackets saved some time that would otherwise be spent on manufacturing.

6.3 Encoder Gear

As mentioned in the previous some kind of transmission is required to enable a rotational motion between the encoder and the wheel. As the force transfer is rather small due to relatively low friction in the encoder, it is possible to 3D-print this part in a plastic material. The small torque demand also means that there is no specific gear ratio requirement. However, since the encoder has a maximal RPM of 3000, a requirement is to not exceed this. The RS Pro motors have a maximum RPM of 366. With a gear ratio of 1:1 the limit will not be exceeded.

When designing the cogs for the encoder gear, the first approach was to use a gear generator software. However, a problem with this was combining the output files from the software with the CAD models. Instead a custom cogwheel outline was drawn, and can be seen in figure 6.7 below.

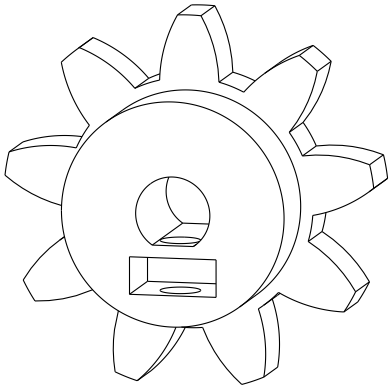


Figure 6.7: Encoder gear outline.

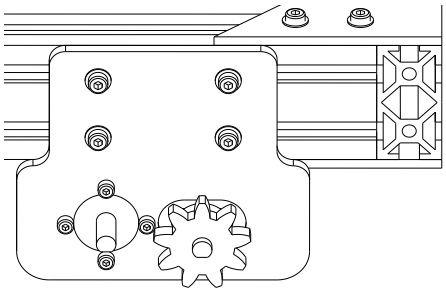


Figure 6.8: Encoder gear on encoder axle.

To ensure a good fit on the encoder axle a transverse hole was drawn, enabling a set screw to lock against the flat axle. As the 3D-printed plastic is to soft to withstand any significant torque the set screw produces, a steel nut containing strong threads can be placed in a slot to support this force. This gear could simply be mounted onto the encoder axle, as can be seen in figure 6.8 above.

6.4 Driving Wheels

Initially, it was discussed whether the driving wheels should be purchased or 3D-printed. The main advantage of buying wheels is that they yield greater overall quality. However it was more costly, and would have limited the flexibility of the build due to its specific dimensions. Also, since the wheels need to be connected to the encoders via the encoder gear, a custom design is more advantageous, as the encoder gear design could be reused and incorporated into the wheel design. Hence, a 3D-print model of the driving wheels were designed in CAD, and this model can be seen in figure 6.9 below.

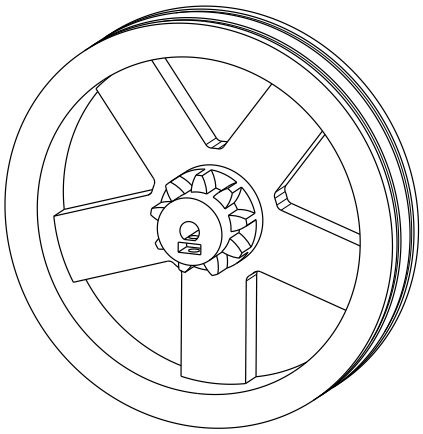


Figure 6.9: Driving wheel outline.

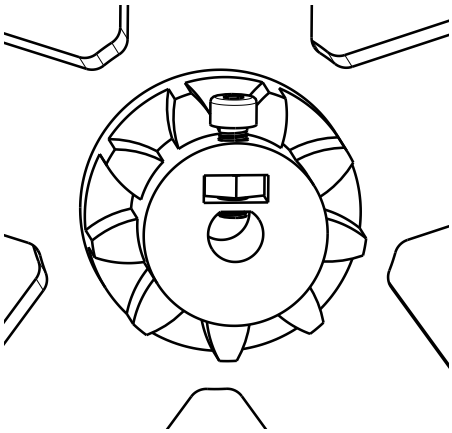


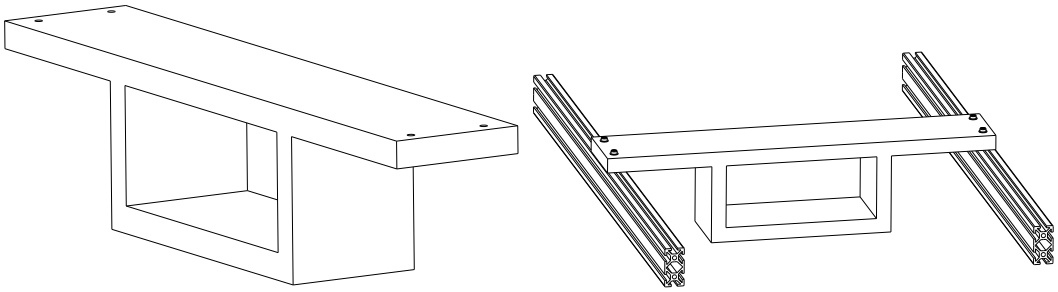
Figure 6.10: Wheel hub with incorporated gear and locking mechanism.

The wheels consists of five spokes, and the gear wheel is incorporated into the wheel hub. Also, there are slots for rubber bands on the outer edges, enabling a way of gaining greater friction. In order to mount the wheel to the motor, there is another slot in the wheel hub that can hold a locking nut, similar to the encoder gear design. The wheel can be fixed on the motor axle with a set screw, as can be seen in figure 6.10 above.

6.5 Battery Holder

For more accurate steering, it is important that the battery is mounted somewhat centrally, as it is quite heavy. Also, to be able to drive forward, frictional force between the wheel and the the ground is necessary. To ensure enough load on the front wheels, the battery was positioned in the middle of the cart.

The design of the battery holder is rather simple. A transverse wooden board was secured to the frame with bolts and nuts. The battery can then be mounted on the bottom side of the board and is held up by two vertical smaller boards, as can be seen in figure 6.11 below.

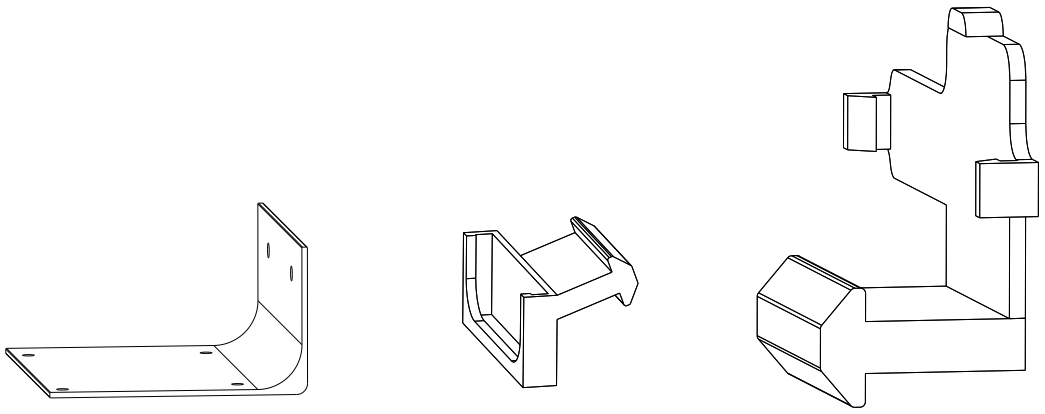


(a) (b)

Figure 6.11: Battery holder and how it is mounted on the frame.

6.6 3D-Printed Attachments

Having the greater components in place, the smaller electronic components needed to be attached. This included the Arduino, the USS:s and the motor controllers. As the electronics are lightweight and relatively small, attachments was custom designed with 3D-printing in mind. As the 3D-prints are plastic, it ensures the printed circuit boards solder points are not short circuited via the abutting material. The attachments outlines can be seen in figure 6.12 below.



(a) (b) (c)

Figure 6.12: 3D-printed frame attachments. (a): L-shaped bracket for the Arduino. (b): USS holder. (c): Motor controller holder.

In figure 6.13 below, a drawing of how the three 3D-printed attachments are mounted to the frame is shown.

