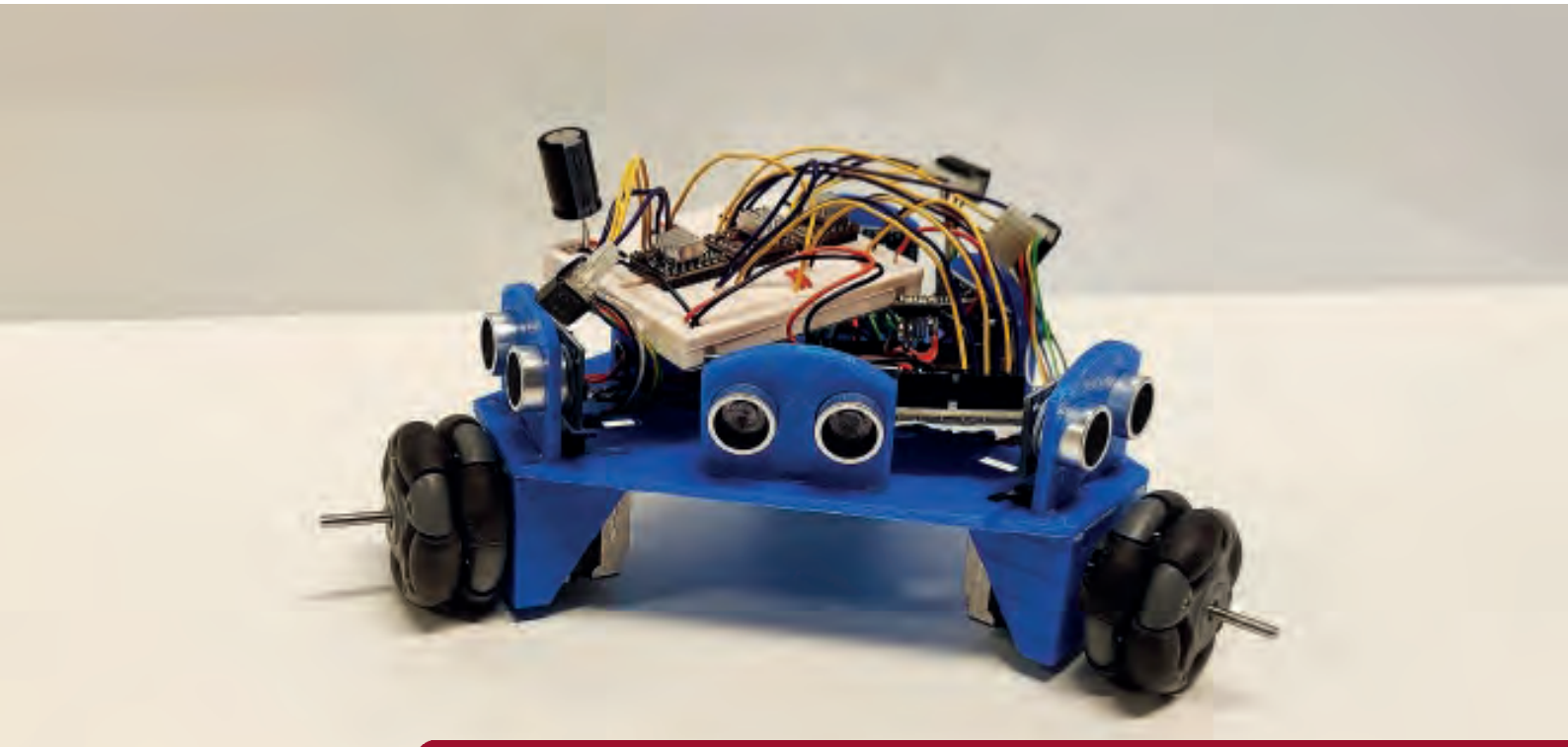
EXAMENSARBETE INOM MASKINTEKNIK,
GRUNDNIVÅ, 15 HP

STOCKHOLM, SVERIGE 2021

Omnidirektionell Robot
Omnidirectional Robot

AXEL HEDVALL

FILIP RYDÉN



KTH
SKOLAN FÖR INDUSTRIELL TEKNIK OCH MANAGEMENT

Omnidirectional Robot

AXEL HEDVALL, FILIP RYDÉN

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

TRITA ITM-EX 2021:29

Abstract

Robots are being used more and more in today's society. These robots need to be mobile and have a good understanding of their surroundings. This bachelor's thesis in mechatronics aims to see how a mobile robot can be constructed, and how it can best map its surroundings.

The robot was built to have three omni wheels to allow it to move freely in the plane and stepper motors to provide accurate movement. Ultrasonic sensors were placed around the robot to be used as a tool to determine its surroundings. The brain of the robot was an Arduino UNO, which with the help of an ESP-01, communicated with a server over Wi-Fi. The server received the data from the ultrasonic senors and drew a map on a web page.

Multiple test were made to evaluate the different systems. The robot moved really well and with high precision after some tweaking. The ultrasonic sensors were also very precise and the communication between the robot and the server worked very well. All the different systems were combined to make the robot move autonomously. The robot could navigate by itself and avoid obstacles. Although the mapping worked from a technical point of view, it was hard to read and could be done better.

Keywords: Mechatronics, Omnidirectional, Omni wheels, Wireless control, Mapping

Referat

Omnidirektionell robot

Robotar är något som används mer och mer i dagens moderna samhälle. Dessa robotar behöver vara mobila och ha en god uppfattning om miljön de befinner sig i. Detta kandidatexamensarbete inom mekatronik ska undersöka hur en mobil robot kan byggas, och hur den kan kartlägga miljön den befinner sig i.

Roboten som konstruerades hade tre omnihjul för att kunna röra sig fritt längs markplanet och stegmotorer för precis drift. Ultraljudsensorer placerades runt om roboten för att ge den en uppfattning av omgivningen. Hjärnan i roboten var en Arduino UNO som med hjälp av en ESP-01 kommunicerade över Wi-Fi till en server. Servern tog emot sensordata från roboten och ritade upp det som en karta i en webbläsare.

Det utfördes tester för att utvärdera de olika delsystemen. Driften på roboten fungerade utmärkt med god precision efter några iterationer. Ultraljudsensorerna hade också god precision och kommunikationen mellan roboten och servern fungerade mycket bra. De olika delsystemen kombinerades för att ge roboten självkörning. Roboten kunde navigera själv och undvika hinder. Trots att kartan fungerade ur ett tekniskt perspektiv så var den svårtydd och kunde förbättrats.

Nyckelord: Mekatronik, Omnidirektionell, Omnihjul, Trådlös styrning, Kartläggning

Acknowledgements

We would like to thank our supervisor during this thesis, Nihad Subasic, for the help and feedback during the process. We would also like to thank Staffan Qvarnström and the assistants at the laboratory, Amir Avdic and Malin Lundvall, for providing practical help with the building process, and a final thank you to our peers for providing opposition and feedback.

Axel Hedvall & Filip Rydén
Stockholm, May 2021

Contents

List of Figures

List of Tables

Glossary

holonomic Free movement along a plane. 3, 19

kiwi drive Three omni wheels in a triangle formation. 3, 19

omni wheel A wheel with rollers along the circumference. 3, 15, 19

omnidirectional All directions. 1–3, 19

Acronyms

ALU Arithmetic Logic Unit. 4

CPU Central Processing Unit. 3

CU Control Unit. 4

DC Direct Current. 4

EEPROM Electrical Erasable Programmable Read Only Memory. 4

FoV Field of View. 5, 8, 12, 20, 21

GPS Global Positioning System. 5

I/O Input/Output. 4, 7, 8

IC Integrated Circuit. 3

LED Light Emitting Diode. 4

LiDAR Light Detection and Ranging. 21

RAM Random Access Memory. 4, 7

VCC Voltage at Common Collector. 8

Wi-Fi Wireless Fidelity. 5, 8, 9, 12, 15

Chapter 1

Introduction

1.1 Background

Today, robots are used more and more in both industry and at home. The reason being that they are more expandable than us humans and in some cases can execute the task more efficiently than a human.

One application where expandable robots are used is to survey environments which are inaccessible or too dangerous for humans to enter. Robots are instead used to gather information about the environment so that a good plan of action can be made. This can be used to save victims stuck in a burning building, or victims after an earthquake. A robot can also be used to survey a dilapidated building. These robots need to have high mobility to adapt to an unknown environment.

Mobility can be defined as "the ability to move freely or be easily moved"[1]. A common solution for robots to achieve this is to use threads or differential steering and with that rotate on the spot to turn. If the need to turn is eliminated the mobility would increase significantly. This can be achieved by using omnidirectional wheels which allows free movement in any direction without turning, and instantaneous change of direction.

1.2 Purpose

The purpose of the project was to control a ground vehicle omnidirectionally, i.e. moves freely along the ground plane, avoid obstacles, and map the environment. Another goal was to transmit the map to the user who can command the robot in a specific direction. These three questions were formulated:

- *What is a simple yet robust configuration of omnidirectional wheels?*
- *How can the sensor data be represented as a map?*

- *What configuration of sensors gives an adequate perception of the environment?*

1.3 Scope

As this project was part of a Bachelor's thesis, the main constraints were time and budget. This led to the robot being constructed was a prototype to demonstrate the technology rather than a finished product. Due to the constraints some assumptions were made.