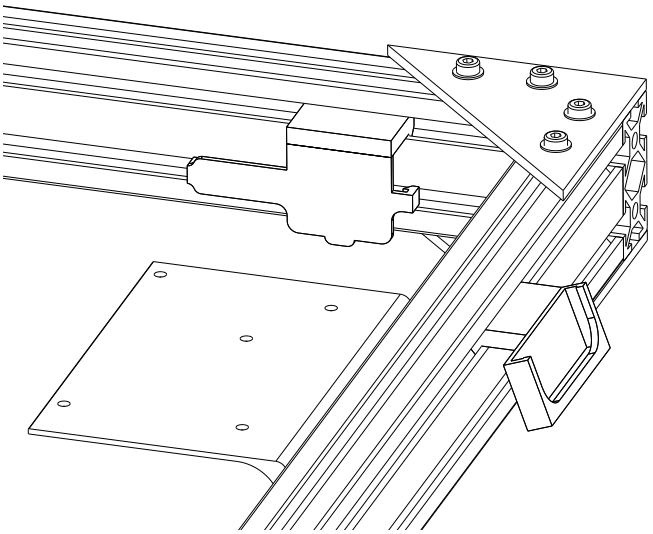
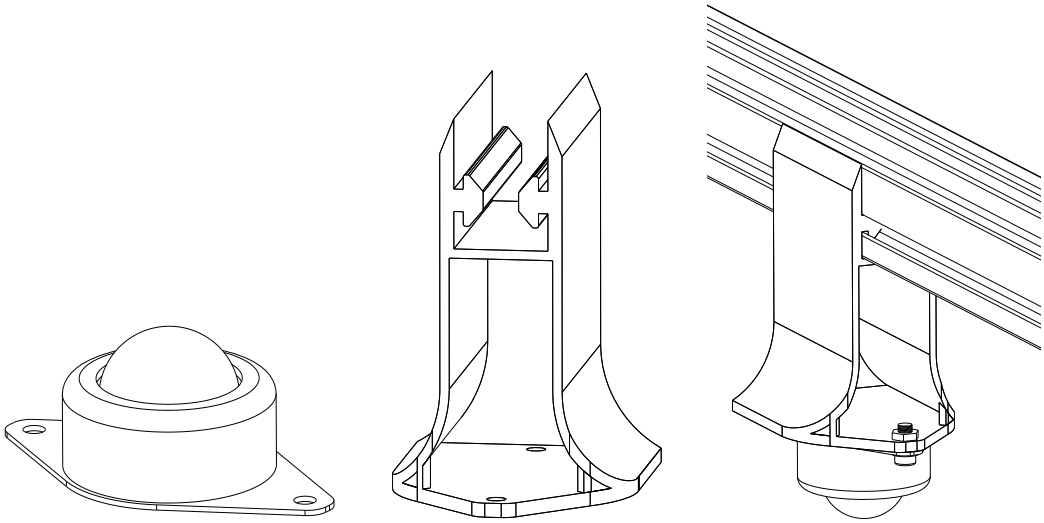


Figure 6.13: Illustration of 3d printed attachments on the frame.

The USS attachment and the motor controller attachment have custom made sliding connectors specially made to fit the strut slots, as could be seen figure 6.12b and 6.12c. However, for the L-shaped Arduino bracket, there was room for bolt holes instead, as it was a bigger component Attaching this with bolts and a locking nut saved printing time and printing material.

Another component that needed an attachment was the rear wheel. For the rear wheels, common shopping cart casters was first considered. However, a specific type of castor, called ball castors, was discovered and an outline of this castor can be seen in figure 6.14a below.

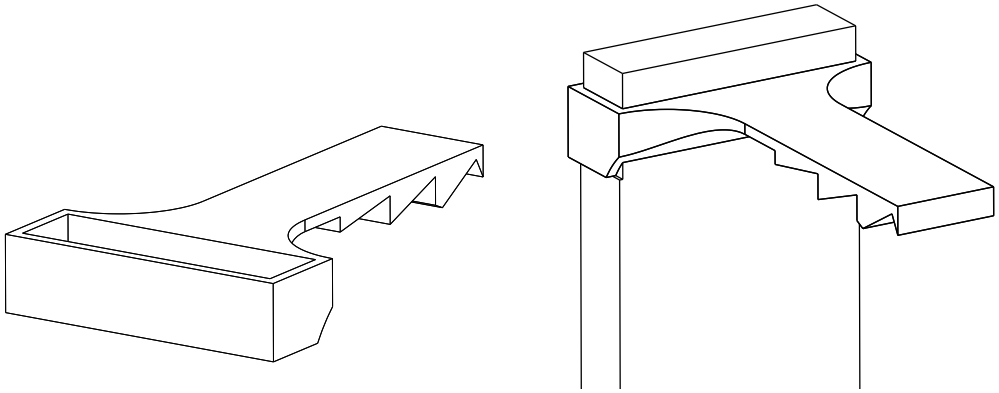


(a) (b) (c)

Figure 6.14: Ball castor attachment.

The ball castors were too short to bolt directly on the frame, which meant that extension attachments were needed. In figure 6.14b above an extension that can be inserted into the strut slots can be seen. The complete rear wheel module, attached to a strut, can be seen in figure 6.14c above.

The last main component that needed an attachment was the camera. As mentioned in chapter 5, the camera should preferably be elevated above the frame to provide vision in front of the cart. In order to elevate the camera, a 450x75 mm wooden board was attached to the front. Further, the camera attachment itself was designed with a rectangular slot for inserting the wooden board. For enabling different camera angle choices, there is three angular settings at the end of the attachment. The outline of the attachment can be seen in figure 6.15a below, and in figure 6.15b the attachment is mounted on the board.



(a) (b)

Figure 6.15: Camera attachment with three different camera positions.

7 Assembly & Software Integration

This chapter will show and describe how the components were assembled into a physical prototype and how the CA and FTC systems were integrated to it.

7.1 Assembly of Hardware Components

This subsection will describe how the prototype was built from the individual components in a number of steps. Pictures will show the process.

First the motor controllers was adhered to the holders with double-sided tape, and connectors were added to the terminals. The ball castors were bolted to the rear wheel extensions. These can be seen in figures 7.1 and 7.2 below.

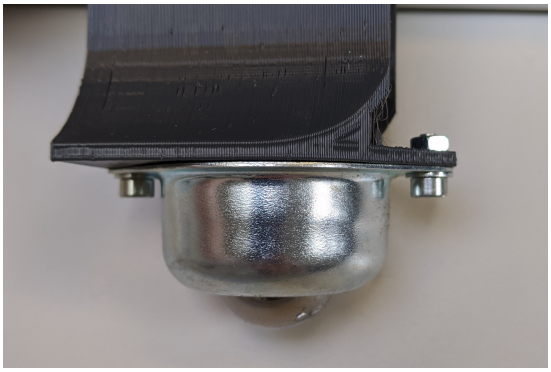


Figure 7.1: One ball castor bolted to its attachment.

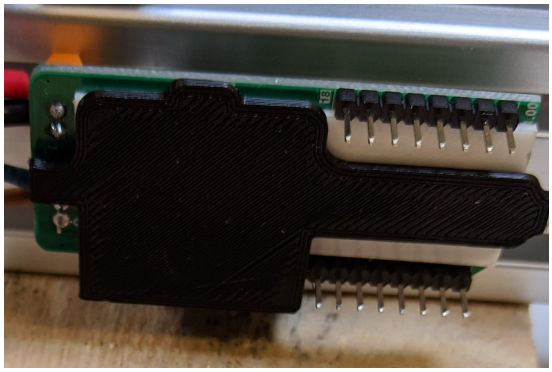


Figure 7.2: One motor controller and mounts, and how it is attached.

After this, one 40 cm and two 60 cm long aluminium struts were connected with four triangle plates into a U-shape. This made it possible to slide the previously made modules onto the U-shaped frame, as can demonstrated in figure 7.3 below.

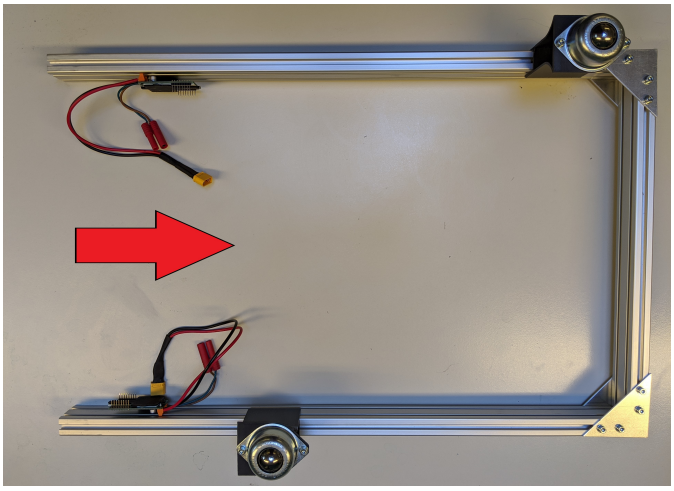


Figure 7.3: How the different mounts and modules are installed on the frame.

The next step was to bolt the motors and the encoders to the 3D-printed bracket. The motors were pushed into the bracket's motor holes. The four screw holes were lined up with the holes in the bracket. After this, the encoders were fixed in the bracket grooves with the included nut and one washer on each side. The bracket with the motor and encoder mounted can be seen in figure 7.4 below.

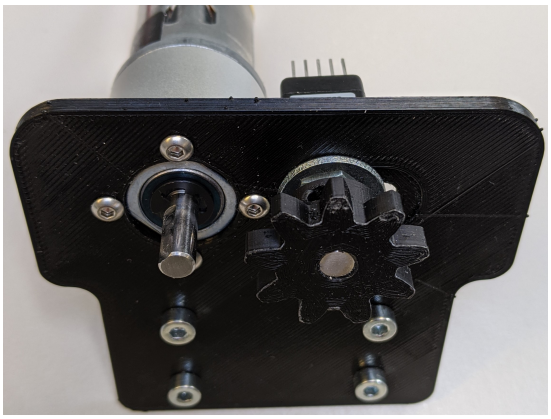
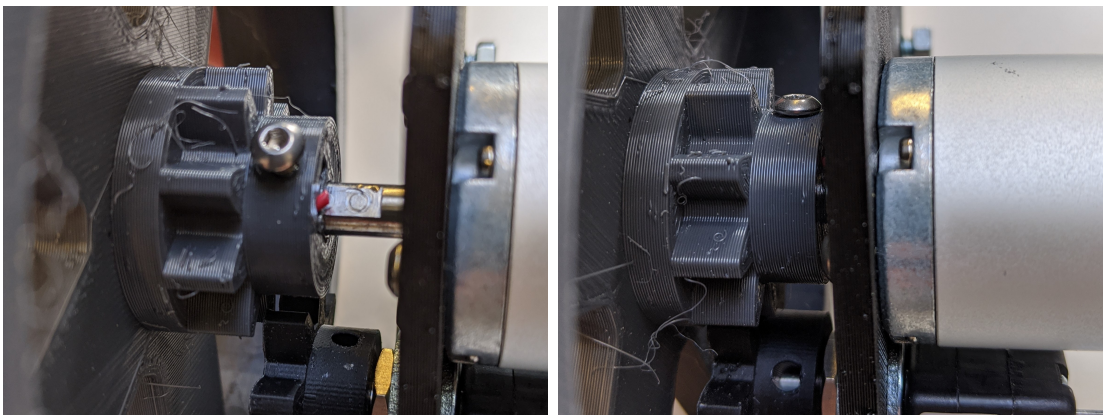


Figure 7.4: Motor and encoder mounted on the 3D-printed bracket.

Figure 7.5: Wheel gear hub with a locking nut in place.

The lock nut was placed in the wheel gear hub, as can be seen in 7.5 above. The fit between the encoder gear and the wheel was then verified. The reason for ensuring this was that the encoder cannot easily be moved after the encoder gear has been pushed onto the encoder. It can be seen in figure 7.4 above, how the encoder gear has been pushed onto the encoder axle. This was done using a vice. Finally, the wheels were carefully pushed onto the motor axles, as shown in figure 7.6 below.

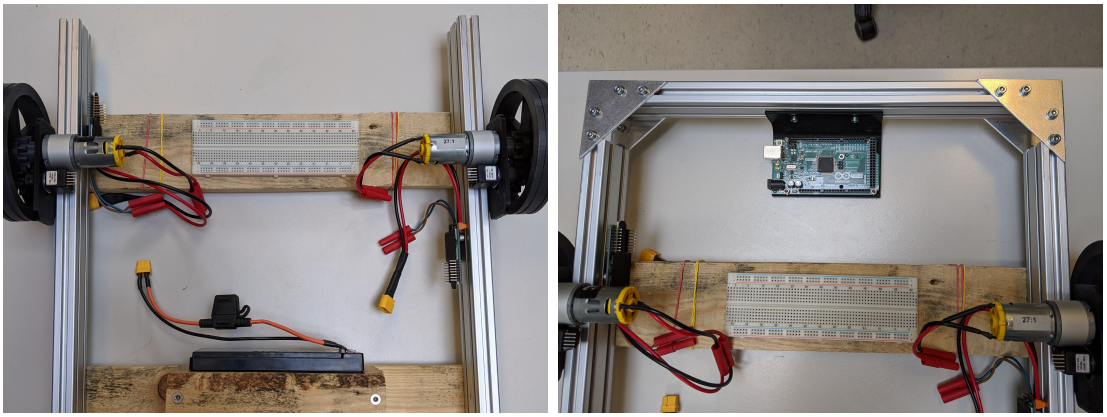


(a) (b)

Figure 7.6: Mounting a wheel and locking it onto the flat motor axle.

With the motors, the encoders and the wheels properly attached to the brackets, bolts were added with nuts partially tightened, leaving enough space to enable the brackets to be slid onto the U-frame, as the custom 3D-prints were. The bracket bolts were then properly tightened at the correct position. This can be seen in

figure 7.7 below. The battery holder holding the battery, and another wooden board with a breadboard have also been slid onto the frame. In figure 7.7b below it can be seen how the frame was closed with the last strut with the Arduino on its holder attached to it.



(a) (b)

Figure 7.7: Front wheel modules, battery holder and a wooden platform mounted on the U-frame in picture. Right picture: The last strut with the Arduino mounted are closing up the frame.

The only hardware left to assemble were the two USS-modules and the Raspberry Pi with its accompanying camera. It can be seen how one USS-module is attached to the frame in figure 7.8 below, and in figure 7.9 below it can be seen how the camera module was mounted on the camera attachment with rubber bands.

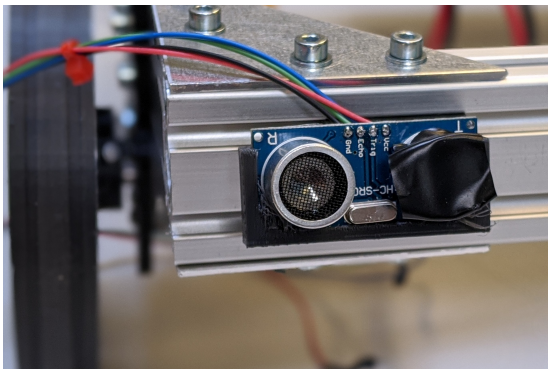


Figure 7.8: One USS-module mounted on the frame.

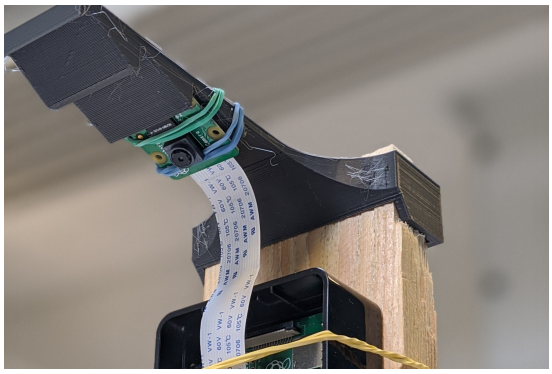
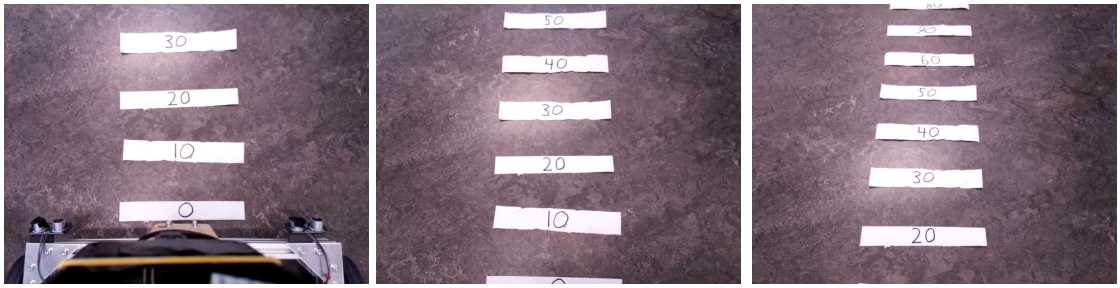


Figure 7.9: The Raspberry Pi and the camera mounted with rubber bands.