- *The space where the robot will operate is flat.*
- *Speed is not important.*
- *The wheels do not slip.*

1.4 Method

To answer the questions in section 1.2, a background study initiated the project. This study covered different types of omnidirectional wheels and different configurations of them. How to keep track of the robots position and what sensors to use were also of importance. After the study, a simulation was created in Acumen[2], which can be found in Appendix A. After the simulation, the components were chosen and the construction began.

Chapter 2

Theory

In this chapter, the theory needed for the construction of the prototype is presented.

2.1 Omni Wheels

An omni wheel is a specific type of omnidirectional wheel. omnidirectional wheels are a type of wheel that can move in all directions. Omni wheels have rollers along the circumference which allows movement in the opposite direction they are facing[3][4]. Omni wheels can be configured with different amount of wheels. A three wheel configuration is called kiwi drive. Three wheel drive is a cheaper option than a four wheel alternative since each wheel requires a motor and a motor driver.

2.1.1 Kiwi drive

A kiwi drive is a type of configuration with the least amount of omnidirectional wheels to achieve holonomic drive[5]. In a kiwi drive configuration, the wheels are placed in a circle 120° from each other with their axis pointing towards the central point of the circle. This configuration gives a set of equations that determines how much each wheel needs to move depending on the x and y translation, see Equation 2.1 to 2.3.

$$s_1 = -x \tag{2.1}$$

$$s_2 = 1/2 * x - \sqrt{3}/2 * y \tag{2.2}$$

$$s_3 = 1/2 * x + \sqrt{3}/2 * y \tag{2.3}$$

Where s_i is the steps for each motor.

2.2 Microcontroller

A microcontroller is a programmable Integrated Circuit (IC), it can be seen as a tiny computer; it has all the same core components. The component that performs the written instructions is the Central Processing Unit (CPU). The CPU consists

of two parts, the Arithmetic Logic Unit (ALU) which does the calculations, and the Control Unit (CU) which parses the instructions and feeds them to the ALU. A microcontroller has two types of memory, Electrical Erasable Programmable Read Only Memory (EEPROM) and Random Access Memory (RAM). The written instructions are saved in the EEPROM and it is preserved on a reboot. The RAM is where temporary data is saved. The RAM is erased on reboot.

Where a microcontroller differs from a personal computer is in the Input/Output (I/O) ports. The microcontroller uses the I/O ports to communicate with external components. Inputs are used to detect changes or receive sensor data. Outputs can be used to e.g. turn on an LED or control a motor[6].

2.3 Motors

Motors are used to drive wheels separately or to drive axles if the wheels are connected to an axle.

2.3.1 Stepper motor

A stepper motor is a type of DC motor where the output shaft can be rotated in discrete steps. The stepper motors have several internal coils grouped in phases. The stepping works by energizing the coils in a phase in a specific sequence. This allows very precise control since the number of steps per revolution commonly varies between 24 and 200, and a stepper motor can be driven one step at a time. This allows the motor to be driven anywhere from one step to several revolutions.

A drawback of a stepper motor compared to a conventional DC motor is that the stepper motor has nearly constant current draw, independent of the load, which leads to great inefficiencies and heat production[7].

The accurate control of a stepper motor allows the needed position by assuming that the robot drives to the calculated position. This open loop system is susceptible to drift, especially if the motors do not have the required torque to turn or the wheels slips[8]. This drift can build up over time.

2.3.2 Stepper driver

A stepper motor cannot be driven directly from the microcontroller since it requires a higher current to energize the coils than a microcontroller can deliver. This can be solved by using a motor driver. There are motor drivers specifically for stepper motors. These have the benefit of keeping track of where in the step sequence the stepper motors are. This means that they only need to receive a signal for step and another signal for direction. Stepper drivers can also do micro stepping, where more

than one phase is energised at a time to orientate the motor between two normal full steps. The micro stepping range may vary from half steps to 1/32 of a step[9].

2.4 Ultrasonic sensor

Ultrasonic sensors send out ultrasonic waves in a cone shape in front of them. The waves hit the object(s), provided there are any in the sensors Field of View (FoV), and reflect back to the sensor. Depending on how long time it takes for the wave to get back to the sensor, the signal from the sensor changes which can be used to measure distances[10]. Since sensors have a limited FoV they need to be spread out to get as much coverage as possible. Spreading them out also avoids having them interfere with each other[11][12].

2.5 External communication

To send data to an external computer the robot needs to be able to communicate with it. This can be done either by a wire going between the robot and the computer, or the robot sends all the data wirelessly to the computer. Wireless allows for more free movement.

Two ways of wireless communication were considered, Bluetooth and Wi-Fi, both with their advantages and disadvantages. Bluetooth requires less energy and is less prone to interference from the surroundings. The disadvantage of Bluetooth is the lower operating range and higher cost compared to Wi-Fi[13].

2.6 Position

Two main ways to achieve the robots position was studied. The first one is keeping track of the offset from an external frame of reference e.g. GPS[14]. Another way is to keep track of the robots offset from the start position by tracking the displacement of the robot.

There are several ways to implement tracking of the robots displacement. Some of them requires advanced sensors and sometimes some preparations in the environment[15]. It is also possible to do it in an easier way at the expense of accuracy and precision. Since the robot is driven, the position of the robot can be calculated based on the rotation of the motors[8][16].

Chapter 3

Demonstrator

This chapter will go into more detail about the chosen components and the construction. See Figure 3.1 for a render of how the finished robot looked.

3.1 Hardware

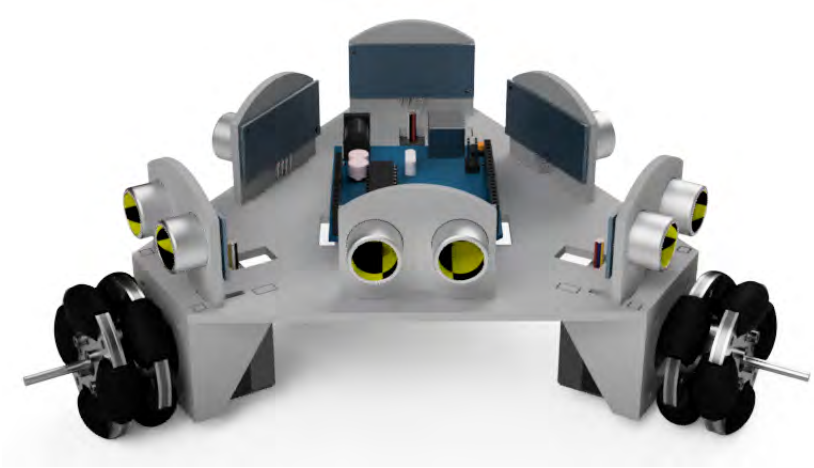


Figure 3.1. Render of the prototype in Autodesk Fusion 360[17]

3.1.1 Microcontroller

The microcontroller used was an Arduino UNO. The Arduino UNO is based on the ATmega328P microcontroller. The ATmega328P has 32 KB flash memory for programs and 2 KB RAM. The Arduino UNO has a recommended input voltage of 7 - 12 V and a total of 20 I/O pins. It can output both 3.3 V and 5 V. It has one hardware serial port and support for multiple software serial ports[18].

3.1.2 Stepper motor & driver

In order map the robots position it was decided to use the displacement of the robot. The best option for measuring displacement of the robot was to calculate how much the motors had rotated. Thus stepper motors were chosen as they do not require a closed loop for position[8].

For this project the Tamawaga TS3214N61 stepper motors were chosen. They have 200 steps per revolution which means an angle of 1.8° per step. The TS3214N61s can handle a maximum of 12 V and 0.25 A per phase.

The stepper drivers chosen for this projects were the DRV8825. The DRV8825 can supply a voltage between 8.2 V to 45 V and a maximum current of 2.2 A per phase. The current is continuously adjustable. The DRV8825 support full steps and five different micro step resolutions; 1/2-step, 1/4-step, 1/8-step, 1/16-step and 1/32-step[9].

3.1.3 Ultrasonic sensor

The ultrasonic sensor used were HY-SRF05 with a range of 0 - 4.5 m and a FoV of ±15°. The robot needed to see in all directions, this was done by placing a senor every 60° around the robot. The sensors have five I/O pins; ground, VCC, echo, trig, and out. VCC and ground were connected to the 5V and ground pins on the microcontroller respectively. To activate the senor the trig pin needs a signal for at least 10 µs. The response is sent back via the echo pin. All pins were connected to the microcontroller. The out pin is used for a different operating mode and was not used in this project. The ultrasonic sensors were used for obstacle avoidance and mapping.

3.1.4 Wi-Fi module

In order for the robot to communicate over Wi-Fi, a Wi-Fi module was required since the microcontroller does not have built in Wi-Fi. For this an ESP-01 ESP8266[19] Wi-Fi module was chosen. This particular Wi-Fi module was programmed using AT commands in order to get it connected to the network, and getting it to send and receive data. The ESP-01 operates on 3.3 V with no internal voltage regulator.

3.2 Electronics

The ultrasonic sensors' VCC, ground, and trig pins were all wired in parallel. All echo pins had individual connections to the microcontroller. This meant that all the sensors would trigger at the same time, but could be distinguished from each other with the individual echo connections.

The Wi-Fi module required 3.3 V and the Arduino could only provide a fixed 3.3 V output to power it, while the logic level of the Arduino was 5 V. This meant that when the Arduino sent data to the ESP-01 it could be damaged from the too high voltage. To solve this a voltage divider was created with a 1 kΩ and a 2 kΩ resistor.

A 100µF capacitor was used over the positive and ground power for the stepper drivers to protect them against the inrush current when power was first supplied.