The camera module was mounted in the three different angles on the camera attachment to test how far the camera could see from each position. This was done by placing slips of paper on the ground in front the prototype. The numbers written on each paper slips represent the distance from the front of the cart. The view from the camera of each position can be seen in figure 7.10 below.



(a) Position 1 - 10° (b) Position 2 - 20° (c) Position 3 - 30°

Figure 7.10: The camera field of view of the three different camera angles. All angles are measured with the horizontal line as zero.

In the first position, figure 7.10a, the camera's field of view is about 35 cm in front of the cart, and also proves that the camera height is sufficient to provide vision of the width of the cart, since both wheels are visible in this picture. In the second position, figure 7.10b, the field of view is over 50 cm, while still being able to see the paper slip numbered zero. In the last position, figure 7.10c, the field of view is increased to 80 cm, but the marks for 0 and 10 cm are no longer visible. Since the second position provided the furthest vision, while still having vision of the immediate front of the cart, it was the chosen camera placement.

7.2 Wiring

In this section, the wiring of electronic components are presented through the use of electronic diagrams. The electronic diagram of the customer unit can be seen in figure 7.11 below.

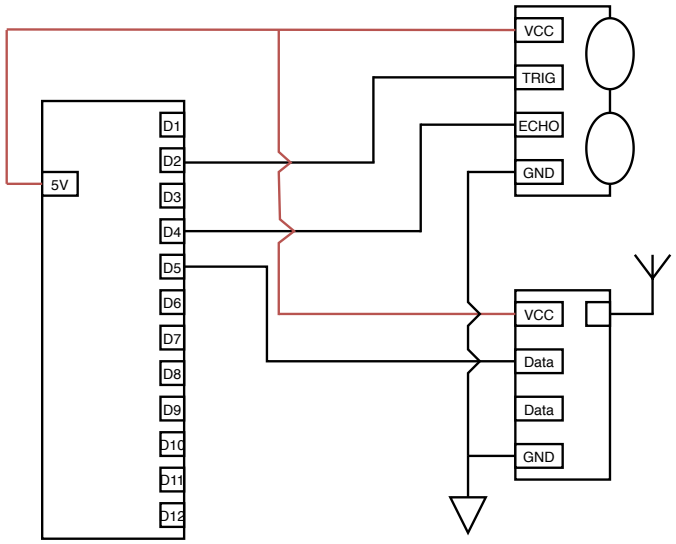


Figure 7.11: Electronic diagram showing the wiring of the customer unit sensors and microcontroller. The left part is the Arduino Nano, the top right part is the USS and the bottom right one is the radio receiver and antenna.

The red wires are supply voltage to the components. On the left the Arduino Nano can be seen with most of its digital pins drawn. Three digital pins were connected to a component pin according to the diagram. Lastly the ground pins were connected to the ground pin on the Arduino, though not shown.

The next wiring task was to connect the motor controllers with motors and battery. From the battery, a parallel connector yielding two outlet connections, one for each motor controller, were connected. Then the two separate motors was connected to one motor controller each. The connections for one motor controller and one motor can be seen in figure 7.12 below.

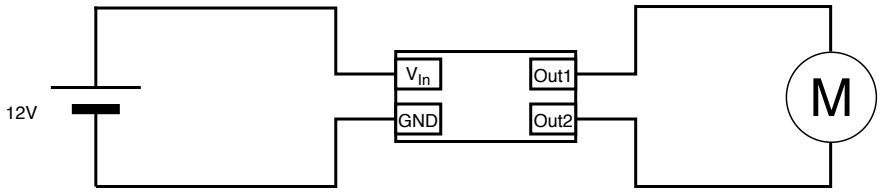


Figure 7.12: Electronic diagram showing the wiring of battery, motor and motor control. The motor control part is also wired to the microcontroller as shown above.

Most connection was done with connectors that were soldered to a cable. For connecting the soldered wires to the motor controllers, there were screw terminals available on the motor controllers.

The more complicated wiring process of the project was wiring all of the cart's components together. Below, in figure 7.13, the wiring is presented.

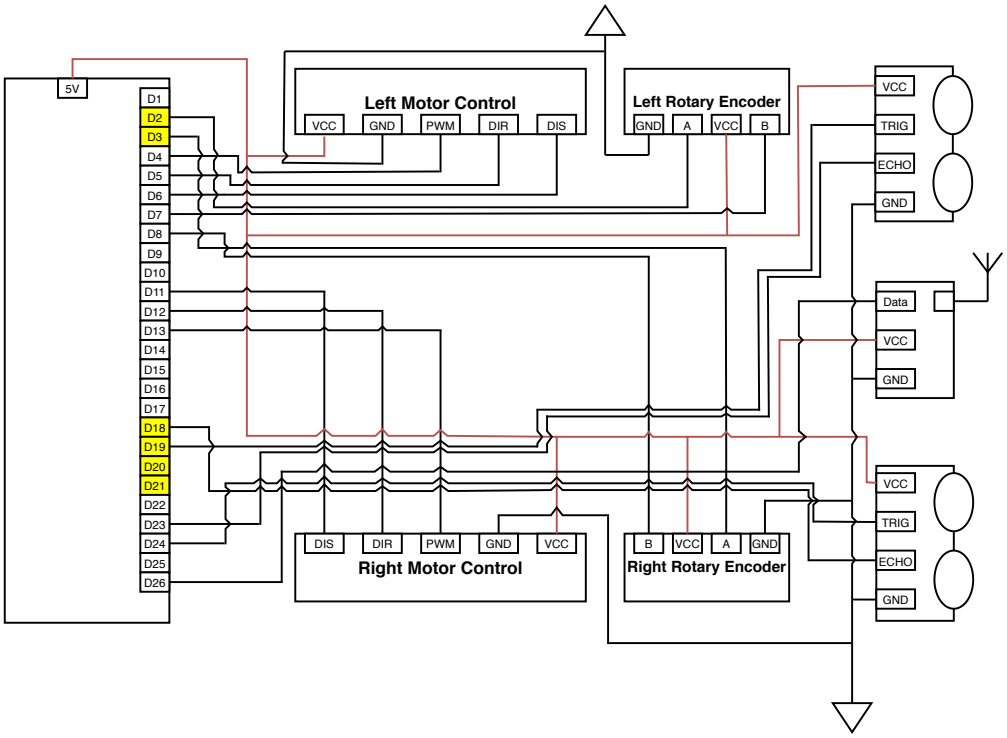


Figure 7.13: Electronic diagram showing the wiring of the microcontroller on the cart. The motor controller are connected to a battery and a motor outside of this chart. The radio and sonic sensors are on the right. The coloured pins on the microcontroller are interrupting.

In this figure the pins on the Arduino Mega that is interrupting are coloured yellow, and all red cables are supply voltages.

7.3 Software Integration

During the development of the project the software for each subsystem was developed independently. In order for all code to run simultaneously on the Arduino all software had to be joined to one Arduino sketch. In an Arduino sketch there are two required functions, setup [32] and loop [33]. The setup function will run once when the Arduino starts, and loop will run continuously. This meant that all separate setup and loop functions had to be joined to a single setup and a single loop. To achieve this the separate setup functions were renamed and called by the main setup function. There were two loop functions, one for the FTC system and one for the motor control. These systems initially worked independent of each other, and therefore the code from the motor control was moved to the start of the loop function for the FTC system.

An Arduino sketch can have separate files, as long as there is only one setup and loop, two header files [34] were created in order to simplify the code . One for all variables and constants, and one for the pin numbering. The code for the motor control was kept in a separate file, and the functions used for the FTC system was

put into another separate file. Finally, the code for the PID controller was kept in one file.

7.3.1 Control System

To enable the cart to follow the customer the motors had to be controlled by the output from the FTC system.The planned solution for this was presented in section 4.4.3, however due to the late delivery of the encoders there is not enough time for this. Instead a more basic system was developed, which will be presented here.

To test the systems a very basic control system was created by enabling the motors when the measured distance to the customer was greater than one meter, and no objects were detected by the collision avoidance system. It was predicted that the cart would then drive in a straight line. When this was tested it was confirmed that the motors received power once the CU was further away than one meter, however the cart did not go in a straight line. This was due to insufficient friction between the wheels and the floor which led to the wheels spinning. This issue was fixed by putting rubber bands on the wheels, which increased the friction and enabled the cart to go in a straight line.

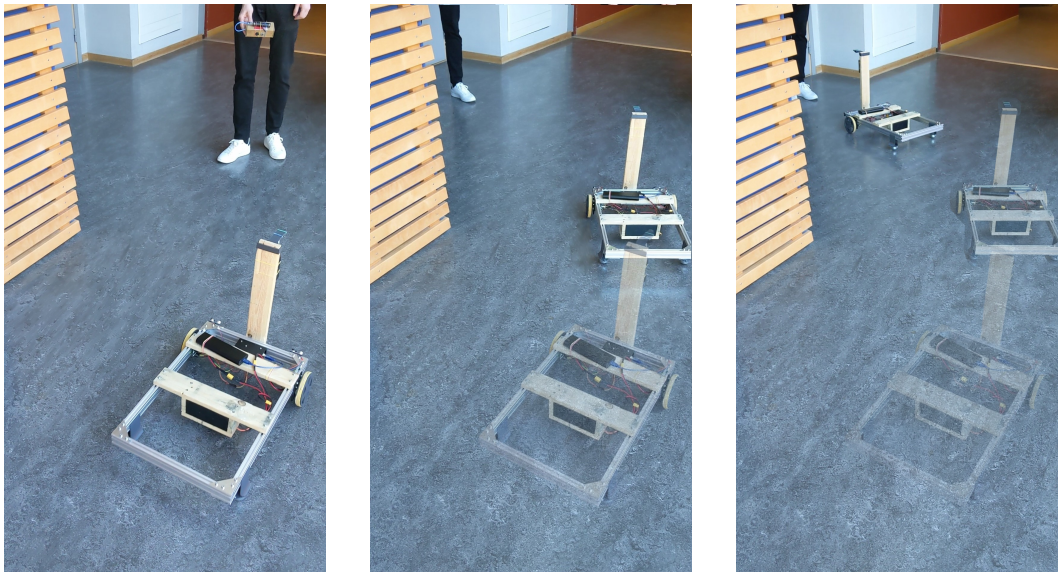
The next step was to enable the cart to turn when the customer is turning. Initially this was done by by lowering the speed of the left or right wheel by 50% if the distance difference between the two USS was greater than 10 cm. It was predicted that the cart would turn until the distance difference was less than 10 cm as this measurement was updated about 10 times per second. When this was tested it was realised that a speed reduction of 50% was too much as the cart turned faster than it could recover from. Thus the speed was set to 80% for the corresponding wheel. This change enabled the cart to turn and follow the customer somewhat. When the customer went around a corner the cart would turn too long and lose track of the customer. To counter this it was tested to reset the wheel speed to the initial value every 300 ms. This was done by incrementing a variable every iteration of the loop function, which takes approximately 100 ms to complete. If this variable was divisible by three the wheel speeds would be reset. When this was tested it was confirmed that it enabled the cart to follow the customer around corners as well as in a straight line.

8 Results and Evaluation

In the following sections, the systems movement and data from the sensors will be presented in a few figures and some data-sets. The total cost of the project is also presented.

8.1 Movement of the Cart

In the three figures below (8.1a, 8.1b and 8.1c) a demonstration of the prototype moving is shown. The first picture shows the starting position of both the cart and the customer. The second shows the cart has moved towards the customers original position, while the customer has moved to a different position. The third shows the cart arriving at the customer. The photos are taken a few seconds apart.



(a) Cart starts moving towards the customer.

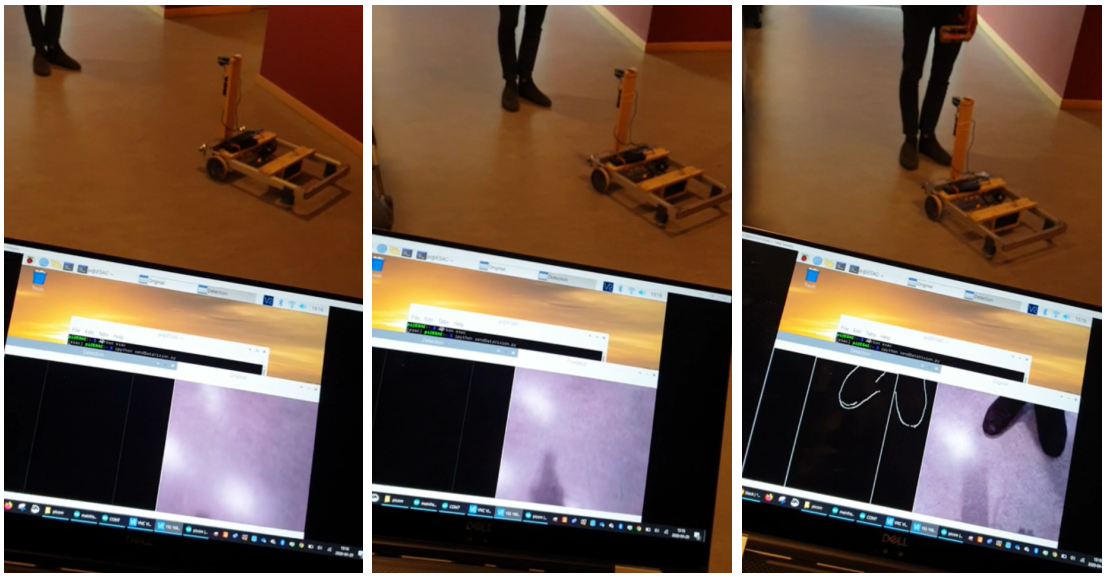
(b) Customer moves around the corner, cart keeps following.

(c) Cart is close enough to the customer and stops.

Figure 8.1: Demonstration of the cart following a customer around a corner, with its previous positions marked.

8.2 Collision Detection

The following figures (8.2a, 8.2b and 8.2c) show how the collision detection system works, and that it only stops when the camera actually sees a real obstacle (the customers feet).



(a) Cart only sees glare, not interpreted as an object.

(b) Cart sees glare and its own shadow, not interpreted as objects.

(c) Cart sees shoes of customer interpreted as collision and stops.

Figure 8.2: Demonstration of carts vision when moving towards customer in low light conditions.

8.3 Triangulation Sensors

In the following two tables (1 and 2) data from the triangulation software is shown. In 1 the distance from the customer to the cart is measured straight ahead, while in 2 the customer unit is placed at an angle to the right and then the left ahead of the cart.

Table 1: Distances measured between the customer unit and the cart at two different distances. The distances at the top were measured with a tape measure, and the columns under each of those are outputs from the sensors. The distances are measured at an angle from the ground, straight between the sensors, so the actual distance between the cart and customer will be slightly shorter.

Distance from sensor to customer (cm)	108	202
Distances measured by sensors (cm)	109,90	199,20
	109,55	198,90
	108,70	200,61
	107,36	198,31
	108,05	202,34

Table 2: Difference in distances measured between the left and right sonic sensors on the cart. If the customer is on the right relative to the cart, the difference is positive, when on the left, the difference is negative.

Distance from left sensor (cm)	207	206
Distance from right sensor (cm)	197	217
Difference in distance measure by the sensors (cm)	11,37	−11,64
	10,80	−12,02
	10,74	−11,67
	11,32	−10,96
	11,56	−11,12

8.4 Hardware Cost

The total cost of the project is presented in table 3 below. Due to some of the components being given in kind, another table is shown 4, presenting the cost of equivalent components and the total cost of the project had they been bought.

Table 3: In this table, the total cost of the project is presented. Some of the components where donated in kind, therefore their cost is presented as 0.