To power the robot a battery was used. This battery was connected to the stepper drivers and the Arduino's internal voltage regulator. The battery was a three cell lithium battery with a voltage of 11.1 V.

See Table 3.1 for a list of the components used and Figure 3.2 for a full wiring diagram.

Table 3.1. Components used

Á	Component	Description
1	Microcontroller	Arduino UNO
3	Stepper Motors	Tamawaga TS3214N61
3	Stepper drivers	DRV8825
3	Omni wheels	Diameter 48 mm
6	Ultrasonic sensors	HY-SRF05
1	Wi-Fi module	ESP-01

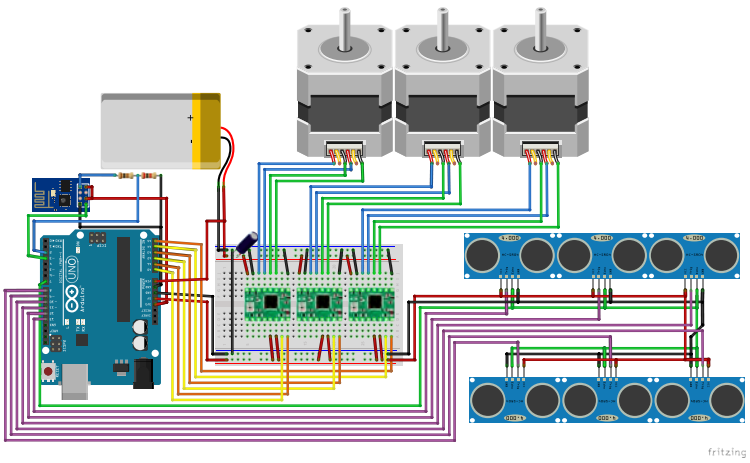


Figure 3.2. Wiring Diagram created in Fritzing[20]

3.3 Software

Two different programs needed to be created. One for the Arduino to control the robot, and one for the server to communicate with the robot and interact with the user.

3.3.1 Arduino

An overarching flowchart was created to have a clear vision for the program, see Figure 3.3.

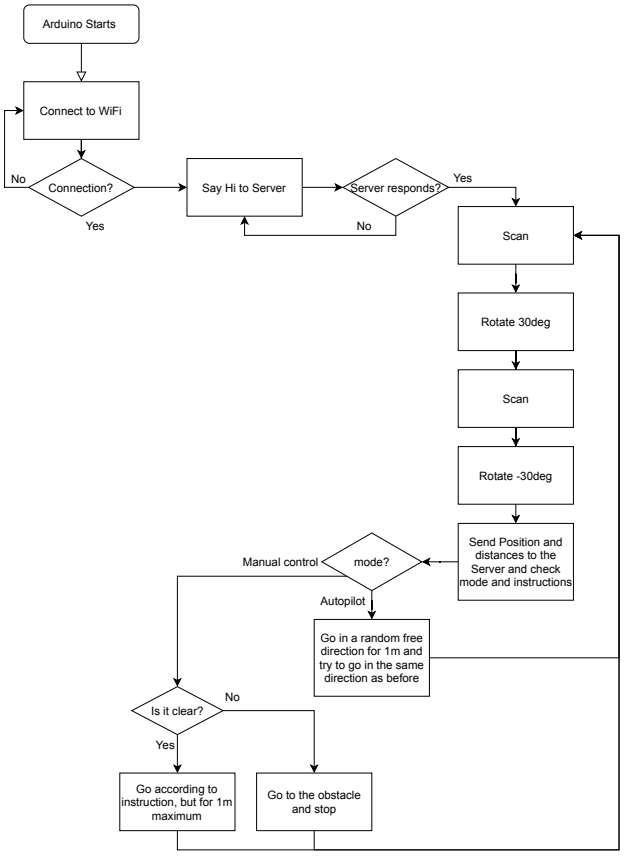


Figure 3.3. Flowchart of the program, created with diagrams.net[21]

The robot had two modes; manual control and autpilot, see subsection 3.3.2 Modes. The two modes ran in the same loop that started by first scanning the environment and sending the data to the server which responded with the current settings.

Scanning

Since the ultrasonic sensors requires precise timing to work, only one ultrasonic sensor could be used at a time. To achieve this, the sensors were cycled one after another. This also prevented the sensors from interfering with each other. However, the sensors only covered a total of 180° around the robot. To get the full 360°, all the sensors scanned once, then the robot rotated 30°, scanned once more and rotated back to the initial rotation. Rotating the robot and scanning again also helped with a problem where the angel from the sensor to the wall was too flat[22]. Sensor data was then sent to the server along with the current position. A library was used to simplify the communication to the server[23].

3.3.2 Modes

The robot had two modes, manual control for controlling the robot manually from the control panel, and autopilot which allowed the robot to move freely on its own.

Manual Control

In this mode the robot is controlled via the control panel on the website. A direction is chosen on the control panel and sent to the robot to perform. If a direction was not chosen the robot would stand still and wait until a new direction is chosen. Before the robot starts driving in the chosen direction, it scans and checks if the path is clear. If the path is clear the robot will drive one meter in the chosen direction. However, should there be an obstacle in the way, the robot will drive up to the obstacle and stop.

Autopilot

The autopilot mode allows the robot to move on its own with no target or goal. The robot will go in a random direction until it detects an object blocking its path in which case it will choose a new direction to travel in. A new direction is chosen based on sensor data. The robot checks all sensor data starting with the first sensor, the one facing 90° in the robots own coordinate system. When it finds a suitable direction, no obstacle closer than half a metre, it starts driving in that direction.

The full Arduino code can be found in Appendix B.

3.3.3 Displaying

To display the map it was determined that the easiest and fastest way to implement it would be to send the sensor data from the robot to a computer. Then process the data on the computer and display it as a map. Since it was already decided that the wireless transmission would be over Wi-Fi, the displaying of the map would be on a website running on the receiving computer.

Since the ultrasonic sensors detected obstacles in a cone shape the map had limited resolution. However, the map had adequate resolution to get an understanding of the environment.

3.3.4 Server

The server ran Node.JS[24] with Express[25] to act as an web server and Socket.io[26] to communicate in real-time. The robot communicated with the server via http GET requests and the control panel used socket. The map was generated by JavaScript in the client's browser, see Figure 3.4. The server code is available in Appendix C.

Node.JS

The Node.JS server kept track of the connected robot and control panels. When the server received a GET request from the robot, it parsed the request and sent it to the connected control panels. The server also responded with the current settings so the robot could act accordingly.

Control Panel

The control panel displayed the map, a communication log, and provided control of the robot. The control panel had a drop down list to select which mode the robot should use and in the case of manual control, arrow keys to select direction, see Figure 3.4. The map was a canvas element that was updated by JavaScript when the server sent new data. Since the sensors were placed horizontally the map was a top down view of where the robot had been. The sensor data was drawn as arcs to represent the cone shape of the ultrasonic sensors' FoV. The control panel code is available in Appendix D.

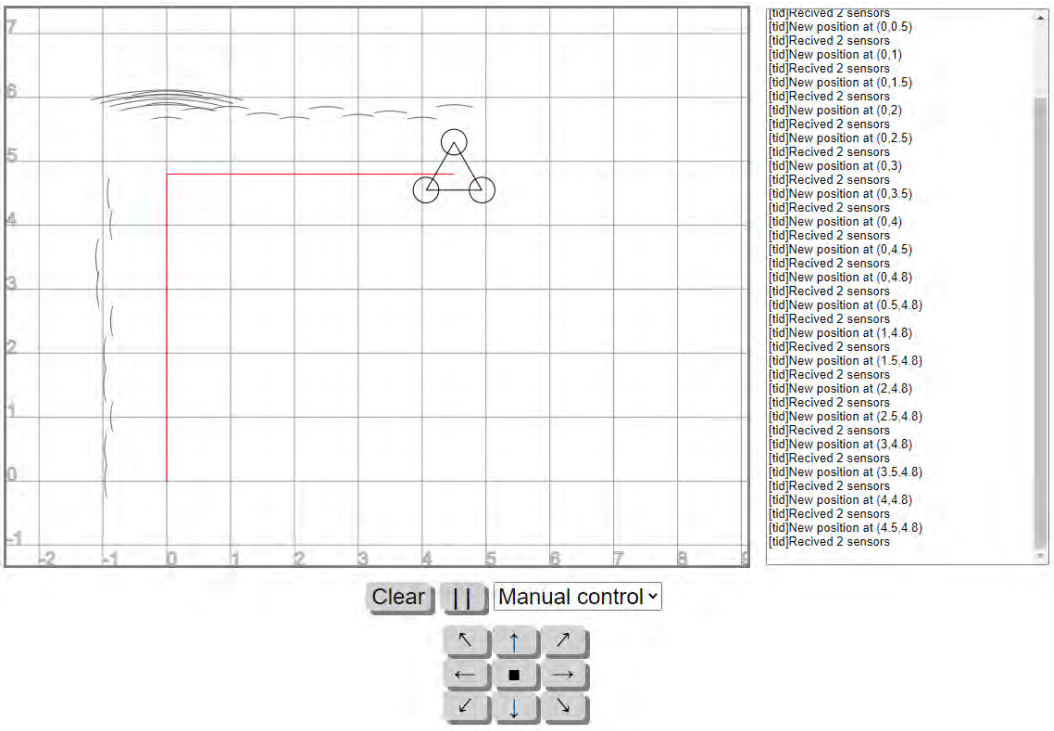


Figure 3.4. How the control panel looks. Screenshot from our web page in Google Chrome[27]

Chapter 4

Tests and Results

The tests performed on the final prototype is described in this chapter.

4.1 Communication Test

A communication test was performed to test the two way communication over Wi-Fi. To test the communication, the robot and a laptop were connected to a Wi-Fi hotspot. When they both were connected, the robot sent a http request to the laptop which then responded. First the robot sent an empty request, which the laptop received and responded to. The response was received by the robot without any problems. When the robot sent data in the request, the server received it and responded, however the the robot did not receive the response and timed-out instead. This problem was found to be a limitation of the Arduino library used and could therefor not be solved. A workaround was created by first sending en empty request to receive data to the robot, and after that send a request with data and waiting for the time-out.

4.2 Manual Control Tests

Three tests were preformed to test the precision and accuracy of the drive.

4.2.1 Test 1

The first test of the manual control system was a success. The robot and server connected to each other without any issues and all commands issued via the control panel was performed by the robot. One issue was that the robot drifted a bit from the expected result. The expected result was that it would return to its original position given that the directions given were back, left, forward, and right. This however, did not happen, and the robot ended up drifting more than half a metre from the starting location. Some drift was expected since the weight was not equally distributed and all omni wheels were not the same. It does not explain all the drift

that was observed and a better result should be possible with some tweaking of the code.

4.2.2 Test 2

In the second round of testing baseline measurements were taken by making the robot drive in four directions; left, right, up, and down, three times. The baseline measurements were used to see how much the robot drifted and how consistent it was in its movements.

After the baseline measurements were completed the code was changed to make the robot move much faster, and the timing of the stepper motors more even. This yielded much better results and an overall lower deviation of about 78%, see Table 4.1. However, the drift was still very noticeable in the cases when the stepper motors took different amount of steps. This was most notably when the robot drove left or right, where one motor took more steps than the other two.

4.2.3 Test 3

A third test was performed to test if an implementation to the code would minimize the remaining drift. In this test, the motor(s) with fewest amount of steps, had their steps multiplied with a constant. When the constant had a value of 1.2, i.e. the motor(s) with fewest amount of steps took 20% more steps, the best result was achieved. With this new change the robot achieved the same accuracy when it drove to the left/right as up/down.

The numerical results of Test 2 and Test 3 can be seen in Table 4.1.

Table 4.1. Distance tests

Goal	**Actual**			**Deviation**		
	Baseline	**Test 2**	**Test 3**	**Baseline**	**Test 2**	**Test 3**
Right: 0°, 1m	305°, 72 cm	350°, 89 cm	1°, 99 cm	83 cm	19 cm	2 cm
	300°, 70 cm	352°, 88 cm	358°, 98 cm	88 cm	17 cm	4 cm
	302°, 74 cm	355°, 92 cm	359°, 101 cm	87 cm	11 cm	2 cm
Up: 90°, 1m	89°, 95 cm	90°, 97 cm	89°, 100 cm	5 cm	3 cm	2 cm
	91°, 1 m	89°, 99 cm	90°, 98 cm	2 cm	2 cm	2 cm
	88°, 1 m	89°, 98 cm	90°, 99 cm	3 cm	3 cm	1 cm
Left: 180°, 1 m	235°, 75 cm	191°, 85 cm	178°, 99 cm	83 cm	23 cm	3 cm
	230°, 74 cm	189°, 88 cm	182°, 101 cm	77 cm	19 cm	4 cm
	237°, 71 cm	190°, 86 cm	179°, 99 cm	86 cm	21 cm	2 cm
Down: 270°, 1 m	272°, 98 cm	271°, 101 cm	270°, 98 cm	4 cm	2 cm	2 cm
	271°, 98 cm	270°, 98 cm	269°, 99 cm	3 cm	2 cm	2 cm
	268°, 99 cm	269°, 99 cm	271°, 98 cm	3 cm	2 cm	3 cm

4.3 Sensor Test

A sensor test was performed to see if the senors were able to detect objects and if the robot sent the sensor data to the server. An obstacle was placed at specific distances from the robot and five measurements were made for every distance. For this test, it was decided that the resolution of one centimetre was enough. All the sensors performed well with some small errors and could detect objects within a three metre range very reliably. See Table 4.2 for the average deviations for the test.

Table 4.2. Sensor tests

Distance	**avg. deviation**
2 cm	0 cm
5 cm	0 cm
10 cm	0 cm
20 cm	0 cm
50 cm	±1 cm
100 cm	-1 cm
150 cm	-2 cm
200 cm	-2 cm
250 cm	-2 cm
300 cm	-1 cm

On the server side it worked perfectly. All sensor data was collected and sent from the robot to the server, where it was deciphered and drawn out on the map.

4.4 Autopilot Test

The autopilot was tested both in a corridor and in a room with chairs and tables. The robot managed to avoid the walls and furniture and draw a map with the sensor data. The map did not give a good image of the rooms, however, since the robot's path was drawn it could be used to see a rough shape of the rooms. See Figure 4.1 for the map after the test in the corridor.

An attempt to solve this was made by finding out which arcs overlapped each other and then drawing them in a darker shade to imply probability of an obstacle. This was implemented by dividing up every arc in pieces and see if they matched any other piece. This was a very inefficient way of doing it and caused the whole control panel to freeze. To avoid having the control panel freeze, the code checking for overlaps was moved to a thread in the background of the web browser. This improved the performance a bit but it was still quite unusable. The code for the separate thread can be found in Appendix E.

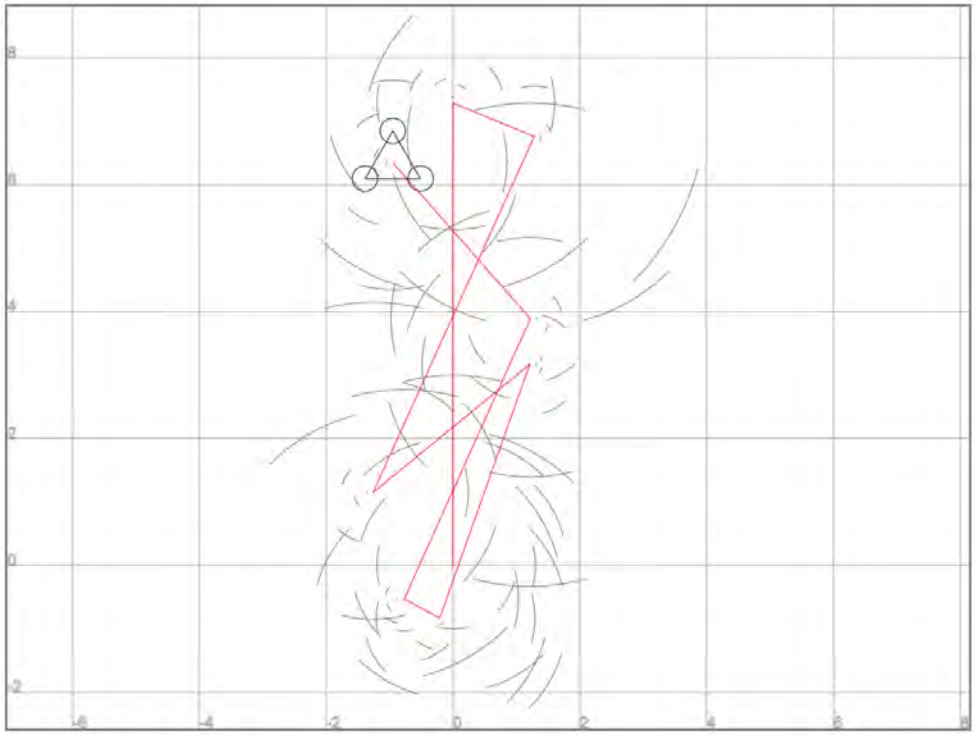


Figure 4.1. Autopilot test. Screenshot from our web page in Google Chrome[27]

Chapter 5

Discussion and conclusions

There were two major flaws with the current setup that arose during the tests. The first one was the result from the communication test where the request timed-out when the robot sent data to the server. This made the robot unresponsive when it periodically froze to receive data. If this problem would be solved, the robot would most likely act a lot smoother. It would move smoother since an empty request took a fraction of the time a sub-optimal time-out request did.

The other flaw was from the autopilot test where the map that was drawn was not that usable as a map. An attempt to solve this was made in the autopilot test by indicating the probability of an obstacle actually being there. However, there were still some arcs being drawn where there was not any obstacles. It was those arcs that made the map unusable since the user then had to figure out which arcs were real and which were not.

With those flaws in mind, the result was still impressive for a prototype, it showed that with some more tweaks the robot would work as initially intended.

To answer the three formulated questions in section 1.2, we begin with the first one:

- *What is a simple yet robust configuration of omnidirectional wheels?*

In this project, a kiwi drive was used, this means the robot had three omni wheels and three motors. This is the least amount of wheels required since three points of contact is needed for something to not fall over without active balancing. Theoretically a holonomic drive can be achieved with only two motors since the ground plane consists of two orthogonal directions. This would require a complex drive system since the motors would have to apply their torque at the center of the robot to avoid rotations. Hence the configuration is simple. With the manual tests we have shown that the configuration is robust since the robot repeatedly drove both accurate and precise.

Let us consider the second question next:

- *How can the sensor data be represented as a map?*

We have shown that the data from the sensors could be sent from the robot and received on a server. After that the data could be represented as a working map displayed on a web page. The issue was that the senors gave out data in the form of distances. This distance could have come from an object anywhere in the sensors FoV which means that when the map was drawn we had to take that into consideration. That is why, when the map was drawn, the sensors data were represented with arcs. This gave us a better view of how the room looked and a more accurate representation of what the sensors saw, since an object could have been anywhere in that arc. However, this resulted in a cluttered map which was hard to read after a while. Some alterations were tested but did not perform well.