Part	**Product**	**Qty.**	**Cost (SEK)**
Ultrasonic Sensor	SparkFun HC-SR04	3	131.64
Rotary Encoder	Bourns ENS1J-B28-L00256L	2	1455
Camera Module	Raspberry Pi Camera Module V2.1	1	540.53
Radio Transmitter	FS1000a	1	0
Radio Receiver	XY-MK-5V	1	0
Transmitter Controller	Arduino Nano 3.0	1	0
Microcontroller	Arduino Mega 2650 REV3	1	443.89
Camera Computer	Raspberry Pi 4 Model B 2G	1	540.53
Geared Motor	RS PRO 834-7663	2	1137.4
Motor Controller	MIKROE-3613	2	465.0
Motor Battery	RS PRO 537-5488	1	238.99
Aluminium Strut	RS PRO 850-8482	2	334.8
3D-printing plastic	PrimaValue PLA 1.75mm	-	120
		Total:	5407.78

Table 4: In this table, the cost of components equivalent to the ones donated are presented. The total cost of the prototype if these components are also presented.

Part	**Product**	**Qty.**	**Cost (SEK)**
Radio Transmitter	Geekcreit XD-FST	1	25.0
Radio Receiver	Greekcreit XD-RF-5V	1	25.0
Transmitter Controller	Arduino Nano 3.0	1	179.87
	Cost with donation:		5407.78
	Cost with equivalent components:		5637.65

8.5 Evaluation

As is shown in the previous section, the prototype follows the customer and stops when encountering obstacles, two of the main goals. This is a bit slower than wanted, but this is mostly due to the optimisation of the regulation not being possible due to time constraints. The fact that the cart pulls to the right when starting is also a result of this. Overall the prototype completes what it was set out to do.

It is also shown that the budget was exceed by around 600 SEK, but this is mostly due to the rotary encoders that were out of stock and more expensive ones had to be used.

9 Discussion

In this chapter, the results in the previous chapter are discussed. How well the chosen solutions work as well as ethical and environmental impact are also discussed.

9.1 Ultra Sonic Triangulation

In chapter 3, the solution used to triangulate customer location is shown. The system uses two ultra sonic receivers to measure a time delay from one transmitter, that transmits when called upon by a radio module.

In order for the receivers to pick up a signal, the customer unit needs to be in front of the cart, otherwise the customer will be outside of the sensors' narrow field of view. A solution to this could be to place more sensors on the cart in order to increase the field of view, which also would improve the cart's ability to follow the customer around corners.

Another problem with the current solution is the variance in time, which mostly seems to come from the radio modules, which were quite cheap, so it was to be expected. Some solutions to this could be to use another type of sensor to signal when to send a sonic ping, like an IR sensor. That solution would be limited by a field of view however, so a better way to reduce the variance may be to use higher quality radio modules.

The chosen solution shows promise, but could use the upgrades described above, if future improvements where to be made.

9.2 Collision Avoidance

The camera based method for avoiding obstacles described in chapter 5 used a Canny edge filter in order to detect the edges of objects. Tuning this filter to pick up objects while filtering out false positives such as floor textures, glare and shadows proved to be one of the biggest challenges, since a slight change in lighting could change the outcome. A basic auto-calibration method was implemented in order to try and make the results consistent. This worked to a certain degree, but in some cases the system still has problems, for example, very sharp shadows produced by direct sunlight will still be picked up as an object. However, such shadows are unlikely to appear inside a grocery store, since windows are uncommon and the lighting is usually more or less consistent inside the store.

The camera placement provided a birds eye view of the immediate area in front of the cart. The width of the visible area was as wide as the cart, and the vision extended out to about 50cm in front of the cart. This relatively small area was ideal to use as a "collision-zone", meaning, that if any objects appear in the cart's field of view, the cart would stop in order to avoid collision with that object. This solution works well if there are mobile objects, such as a person stepping in front

of the cart. However, if the object blocking the cart is static, the cart will be stuck until the object is moved. To solve this problem an active collision avoidance system was developed. This system was intended to actively choose a path around any obstacle in the cart's path. While the idea behind this solution might be sound, the narrow field of view now became an issue, as a larger area would be needed in order to be able to pathfind around the objects. Placing the camera higher up or using a camera with a larger field of view would result in a larger area, which would help solve this issue. And in the end, considering that the system is intended to be implemented into a full-sized shopping cart the camera mounting would by default be higher up.

9.3 Environmental and Ethical Aspects

Due to the life cycle of the lead-acid battery, the cart will mostly impact the environment negatively compared to a regular shopping cart, without electrical motors. However, the improvements it can provide to elderly an people with impaired movement who can have a hard time using regular shopping cart, is an important aspect that can outweigh the negative environmental aspect, if taken care of properly.

10 Conclusions

The assembled prototype provides the customer with an autonomous cart carrying a shopping basket. The cart can follow the customer in a store. The FTC-system provides the motor control system with data which enables the motors to respond to the customer's movements. The CA system provides safety as it disables the motors if any object is in a direct path in front of the cart. The aim for the project has been reached, and therefor the project is deemed successful.

10.1 Future Work

Due to the limitations of the project some solutions and functionality was not considered. These limitations include time, budget, and unavailability of hardware. The unconsidered solutions and functionality will be discussed in this chapter.

During the planning stage of the project it was discussed whether the prototype would be designed as a miniature or full sized cart. It was argued that too much time would be spent building a full size shopping cart, and that it would require more powerful motors than the budget could cover. Therefor a smaller prototype was build, so that more time could be spend on the FTC system, the CD and CA systems. A proposal for future work is to build a full size shopping cart based on the produced smaller prototype.

Two proposed functions for the prototype was a manual override, and for the prototype to be able to reverse. A manual override could be needed if the cart encounters an obstacle which it cannot go around. To be able to reverse could be useful in smaller aisles or if the customer wants to go straight back. Future work could be done implementing these functions.

As described in subsection 7.3.1 the control system is fairly basic. Further work could be spent on refining the control system in order for the cart to work better in its proposed environment.

Another function that was not implemented due to complexity and time limitation was a pathfinding system. Such a system could be used to the cart to drive to a specific place in a store based the items on a shopping list. It could also be used to autonomously drive to a charging station to charge its batteries. Further use includes waiting at set location and driving to the customer when called upon.

The subsystems build in this project can also be applied elsewhere. The FTC system could be used in many different areas, one of which was the inspiration for how it was build, as seen in [2]. Another use might be where an action takes place once a person appears within a certain area, for example to tell when to start the coffee machine, or when to turn on a light. The CA system could be used in any situation when objects need to be detected, for example in cars.

References

[1] G. Synek, "Lg is putting robot shopping carts into retail stores," Available at https://www.techspot.com/news/77273-lg-putting-robot-shopping-carts-retail-stores.html , Accessed on 2020/02/11.

[2] B. Heck and J. Robinsson, "Make your suitcase haul itself around with robot luggage!" Available at https://youtu.be/tAyWrJoVUbs, Accessed on 2020/02/11, 2012.

[3] B. V. Pradeep, E. S. Rahul, and R. R. Bhavani, "Follow me robot using bluetooth-based position estimation," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2017, pp. 584–589.

[4] Q. K. Dang and Y. S. Suh, "Human-following robot using infrared camera," in *2011 11th International Conference on Control, Automation and Systems*, Oct 2011, pp. 1054–1058.

[5] J. Constine, "Meet caper, the ai self-checkout shopping cart," Available: https://techcrunch.com/2019/01/10/caper-shopping-cart/ , Accessed on 2020/02/13.

[6] J. Carlback, F. Eriksson, J. Gästrin, M. Hjorth, J. Holmström, and O. Thompsson, "Development and evaluation of an autonomous electrically assisted shopping cart - planning report," 2020, our own planning report for this project.

[7] Elecfreaks, "Ultrasonic ranging module hc - sr04," Avaliable: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf , Accessed on 2020/04/02.

[8] Last Minute Engineers, "How hc-sr04 ultrasonic sensor works & interface it with arduino," Avaliable: https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/, Accessed on 2020/04/16.

[9] P. S, "Fs1000a and xy-mk-5v 433mhz rf modules: overview," Available: https://quadmeup.com/fs1000a-and-xy-mk-5v-433mhz-rf-modules-overview/, Accessed on 2020/04/02.

[10] diy_bloke, "433 mhz coil loaded antenna," Avaliable: https://www.instructables.com/id/433-MHz-Coil-loaded-antenna/, Accessed on 2020/04/02.

[11] Arduino, "pulsein," Avaliable: https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/, Accessed on 2020/04/23.