Lastly, consider the third question:

- *What configuration of sensors gives an adequate perception of the environment?*

The robot needed to avoid obstacles in every direction which meant that the sensors needed to cover 360°. This would require the robot to have 12 sensors to be able to see 360° at all times. However, this would not be possible without getting a bigger microcontroller since every sensor needed one port on the controller and with all the other equipment there was not enough room. Thus six senors were deemed to be enough and the robot would instead rotate 30° and scan again to get the full coverage, see section 3.3.1 Scanning.

Chapter 6

Further work

The problem that arose in the communication test can be solved by programming custom firmware for the ESP-01. This could also solve a problem where the robot has to wait for a response when it sends a request to the server. With custom firmware, the ESP-01 can handle the requests by it self, and the Arduino can prompt the request, and retrieve the response later.

While the map worked to get an idea of what the room looked like it became very cluttered with all the sensors' data and some of the data was false data i.e it was not actually an obstacle there. A solution to this was tested but deemed to be too slow and ineffective, see section 4.4 Autopilot Test. Another solution can be to first send the data to another program that can calculate and remove some of the data that is false through geometry.

To further improve the mapping, better senors can be used so that false data would be less prevalent. While the ultrasonic sensors were adequate they had their issues, see section 3.3.1 Scanning. With more sophisticated senors, like Light Detection and Ranging (LiDAR), those issues can be mitigated. A LiDAR also has a much narrower FoV than an ultrasonic sensor since a LiDAR uses a thin laser beam. This makes it much more accurate and less prone to detecting something that is not in the robot's path. LiDAR also comes with the benefit of being much faster than an ultrasonic senor since it moves at the speed of light instead of the much slower speed of sound.

Bibliography

[1] Cambridge dictionary. (n.d.) *mobility*. Cambridge dictionary. Retrieved February 11, 2021, from https://dictionary.cambridge.org/dictionary/english/mobility

[2] Acumen. (n.d.) *Acumen*. Acumen Language. Retrieved February 1, 2021, from http://www.acumen-language.org/

[3] NORMELIUS, A., & BECKMAN, K. (2020) *Hand Gesture Controlled Omnidirectional Vehicle* (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-279822

[4] BJÖRKLUND, F., & STRAND, C. (2019) *Omnidirectional pong playing robot: Pong playing robot using kiwi drive and a PID controller* (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-264456

[5] SWART, D. (2014) *R/C Omniwheel Robot*. Make:. Retrieved February 11, 2021, from https://makezine.com/projects/make-40/kiwi/

[6] GUDINO, M. (2018) *Introduction to Microcontrollers*. Arrow. Retrieved February 12, 2021, from https://www.arrow.com/en/research-and-events/articles/engineering-basics-what-is-a-microcontroller

[7] EARL, B. (2020) *All About Stepper Motors*. Adafruit. Retrieved February 11, 2021, from https://cdn-learn.adafruit.com/downloads/pdf/all-about-stepper-motors.pdf

[8] ANDERSSON, P., & KUGELBERG, E. (2018) *Home Assistant Navigation - Smart Optical and Laser Orientation: H.A.N.S.O.L.O* (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-233139

[9] TEXAS INSTRUMENTS. (2014) *DRV8825 Stepper Motor Controller IC datasheet Rev. F*. Texas Instruments. Retrieved February 11, 2021, from https://www.ti.com/lit/ds/symlink/drv8825.pdf

[10] Andria, G., Attivissimo, F., & Giaquinto, N. (2001) Digital signal processing techniques for accurate ultrasonic sensor measurement. *Measurement, 30(1)*, 105-114. https://doi.org/10.1016/S0263-2241(00)00059-2

[11] HALTORP, E., & BREDHE, J. (2020) *ODAR: Obstacle Detecting Autonomous Robot* (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-279834

[12] OTTOSON, J., & RENSTRÖM, N. (2019) *aMAZEing robot: A method for automatic maze solving* (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-264491

[13] SATTLE, S. (n.d.) *WiFi vs. Bluetooth: Wireless Electronic Basics*. Autodesk. Retrieved February 13, 2021, from https://www.autodesk.com/products/eagle/blog/wifi-vs-bluetooth-wireless-electronics-basics/

[14] Garmin Ltd. (n.d.) *ABOUT GPS*. Garmin. Retrieved February 11, 2021, from https://www.garmin.com/sv-SE/aboutGPS/

[15] XinReality. (n.d.) *Inside-out tracking*. XinReality. Retrieved February 13, 2021, from https://xinreality.com/wiki/Inside-out_tracking

[16] ANTONOVA, A., & LUNDIN, H. (2019) *Photobot: An Exploring Robo* (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-264452

[17] Autodesk. (n.d) *Fusion 360*. Autodesk fusion Retrieved May 9, 2021, from https://www.autodesk.com/products/fusion-360/overview

[18] Arduino. (n.d.) *Arduino UNO REV3*. Arduino store. Retrieved April 10, 2021, from https://store.arduino.cc/arduino-uno-rev3

[19] ESPRESSIF SYSTEMS. (2020) *ESP8266 AT Instruction Set v3.0.3*. Espressif. Retrieved February 11, 2021, from https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf

[20] Fritzing. (n.d.) *Fritzing*. Fritzing. Retrieved May 8, 2021, from https://fritzing.org/

[21] diagrams.net. (n.d) *diagrams.net*. Retrieved May 8, 2021, from https://www.diagrams.net/

[22] NAGAI, I., NIWA,K., & WATANABE, K. (2017) *A detection method using ultrasonic sensors for avoiding a wall collision of Quadrotors* Retrieved from https://ieeexplore.ieee.org/abstract/document/8016028

[23] bportaluri. (2019) *WiFiEsp*. GitHub. Retrieved March 21, 2021, from https://github.com/bportaluri/WiFiEsp

[24] Node.js (n.d.) *About*. Node.js. Retrieved April 7, 2021, from `https://nodejs.org/en/about/`

[25] Express (n.d.) *Express*. Express. Retrieved April 7, 2021, from `https://expressjs.com/`

[26] Socket.IO (n.d.) *Get started*. socket.io. Retrieved April 7, 2021, from `https://socket.io/get-started/`

[27] Google Chrome (n.d) *Google Chrome*. Chrome Retrieved April 7, 2021, from `https://www.google.com/chrome/`

Appendix A

Acumen Simulation

```
// Bachelor's Thesis in mechatronics 2021 at KTH Royal Institute of
↪  Technology
// TRITA-ITM-EX 2021:29
// MF133X Group 11
// Axel Hedvall, Filip Rydén
// 2021-02-28
// Task:
//     Simulate the robot in a 3d environment.
// Notes:
//     In this simulation the platform is represented as a circle.
//     In reality it's probably going to be a triangle.

model Main(simulator) =
initially
  y = 0,  //distance in y-direction
  y' = 1, //speed in y-direction
_3D = ()

always
 y' = 1, //speed in y-direction, +y in all geometric shapes' y
 ↪  coordinate to simulate movement

  _3D = ((Cylinder                // Platform
           center = (0,y,0)       // Where the platform is in the
           ↪  room
           radius = 1             // radius of platform
           length = 0.1           // platform thickness
           color = yellow         // platform colour
           rotation = (pi/2,0,0)  // platform rotation
           )
```

```
(Cylinder                  //wheel
center = (0,1.05+y,-0.15) //where the wheel is
radius = 0.2               //radius of wheel
length = 0.1               //wheel ''width''
color = black              //wheel colour
rotation = (0,0,0)         //rotation
)
(Box                        //motor
center = (0,0.875+y,-0.15) //placement, right next to
↪  motor
size = (0.15,0.25,0.15)    //size of motor
color = green              //motor colour
rotation = (0,0,0)         //rotation
 )
 (Box                      //motor controller
 center = (0,0.6+y,-0.15)  //placement, right next to
 ↪  motor
 size = (0.25,0.3,0.3)     //size of motor controller
 color = red               //motor controller colour
 rotation = (0,0,0)        //rotation
   )

(Cylinder                  //wheel
center = (0.92,-0.5+y,-0.15)
radius = 0.2
length = 0.1
color = black
rotation = (0,y,pi/3)      //rotation around y
                           //to simulate movement
 )
 (Box                      //motor
center = (0.76,-0.4375+y,-0.15)
size = (0.15,0.25,0.15)
color = green
rotation = (0,0,pi/3)
 )
 (Box                      //motor controller
 center = (0.52,-0.3+y,-0.15)
 size = (0.25,0.3,0.3)
 color = red
 rotation = (0,0,pi/3)
   )
```

```
(Cylinder                    //wheel
center = (-0.92,-0.5+y,-0.15)
radius = 0.2
length = 0.1
color = black
rotation = (0,y,-pi/3)    //rotation around y
                          //to simulate movement
 )
 (Box                     //motor
center = (-0.76,-0.4375+y,-0.15)
size = (0.15,0.25,0.15)
color = green
rotation = (0,0,-pi/3)
 )
 (Box                     //motor controller
 center = (-0.52,-0.3+y,-0.15)
 size = (0.25,0.3,0.3)
 color = red
 rotation = (0,0,-pi/3)
   )
)
```