[12] Arduino, "interrupt," Avaliable: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/, Accessed on 2020/04/23.

[13] Arduino, "micros()," Avaliable: https://www.arduino.cc/reference/tr/language/functions/time/micros/, Accessed on 2020/04/16.

[14] Servocity, "Common steering methods," Avaliable: https://www.servocity.com/blog/common-steering-methods , Accessed on 2020/05/11.

[15] A. R. Hambley, *Electrical Engineering: Principles and Applications*, 6th ed. A. Gillfillan, Ed. Upper Saddle River, NJ, USA: Pearson, 2014, pp. 754–802.

[16] MIKROE, "H-bridge 3 click schematic v100," Avaliable: https://www.elfa.se/Web/Downloads/_t/ds/MIKROE-3613_eng_tds.pdf , Accessed on 2020/05/14.

[17] B. Lennartson, *Reglerteknikens grunder*, 4th ed. Lund, Sweden: Studentlitteratur, 2002, pp. 301–356.

[18] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu, "The obstacle detection and obstacle avoidance algorithm based on 2-d lidar," in *2015 IEEE International Conference on Information and Automation*, 2015, pp. 1648–1653.

[19] N. Baras, G. Nantzios, D. Ziouzios, and M. Dasygenis, "Autonomous obstacle avoidance vehicle using lidar and an embedded system," in *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2019, pp. 1–4.

[20] D. Ghorpade, A. D. Thakare, and S. Doiphode, "Obstacle detection and avoidance algorithm for autonomous mobile robot using 2d lidar," in *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, 2017, pp. 1–6.

[21] Teng Wang, Leping Bu, and Zhongyi Huang, "A new method for obstacle detection based on kinect depth image," in *2015 Chinese Automation Congress (CAC)*, 2015, pp. 537–541.

[22] D. Yi, T. Lee, and D. D. Cho, "An inverse perspective mapping technique based on a virtual-vertical plane model for obstacle detection using mono camera," in *2015 15th International Conference on Control, Automation and Systems (ICCAS)*, 2015, pp. 1216–1219.

[23] K. A. B. Za'aba and L. B. Theng, "Edge based obstacle detection model for outdoor type obstacles," in *2019 International UNIMAS STEM 12th Engineering Conference (EnCon)*, 2019, pp. 28–33.

[24] R. Zhang, Y. Yang, W. Wang, L. Zeng, J. Chen, and S. McGrath, "An algorithm for obstacle detection based on yolo and light filed camera," in *2018 12th International Conference on Sensing Technology (ICST)*, 2018, pp. 223–226.

[25] Rostam Affendi Hamzah, Hasrul Nisham Rosly, and S. Hamid, "An obstacle detection and avoidance of a mobile robot with stereo vision camera," in *2011 International Conference on Electronic Devices, Systems and Applications (ICEDSA)*, 2011, pp. 104–108.

[26] Laserlicht, "Raspberry pi 4 model b - side," Available at https://commons.wikimedia.org/wiki/File:Raspberry_Pi_4_Model_B_-_Side.jpg Accessed on: 2020/05/13.

[27] T. R. P. Foundation, "Raspberry pi camera module v2 with ribbon," Available at https://commons.wikimedia.org/w/index.php?curid=69775243 Accessed on: 2020/05/13.

[28] O. Tutorials, "Canny edge detection – opencv - python tutorials 1 documentation," Avaliable: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html , Accessed on 2020/03/12.

[29] dockerizeme, "cv2.canny python example," Avaliable: https://www.programcreek.com/python/example/89361/cv2.Canny , Accessed on 2020/03/15.

[30] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science Engineering*, vol. 13, no. 2, pp. 31–39, 2011.

[31] A. Rosebrock, "Fast, optimized 'for' pixel loops with opencv and python," Avaliable: https://www.pyimagesearch.com/2017/08/28/fast-optimized-for-pixel-loops-with-opencv-and-python/ , Accessed on 2020/04/15.

[32] Arduino, "setup," Avaliable: https://www.arduino.cc/reference/en/language/structure/sketch/setup/, Accessed on 2020/05/04.

[33] Arduino, "loop," Avaliable: https://www.arduino.cc/reference/en/language/structure/sketch/loop/, Accessed on 2020/05/04.

[34] TutorialsPoint, "header file," Avaliable: https://www.tutorialspoint.com/cprogramming/c_header_files.htm, Accessed on 2020/05/07.

A Appendix

A.1 Arduino Source Code

A.1.1 The Main Program Running on the Arduino

```
void setup() {
    setup_FTC();
    setup_motor();
}
void loop() {
  yolo++; // + 1 ish = 100 ms
if (Serial.available() > 0) {
    int data = Serial.read() - '0'; //Reads next byte, and converts
    to int (fulkodat)
    if (data == 1) {
      blocked = true;
    }
    else if(data ==0){
      blocked = false;
    }
  }
  if(!blocked){
    enableMotors();
    if(distance_result > 100){
      if (result > 10){
        setpointLeft = (int)setpointLeft*0.8;
        if(yolo%3 == 0)
          setpointLeft = setpointInit;
        // turn right
      }
      else if(result < -10){
        setpointRight = (int)setpointRight*0.8;
        if(yolo%3 == 0)
          setpointRight = setpointInit;
      }//turn left
      else{
         setpointRight=setpointInit;
         setpointLeft=setpointInit;
      }
    }
    else{
      setpointLeft = 0;
      setpointRight = 0;
    }
  }//if(!blocked)
  else{
    disableMotors();
    setpointLeft = 0;
    setpointRight = 0;
    }
  updateRPM();
  computePID();
  setPWM(LEFT, outputLeft);
  setPWM(RIGHT, outputRight);
  }
  request_ping();
  sens_trig(sens_right.trigPin);
```

```
52    sens_trig(sens_left.trigPin);
53    delay(50); //Needed for some reason
54
55    sens_right.dist = sens_right.timer*distmul;
56    sens_left.dist = sens_left.timer*distmul;
57    diff = sens_right.dist-sens_left.dist;
58    distance = (sens_right.dist+sens_left.dist-140)/2; //constant +70cm
        for each sensor, hence -140
59
60    if(abs(diff) < width){ //Max diff possible is width between the two
        sensors.
61      mean[counter] = diff;
62      mean_dist[counter] = distance;
63      counter++;
64      if(counter > cap) //Used for calculating means
65        counter = 0;
66      result = calc_mean_diff();
67      distance_result = calc_mean_dist();
68    }
69    else{
70    }
71    delay(50); //update once every 50 ms
72  }
```

Listing 1: The main program running on the arduino

A.1.2 Source Code for PulseIn

```
1   unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long
      timeout)
2   {
3     // cache the port and bit of the pin in order to speed up the
4     // pulse width measuring loop and achieve finer resolution.
        calling
5     // digitalRead() instead yields much coarser resolution.
6     uint8_t bit = digitalPinToBitMask(pin);
7     uint8_t port = digitalPinToPort(pin);
8     uint8_t stateMask = (state ? bit : 0);
9
10    // convert the timeout from microseconds to a number of times
        through
11    // the initial loop; it takes approximately 16 clock cycles per
        iteration
12    unsigned long maxloops = microsecondsToClockCycles(timeout)/16;
13
14    unsigned long width = countPulseASM(portInputRegister(port), bit,
        stateMask, maxloops);
15
16    // prevent clockCyclesToMicroseconds to return bogus values if
        countPulseASM timed out
17    if (width)
18      return clockCyclesToMicroseconds(width * 16 + 16);
19    else
20      return 0;
21  }
```