Appendix B

Arduino code

```
// Bachelor's Thesis in mechatronics 2021 at KTH Royal Institute of
↪  Technology
// TRITA-ITM-EX 2021:29
// MF133X Group 11
// Axel Hedvall, Filip Rydén
// 2021-05-09
// Task:
//    Control a robot with three omniwheels
//    Connect to a server via Wi-Fi and perform recived instructions
// Hardware:
//    1 Arduino UNO
//    1 ESP-01
//    3 Omni wheels
//    3 Stepper drivers, DRV8825
//    3 Stepper motors, Tamawaga TS3214N61
//    6 Ultrasonic sensors, HY-SRF05

#include <WiFiEsp.h> //Ads the WiFiEsp Library

// Define constants
#define SQRT32 0.866025403784439    // sqrt(3)/2
#define DEG2RAD 0.017453292519943
#define RADIUS 0.024
#define DIST2STEP 1326.291192432461 // 1/(2*r*pi)*steps
#define stepsPerRevolution 200
#define ANGLE2STEP 117.0256934499231 // 2*pi*R * dist2step
#define STEPSPEED 1500 // Microseconds to wait between steps

// Use hardware serial if available use software else
#ifndef HAVE_HWSERIAL1
```

```
#include "SoftwareSerial.h"
SoftwareSerial Serial1(3, 2); // RX, TX
#endif

// Which state the robot is in
enum State {
  scanning, driving, sending, reciving
};

// Define stepper motor connections
const int dirPin[]  = {A1, A3, A5};
const int stepPin[] = {A0, A2, A4};

// Define sonic sensor connections
const int sensors[] = {13, 12, 11, 10, 9, 8};
const int trig = 7;

float s[] = {0, 0, 0, 0}; // Steps each motor will take

int status = WL_IDLE_STATUS; // The Wifi radio's status
char server[] = "192.168.43.220"; // The IP of the server
char ssid[] = "WiFi";
char pass[] = "password";

// Initialize the client object
WiFiEspClient client;

float distances[12]; // The distances measured
float posX, posY; // The position of the robot

int mode = 0; // The current mode
String instructions = ""; // The current instructions
State state; // The current state

void setup() {
  // Setting the pins
  for (int i = 0; i < 6; i++) {
    pinMode(sensors[i], INPUT);
  }
  pinMode(trig, OUTPUT);
  for (int i = 0; i < 3; i++) {
    pinMode(dirPin[i], OUTPUT);
    pinMode(stepPin[i], OUTPUT);
  }
```

```
  // Start HW serial
  Serial.begin(115200);

  // initialize serial for ESP module
  Serial1.begin(9600);
  // initialize ESP module
  WiFi.init(&Serial1);

  // Check for the presence of the module
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi module not present");

    // Don't continue
    while (true);
  }

  // Connect to the WiFi
  connectWiFi(ssid, pass);
  // Print the wifi status
  printWifiStatus();

  // Start with scannning
  state = scanning;
}

// The angle the robot tries to drive in autopilot
unsigned int prevAngle = 90;
String response = ""; // The http request response
void loop() {
  Serial.println(state);
  switch (state) {
    case scanning: {
      scan(); // Scan the sonic sensors
      state = reciving; // Next state
      httpRequest2(); // Request settings from server
      break;
    }
    case reciving: {
      // Read the response from the ESP
      while (client.available()) {
        char c = client.read();
        response += c;
      }
```

```
  // If we have the "}" the whole message has been recived
  if (response.indexOf("}") == -1) {
    Serial.print(response);
  } else {
    Serial.println(response);
    // Takes out the json data
    response = response.substring(response.indexOf('{'),
    ↪  response.lastIndexOf('}') + 1);

    // Parses the json data
    mode = response.substring(response.indexOf(':')+1,
    ↪  response.indexOf(',')).toInt();
    instructions =
    ↪  response.substring(response.lastIndexOf(':')+2,
    ↪  response.indexOf('}')-1);
    // Prints the formated recived data
    Serial.print(mode);
    Serial.print(",");
    Serial.println(instructions);

    response = ""; // Resets the response for next time
    httpRequest(); // Sends position and sensor data to server
    state = sending; // Next state
  }
  break;
}
case sending: {
  // if there's incoming data from the net connection send it
  ↪  out the serial port
  // this is for debugging purposes only
  while (client.available()) {
    Serial.print(client.read());
  }

  // Go to next state if the ESP has disconnected
  if (!client.connected()) {
    state = driving; // Next state
    // Empty the serial buffer
    while (client.available()) {
      Serial.print(client.read());
    }
    delay(500); // Waits 500ms for the ESP to really disconnect
  }
```

```
  break;
}
case driving: {
  Serial.print(mode);
  Serial.print(",");
  Serial.println(instructions);
  // If we are in manual control
  if (mode == 0) {
    // If we have instructions
    if (instructions == "N" || instructions == "E" ||
    ↪  instructions == "S" || instructions == "W" ||
    ↪  instructions == "NW" || instructions == "NE" ||
    ↪  instructions == "SW" || instructions == "SE") {
      if (instructions == "N") { // 90
        drive(90, getDistance(90));
      } else if (instructions == "S") { // 270
        drive(270, getDistance(270));
      } else if (instructions == "E") { // 0
        drive(0, getDistance(0));
      } else if (instructions == "W") { // 180
        drive(180, getDistance(180));
      } else if (instructions == "NW") { // 135
        drive(135, getDistance(135));
      } else if (instructions == "NE") { // 45
        drive(45, getDistance(45));
      } else if (instructions == "SW") { // 225
        drive(255, getDistance(225));
      } else if (instructions == "SE") { // 315
        drive(315, getDistance(315));
      }
    }
  } else if (mode == 1) { // If we are in autopilot
    int firstAngle = prevAngle; // A tmp version of prevAngle
    while(firstAngle == prevAngle) { // While the robot is
    ↪  driving in the same direction
      int index = ((450 - prevAngle)/ 30) % 12; // The index for
      ↪  the sensor towards prevAngle
      float dist = distances[index]; // The saved distance in
      ↪  that direction
      // Find a clear direction by checking the three closest
      ↪  distances
      while (dist > 0 && dist < 0.15 || (distances[(index+1) %
      ↪  12] < 0.1 || distances[(index+11) % 12] < 0.1)) {
        prevAngle += 60; // Check +60deg
```

```
            if(prevAngle > 360) {
              prevAngle -= 360;
            }
            index = ((450 - prevAngle)/ 30) % 12; // New index
            dist = distances[index]; // and distance
          }
          // If we have a new direcion
          if (firstAngle != prevAngle) {
            state = scanning;
            return;
          }
          if (dist > 0) { // If we can see anything, stop at it
            drive(prevAngle, dist - 0.1);
          } else { // else drive 4 meters
            drive(prevAngle, 4);
          }
          // Scan to refresh the distances
          if(firstAngle == prevAngle) {
            scan();
          }
        }
      }
      state = scanning; // Next state
      break;
    }
  }
}

// This function calculates the safe distance to drive in the
↪  specified direction
float getDistance(int angle) {
  int index = ((450 - angle)/ 30) % 12; // The index for the sensor
  ↪  towards the direction
  float dist = distances[index]; // The saved distance in that
  ↪  direction
  // If the direction is clear return 1 meter
  if ((dist == 0 || dist > 1) && ((distances[(index+1) % 12] == 0 ||
  ↪  distances[(index+1) % 12] > 0.5) && (distances[(index+11) %
  ↪  12] == 0 || distances[(index+11) % 12] > 0.5))) {
    return 1;
  } else if (dist > 0.2 & (distances[(index+1) % 12] > 0.2 &&
  ↪  distances[(index+11) % 12] > 0.2)) {
    // If it is less than one meter to an obsticle, drive to it
    return dist - 0.1;
```

```
  } else { // If it's unsafe to drive at all
    return 0;
  }
}

// Tihs function performes the saved steps
void takeSteps() {
  // Calculate the most number of steps
  float _max = max(max(s[0], s[1]), s[2]);
  // Increase the steps with less steps than max
  for(int i = 0; i < 3; i++) {
    if(s[i] < _max) {
      s[i] *= 1.2;
    }
  }
  // How often ewery stepper should take a step
  int ffs[] = {(int)round(_max/s[0]), (int)round(_max/s[1]),
  ↪  (int)round(_max/s[2])};
  // For every step
  for(int steg = 0; steg < _max; steg++) {
    // Wich steppers should take a step
    bool b[] = {steg % ffs[0] == 0, steg % ffs[1] == 0, steg %
    ↪  ffs[2] == 0};
    // For every stepper
    for(int motor = 0; motor < 3; motor++) {
      // If the stepper no. j should take a step
      if(b[motor]){
        digitalWrite(stepPin[motor], HIGH);
      }
    }
    delayMicroseconds(STEPSPEED);
    // For every stepper
    for(int motor = 0; motor < 3; motor++) {
      // If the stepper no. j should take a step
      if(b[motor]){
        digitalWrite(stepPin[motor], LOW);
      }
    }
    delayMicroseconds(STEPSPEED);
  }
  // Reset the saved steps
  s[0] = 0;
  s[1] = 0;
  s[2] = 0;
```

```
}

// This function activates the sonic sensors one after another
void scan() {
  // Scan every sensor
  for (int i = 0; i < 6; i++) {
    // Turn off trig if would be on
    digitalWrite(trig, LOW);
    delayMicroseconds(2);
    // Send trig signal for 10us
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);
    // Messures the lenght of recived signal
    long duration = pulseIn(sensors[i], HIGH);
    // Calculates the distance in meters
    distances[i*2] = (duration / 2.0) / 2910.0;
    // If the distance is wrong set it to 0
    if (distances[i*2] > 4.5) {
      distances[i*2] = 0;
    }
    // Wait before the next sensor
    delay(100);
  }

  // Rotate
  rotate(-30);
  // Scan every sensor again
  for (int i = 0; i < 6; i++) {
    // Turn off trig if would be on
    digitalWrite(trig, LOW);
    delayMicroseconds(2);
    // Send trig signal for 10us
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);
    // Messures the lenght of recived signal
    long duration = pulseIn(sensors[i], HIGH);
    // Calculates the distance in meters
    distances[i*2+1] = (duration / 2.0) / 2910.0;
    // If the distance is wrong set it to 0
    if (distances[i*2+1] > 4.5) {
      distances[i*2+1] = 0;
    }
```

```
    // Wait before the next sensor
    delay(100);
  }
  // Rotate back
  rotate(31);
}

// This function calculates the steps for every motor to drive a
// specific distance in a specific angle
void drive(float angle, float distance) {
  angle = angle * DEG2RAD; // Degrees to radians

  // The robots x and y position after the driving
  posX += cos(angle) * distance;
  posY += sin(angle) * distance;

  // Calculates the x and y distance
  float x = cos(angle) * distance;
  float y = sin(angle) * distance;

  // Calculates the distance for every wheel
  float d1 = -x;
  float d2 = 0.5 * x - SQRT32 * y;
  float d3 = 0.5 * x + SQRT32 * y;

  // Calculates the steps for every motor
  s[0] = d1 * DIST2STEP * 1.0;
  s[1] = d2 * DIST2STEP * 1.0;
  s[2] = d3 * DIST2STEP * 1.0;

  // Prints the calculated steps
  for (int j = 0; j < 3; j++) {
    Serial.print("Target");
    Serial.print(j);
    Serial.print(": ");
    Serial.println(s[j]);
    if (s[j] < 0) {
      digitalWrite(dirPin[j], HIGH);
      s[j] = -s[j];
    } else {
      digitalWrite(dirPin[j], LOW);
    }
  }
  takeSteps();
```

```
}

// This function rotates the robot a specified degrees
void rotate(float angle) {
  angle = angle * DEG2RAD; // Degrees to radians

  s[0] = angle * ANGLE2STEP;
  s[1] = angle * ANGLE2STEP;
  s[2] = angle * ANGLE2STEP;

  // Prints the calculated steps
  for (int j = 0; j < 3; j++) {
    Serial.print("Target");
    Serial.print(j);
    Serial.print(": ");
    Serial.println(s[j]);
    if (s[j] < 0) {
      digitalWrite(dirPin[j], HIGH);
      s[j] = -s[j];
    } else {
      digitalWrite(dirPin[j], LOW);
    }
  }

  takeSteps();
}

// This function connects the ESP to a WiFi network
void connectWiFi(char ssid[], char pass[]) {
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network
    status = WiFi.begin(ssid, pass);
  }
  // we're connected now, so print out the data
  Serial.println("You're connected to the network");
}

// This function prints out information of the current WiFi
↪  connection
void printWifiStatus() {
  // print the SSID of the network we're attached to
  Serial.print("SSID: ");
```

```
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength
  long rssi = WiFi.RSSI();
  Serial.print("Signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}

// This function sends the current position and sensor data to the
↪  server
void httpRequest() {
  delay(100);
  Serial.println();

  // close any connection before send a new request
  // this will free the socket on the WiFi module
  client.stop();

  // if there's a successful connection
  if (client.connect(server, 3000)) {
    Serial.println("Connecting...");

    String content = "position=" + String(posX) + "," +
    ↪  String(posY);
    content.concat("&sensor=90," + String(distances[0]));
    content.concat("&sensor=60," + String(distances[1]));
    content.concat("&sensor=30," + String(distances[2]));
    content.concat("&sensor=0," + String(distances[3]));
    content.concat("&sensor=330," + String(distances[4]));
    content.concat("&sensor=300," + String(distances[5]));
    content.concat("&sensor=270," + String(distances[6]));
    content.concat("&sensor=240," + String(distances[7]));
    content.concat("&sensor=210," + String(distances[8]));
    content.concat("&sensor=180," + String(distances[9]));
    content.concat("&sensor=150," + String(distances[10]));
    content.concat("&sensor=120," + String(distances[11]));
    Serial.println("->" + String(content.length()));
    // send the HTTP GET request
```

```
    client.println(F("GET /robot HTTP/1.1"));
    client.println(("Host: " + String(server) + ":3000"));
    client.println("Accept: */*");
    client.println("Content-Length: " + String(content.length()));
    client.println("Content-Type:
    ↪  application/x-www-form-urlencoded");
    client.println("Connection: close");
    client.println();
    client.println(content);
  } else {
    // if you couldn't make a connection
    Serial.println("Connection failed");
  }
}

// This function requests te settings from the server
void httpRequest2() {
  Serial.println();

  // close any connection before send a new request
  // this will free the socket on the WiFi shield
  client.stop();

  // Try to connect three times
  for(int i = 0; i < 3; i++) {
    // if there's a successful connection
    if (client.connect(server, 3000)) {
      Serial.println("Connecting...");

      // send the HTTP GET request
      client.println(F("GET /robot HTTP/1.1"));
      client.println(("Host: " + String(server) + ":3000"));
      client.println("Connection: close");
      client.println();
      return;
    } else {
      // if you couldn't make a connection
      Serial.println("Connection failed");
      Serial.println("Trying " + String(2 - i) + " more times");
      client.stop();
      delay(100*(i+1));
    }
  }
  // if we fail three times go to next state
```

```
  httpRequest(); // Sends position and sensor data to server
  state = sending; // Next state
}
```

Appendix C

Node.JS code

```
// Bachelor's Thesis in mechatronics 2021 at KTH Royal Institute of
↪  Technology
// TRITA-ITM-EX 2021:29
// MF133X Group 11
// Axel Hedvall, Filip Rydén
// 2021-05-09
// Task:
//    Comunicate with the robot and the control panel
// Hardware:
//    A computer with node.js, express and socket.io

const express = require('express');
const app = express();
const http = require('http').createServer(app);
const io = require('socket.io')(http);

var robots = []; // The connected robots
var controllers = []; // The connected controllers
var robot = {mode: 0, instructions: null}; // An object for the
↪  robot

app.use(express.static('public')); // Serves the javascript files
app.use(express.json()) // for parsing application/json
app.use(express.urlencoded({ extended: true })) // for parsing
↪  application/x-www-form-urlencoded

// GET request from /robot
app.get('/robot', (req, res) => {
    var obj = { mode: robot.mode, instructions: robot.instructions
    ↪  }; // JSON obj to send
```

```
    res.contentType('application/json'); // It's a JSON response
    res.status(200); // Everything ok
    res.json(obj); // Send the JSON obj back

    Robot(req); // parse the request
});

app.post('/robot', (req, res) => {
    var obj = { mode: robot.mode, instructions: robot.instructions
    ↪  }; // JSON obj to send

    Robot(req); // parse the request

    res.contentType('application/json'); // It's a JSON response
    res.status(200); // Everything ok
    res.json(obj); // Send the JSON obj back
});

// function for parsing the request and send the data to the
↪  connected controllers
function Robot(req) {
    // If it's a new robot
    if (!robots.includes(req.ip)) {
        robots.push(req.ip);
        // Send "Robot connected" to all connected controllers
        controllers.forEach(element => {
            element.emit('message', "Robot connected");
        });
    }

    // If the request contains position data
    if(req.body.position || req.query.position) {
        var pos = req.body.position ? req.body.position :
        ↪  req.query.position;
        pos = pos.split(',');
        var data = {position:{x: pos[0]*1, y: pos[1]*1}}; // Create
        ↪  the JSON data
        // Send the JSON data to all connected controllers
        controllers.forEach(element => {
            element.emit('position', data);
        });
    }
    // If the request contains sensor data
```

```
    if(req.body.sensor || req.query.sensor) {
        var data = {sensors:[]}; // Create the JSON data
        if (typeof(req.body.sensor) == "string") { // if there is
        ↪  one or more sensors
            var sens = req.body.sensor.split(',');
            // Append to the JSON data
            data.sensors.push({angle:sens[0]*1,
            ↪  distance:sens[1]*1});
        } else {
            req.body.sensor.forEach(element => {
                var sens = element.split(',');
                // Append to the JSON data
                data.sensors.push({angle:sens[0]*1,
                ↪  distance:sens[1]*1});
            });
        }
        // Send the JSON data to all connected controllers
        controllers.forEach(element => {
            element.emit('sensor', data);
        });

    }
}

// Send the control.html to the client
app.get('/control', (req, res) => {
    res.sendFile(__dirname + '/control.html');
});

// Start socket.io connection
io.on('connection', (socket) => {
    console.log("A client connected: " + socket.id);
    // The client requests a controller registartion
    socket.on('register', (data) => {
        if(data.id == socket.id) {
            console.log(data.role + " connected");
            socket.emit('message', 'Registration Ok');
            // Append the new controller
            if (data.role === "control")
                controllers.push(socket);
        }
    });
    // A controller has requested to change mode
    socket.on("setmode", (mode) => {
```

```
        if (controllers.includes(socket)) {
            console.log("Mode: " + mode);
            socket.emit('message', 'Changed to ' + mode);
            robot.mode = mode * 1; // Save the new mode
        }
    });
    // A controller has requested the current mode
    socket.on("getmode", () => {
        if (controllers.includes(socket)) {
            socket.emit('mode', robot.mode); // Send the mode to the
            ↪  controller
        }
    });
    // A controller has sent new instructions
    socket.on('setinstruction', (instruction) => {
        // Save the new instructions, null if it's stop
        robot.instructions = instruction == "Stop" ? null :
        ↪  instruction;
    });
    // A controller has disconnected
    socket.on('disconnect', () => {
        controllers = controllers.filter(function(value, index,
        ↪  array) {
            return value != socket.id;
        });
        console.log('Client disconnected');
    });
});

// Start a webserver on port 3000
http.listen(3000, () => {
    console.log('listening on *:3000');
});
```