Listing 2: Snippet of arduino source code

A.1.3 ASM Code Called by PulseIn

Listing 3: The assembler code called by code in the previous section

```
/*
  wiring_pulse.s - pulseInASM() function in different flavours
  Part of Arduino - http://www.arduino.cc/

  Copyright (c) 2014 Martino Facchin

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General
  Public License along with this library; if not, write to the
  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
  Boston, MA 02111-1307 USA
*/

/*
 * The following routine was generated by avr-gcc 4.8.3 with
 * the following parameters
 * -gstabs -Wa,-ahlmsd=output.lst -dp -fverbose-asm -O2
 * on the original C function
 *
 * unsigned long pulseInSimpl(volatile uint8_t *port,
 * uint8_t bit, uint8_t stateMask, unsigned long maxloops)
 * {
 *     unsigned long width = 0;
 *     // wait for any previous pulse to end
 *     while ((*port & bit) == stateMask)
 *         if (--maxloops == 0)
 *             return 0;
 *
 *     // wait for the pulse to start
 *     while ((*port & bit) != stateMask)
 *         if (--maxloops == 0)
 *             return 0;
 *
 *     // wait for the pulse to stop
 *     while ((*port & bit) == stateMask) {
```

```
 *          if (++width == maxloops)
 *              return 0;
 *      }
 *      return width;
 * }
 *
 * some compiler outputs were removed but the rest of the
 * code is untouched
 */

#include <avr/io.h>

.section .text

.global countPulseASM

countPulseASM:

.LM0:
.LFBB1:
    push r12    ;    ;  130 pushqi1/1 [length = 1]
    push r13    ;    ;  131 pushqi1/1 [length = 1]
    push r14    ;    ;  132 pushqi1/1 [length = 1]
    push r15    ;    ;  133 pushqi1/1 [length = 1]
    push r16    ;    ;  134 pushqi1/1 [length = 1]
    push r17    ;    ;  135 pushqi1/1 [length = 1]
/* prologue: function */
/* frame size = 0 */
/* stack size = 6 */
.L__stack_usage = 6
    mov r30,r24   ;  port, port    ;  2 *movhi/1  [length = 2]
    mov r31,r25   ;  port, port
/*      unsigned long width = 0;
***       // wait for any previous pulse to end
***       while ((*port & bit) == stateMask)
*/
.LM1:
    rjmp .L2    ;    ;  181 jump  [length = 1]
.L4:
/*          if (--maxloops == 0) */
.LM2:
    subi r16,1    ;  maxloops,   ;  17  addsi3/2  [length = 4]
    sbc r17, r1    ;  maxloops
    sbc r18, r1    ;  maxloops
    sbc r19, r1    ;  maxloops
    breq .L13   ; ,    ;  19  branch  [length = 1]
.L2:
```

```
/*          if (--maxloops == 0) */
.LM3:
    ld r25,Z    ;  D.1554, *port_7(D)    ;  22  movqi_insn/4
[length = 1]
    and r25,r22   ;  D.1554, bit   ;  24  andqi3/1  [length = 1]
    cp r25,r20    ;  D.1554, stateMask   ;  25  *cmpqi/2  [length = 1]
    breq .L4    ; ,    ;  26  branch  [length = 1]
    rjmp .L6    ;    ;  184 jump  [length = 1]
.L7:
/*              return 0;
***
***     // wait for the pulse to start
***     while ((*port & bit) != stateMask)
***         if (--maxloops == 0)
*/
.LM4:
    subi r16,1    ;  maxloops,   ;  31  addsi3/2  [length = 4]
    sbc r17, r1    ;  maxloops
    sbc r18, r1    ;  maxloops
    sbc r19, r1    ;  maxloops
    breq .L13   ; ,    ;  33  branch  [length = 1]
.L6:
/*          if (--maxloops == 0) */
.LM5:
    ld r25,Z    ;  D.1554, *port_7(D)    ;  41  movqi_insn/4
[length = 1]
    and r25,r22   ;  D.1554, bit   ;  43  andqi3/1  [length = 1]
    cpse r25,r20    ;  D.1554, stateMask   ;  44  enable_interrupt-3
[length = 1]
    rjmp .L7    ;
    mov r12, r1    ;  width   ;  7 *movsi/2  [length = 4]
    mov r13, r1    ;  width
    mov r14, r1    ;  width
    mov r15, r1    ;  width
    rjmp .L9    ;    ;  186 jump  [length = 1]
.L10:
/*              return 0;
***
***     // wait for the pulse to stop
***     while ((*port & bit) == stateMask) {
***         if (++width == maxloops)
*/
.LM6:
    ldi r24,-1    ; ,    ;  50  addsi3/3  [length = 5]
    sub r12,r24   ;  width,
    sbc r13,r24   ;  width,
    sbc r14,r24   ;  width,
```

```
    sbc r15,r24   ;  width,
    cp r16,r12    ;  maxloops, width  ;  51  *cmpsi/2  [length = 4]
    cpc r17,r13   ;  maxloops, width
    cpc r18,r14   ;  maxloops, width
    cpc r19,r15   ;  maxloops, width
    breq .L13   ; ,    ;  52  branch  [length = 1]
.L9:
/*          if (++width == maxloops) */
.LM7:
    ld r24,Z    ;  D.1554, *port_7(D)    ;  60  movqi_insn/4
[length = 1]
    and r24,r22   ;  D.1554, bit  ;  62  andqi3/1  [length = 1]
    cp r24,r20    ;  D.1554, stateMask  ;  63  *cmpqi/2  [length = 1]
    breq .L10   ; ,    ;  64  branch  [length = 1]
/*              return 0;
***     }
***     return width;
*/
.LM8:
    mov r22,r12   ;  D.1553, width  ;  108 movqi_insn/1  [length = 1]
    mov r23,r13   ;  D.1553, width  ;  109 movqi_insn/1  [length = 1]
    mov r24,r14   ;  D.1553, width  ;  110 movqi_insn/1  [length = 1]
    mov r25,r15   ;  D.1553, width  ;  111 movqi_insn/1  [length = 1]
/* epilogue start */
.LM9:
    pop r17   ;    ;  171 popqi [length = 1]
    pop r16   ;    ;  172 popqi [length = 1]
    pop r15   ;    ;  173 popqi [length = 1]
    pop r14   ;    ;  174 popqi [length = 1]
    pop r13   ;    ;  175 popqi [length = 1]
    pop r12   ;    ;  176 popqi [length = 1]
    ret   ;  177 return_from_epilogue  [length = 1]
.L13:
.LM10:
    ldi r22,0   ;  D.1553    ;  120 movqi_insn/1  [length = 1]
    ldi r23,0   ;  D.1553    ;  121 movqi_insn/1  [length = 1]
    ldi r24,0   ;  D.1553    ;  122 movqi_insn/1  [length = 1]
    ldi r25,0   ;  D.1553    ;  123 movqi_insn/1  [length = 1]
/* epilogue start */
.LM11:
    pop r17   ;    ;  138 popqi [length = 1]
    pop r16   ;    ;  139 popqi [length = 1]
    pop r15   ;    ;  140 popqi [length = 1]
    pop r14   ;    ;  141 popqi [length = 1]
    pop r13   ;    ;  142 popqi [length = 1]
    pop r12   ;    ;  143 popqi [length = 1]
    ret   ;  144 return_from_epilogue  [length = 1]
```

A.1.4 The main loop running on the Raspberry Pi

```
from picamera import PiCamera
from picamera.array import PiRGBArray

camera = PiCamera()
rawCapture = PiRGBArray(camera, size=(w, h))

camera.resolution = (w, h)

time.sleep(1) #Wait for camera to warm up
running = True
blockedSent = True
framecount=0
framesSinceUpdate=0
blockCounter=0

while(running):
    for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image = frame.array
        median = np.average(image)
        sigma = 0.33 #0.2

        lowThresh= 70 +int(max(0, (1.0 - sigma) * median)) #125
        highThresh = 140  +int(min(255, (1.0 + sigma) * median)) #200

        newEdge = cv2.Canny(image, lowThresh, highThresh)
        blocked = detect(newEdge, imageGrid, gridData, h, w, scandisX, scandisY, blocked, init, xMid)
        gridScan(newEdge, imageGrid, gridPixelSize, gridHeight, gridWidth)

        if(blocked):
            blockCounter = blockCounter + 1
        else:
            blockCounter = 0
        if (((blocked and not blockedSent) or (blocked and framesSinceUpdate==100))and blockCounter > 3): #could be set to blocked instead, to only send data when a object is in front of the cart
            ser.write(str("1").encode('utf-8'))
            blockedSent=True
            framesSinceUpdate=0

        #ser.write(str("D").encode('utf-8'))
            #ser.write(imageGrid)
            #ser.write(str("d").encode('utf-8'))
        elif ((not blocked and blockedSent) or init or (not blocked and framesSinceUpdate==100)):
            ser.write(str("0").encode('utf-8'))
            blockedSent=False
            framesSinceUpdate=0
        init=False
        framesSinceUpdate=framesSinceUpdate + 1
        rawCapture.truncate(0)
```