Appendix D

control.html

```
<!--
Bachelor's Thesis in mechatronics 2021 at KTH Royal Institute of
↪  Technology
TRITA-ITM-EX 2021:29
MF133X Group 11
Axel Hedvall, Filip Rydén
2021-05-09
Task:
   Display the control panel and send instructions via socket.io to
   ↪  the server
Hardware:
   A computer with a web browser
-->
<!DOCTYPE html>
<html>
  <head>
    <title>Robot control</title>
    <style>
        * {
          font-family: sans-serif;
        }

        .flex {
          display: flex;
          flex-direction: row;
          justify-content: center;
        }
        #output {
          resize: none;
          align-self: stretch;
```

```
  flex-basis: 300px;
  cursor: unset;
}
#output:focus {
  outline: none;
}
canvas {
  margin: 0 auto;
  display: inline-block;
}

button {
  font-size: 1.5rem;
  border-radius: 5px;
  background-color: lightgray;
  box-shadow: 5px 5px 0 0 gray;
  border: unset;
  cursor: pointer;
  text-align: center;
  min-width: 2em;
}
button:active {
  box-shadow: none;
  transform: translate(5px, 5px);
  border: 1px solid gray;
}
button:focus {
  outline: none;
}

#toolbar {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
}
#toolbar > * {
  margin-right: 10px;
}

#dpad {
  margin-top: 1em;
  display: inline-grid;
  grid-template-rows: auto auto;
  grid-template-columns: auto auto auto;
```

```
        justify-content: center;
        gap: 5px;
        width: 100vw;
      }

      select {
        font-size: 1.5rem;
      }
  </style>
</head>
<body>
  <div class="flex">
    <canvas width="800" height="600" style="outline: solid
    ↪  gray;"></canvas>
    <textarea id="output" readonly></textarea>
  </div>
  <br>
  <div id="toolbar">
    <button onclick="drawer.Clear()">Clear</button>
    <button><strong>| |</strong></button>
    <select onchange="SetMode(event, this)" id="modeselect">
      <option value="0">Manual control</option>
      <option value="2">Tesla autopilot</option>
      <option value="1">Roomba mode</option>
    </select>

    <div id="dpad">
      <button
      ↪  onmousedown="SetInstruction('NW')">&nwarr;</button><button
      ↪  onmousedown="SetInstruction('N')">&uarr;</button><button
      ↪  onmousedown="SetInstruction('NE')">&nearr;</button>
      <button
      ↪  onmousedown="SetInstruction('W')">&larr;</button><button
      ↪  onmousedown="SetInstruction('Stop')">&#9632;</button><button
      ↪  onmousedown="SetInstruction('E')">&rarr;</button>
      <button
      ↪  onmousedown="SetInstruction('SW')">&swarr;</button><button
      ↪  onmousedown="SetInstruction('S')">&darr;</button><button
      ↪  onmousedown="SetInstruction('SE')">&searr;</button>
    </div>
  </div>
  <script src="/socket.io/socket.io.js"></script>
  <script src="/js/2D-Draw.js"></script>
  <script src="/js/Shapes.js"></script>
```

```
<script src="/js/Logger.js"></script>
<script>
  // Initalize socket.io
  var socket = io();

  // On connection register as a controller
  socket.on("connect", () => {
    console.log(socket.id);
    LOGGER.log("Connected with id " + socket.id);
    LOGGER.log("Register as controller");
    socket.emit("register", {id: socket.id, role: "control"});

    socket.emit("getmode"); // Ask for the current mode
  });

  // A new mode recived
  socket.on("mode", (mode) => {
    robot.mode = mode; // Save the new mode
    document.getElementById("modeselect").value = mode; // Set
    ↪  the select to the new mode
    LOGGER.log("Current mode: " +
    ↪  document.getElementById("modeselect").selectedOptions[0].innerText);
  });

  // A generic message recived
  socket.on("message", function(msg) {
    console.log(msg);
    LOGGER.log(msg);
  });

  // A new position recived
  socket.on('position', function(msg) {
    console.log(msg);
    LOGGER.log("New position at (" + msg.position.x + "," +
    ↪  msg.position.y + ")");
    robot.pos = {x: msg.position.x, y: msg.position.y};
  });

  // New sensors recived
  socket.on('sensor', function(msg) {
    console.log(msg);
    LOGGER.log("Recived " + msg.sensors.length + " sensors");
    // Add the new sensors
    msg.sensors.forEach(element => {
```

```
      if(element.distance > 0) {
        robot.AddSensor(element.distance, element.angle);
      }
    });
  });

  socket.on('reset', function() {
    drawer = new
    ↪  Drawer(document.getElementsByTagName("canvas")[0]);
    drawer.isGridEnabled=true;
  });
</script>
<script>
  const LOGGER = new Logger(document.getElementById("output"));
  var drawer = new
  ↪  Drawer(document.getElementsByTagName("canvas")[0]);
  drawer.isGridEnabled=true; // Enable the grid

  var path = new Path(0, 0, "#FF0000", 1); // A path for the
  ↪  robot
  drawer.Add(path);

  // Draw the robot
  robot1 = new Compound(0,0,new Polygon(0,0,3,0.5,"#000000"),new
  ↪  Circle(0.5,0,0.2,"#000000"),new
  ↪  Circle(-0.25,0.44,0.2,"#000000"),new
  ↪  Circle(-0.25,-0.44,0.2,"#000000"));
  robot1.Rotate(90);
  drawer.Add(robot1);

  // An obj for the robot
  var robot = {
    xpos: 0,
    ypos: 0,
    mode: 0,
    obstacles: [],
    set pos(p) {
      this.SetPos(p.x, p.y);
    },
    get pos() {
      return {
        x: xpos,
        y: ypos,
      };
```

```
  },
  SetPos: function(x, y) {
    this.xpos = x;
    this.ypos = y;

    path.AddNode(x, y); // Adds the new position to the path
    robot1.SetPosition(x, y); // Moves the robot to the new
    ↪ position
  },
  AddSensor: function(distance, angle) {
    // Creates a object for the new sensor data
    obst = {position:{x:this.xpos, y:this.ypos},
    ↪ radius:distance, angle:[(angle-15)*Math.PI/180,
    ↪ (angle+15)*Math.PI/180], shapes:[]};

   var myWorker = new Worker('js/worker.js'); // Start a new
   ↪ thread
    myWorker.addEventListener('message', function(e) { // Set
    ↪ response handler from thread
      robot.obstacles[e.data].shapes[0].color = "#000000"; //
      ↪ Change the color
      drawer.Draw(); // Redraw
    }, false);
    var tmp = []; // New list of obstecles
    // Add only the essensial information from the obsticles
    this.obstacles.forEach(element => {
      tmp.push({position:element.position,
      ↪ radius:element.radius, angle:element.angle});
    });
    myWorker.postMessage([obst, tmp, this.obstacles.length]);
    ↪ // Start the thread

    // Creates an arc for the sensordata
    shape = new Arc(this.xpos, this.ypos, distance, angle-15,
    ↪ angle+15, "#777777", -1);
    obst.shapes.push(shape);
    this.obstacles.push(obst);
    // Draws the new arc
    drawer.Add(shape);
  }
}

// Events for the arrow keys press
window.onkeydown = function(evnt) {
```

```
  if(evnt.repeat || robot.mode != 0) { // Ignore if the key is
  ↪  repeated or not in manual control
    return;
  }
  // Sets the corresponding instruction
  switch (evnt.code) {
    case "ArrowUp":
      SetInstruction("N");
      break;
    case "ArrowLeft":
      SetInstruction("W");
      break;
    case "ArrowDown":
      SetInstruction("S");
      break;
    case "ArrowRight":
      SetInstruction("E");
      break;
    default:
      return;
  }
};
// Events for the arrow keys release
window.onkeyup = function(evnt) {
  if(evnt.repeat || robot.mode != 0) { // Ignore if the key is
  ↪  repeated or not in manual control
    return;
  }
  // Sets the stop instruction
  switch (evnt.code) {
    case "ArrowUp":
    case "ArrowLeft":
    case "ArrowDown":
    case "ArrowRight":
      SetInstruction("Stop");
      break;
    default:
      return;
  }
};

// Sends the instruction the the server
function SetInstruction(inst) {
  LOGGER.log("Sending " + inst);
```

```
        socket.emit("setinstruction", inst);
      }

      // Sends the new mode to the server
      function SetMode(evnt, elmt) {
        // Shows the dpad if manual control
        document.getElementById("dpad").style.display = elmt.value
        ↪  == 0 ? "inline-grid" : "none";

        console.log(elmt.value);
        robot.mode = elmt.value; // Save the new mode
        LOGGER.log("Sending " + elmt.selectedOptions[0].innerText);
        socket.emit("setmode", elmt.value); // Send to the server
      }
    </script>
  </body>
</html>
```

Appendix E

worker.js

```
// Bachelor's Thesis in mechatronics 2021 at KTH Royal Institute of
↪  Technology
// TRITA-ITM-EX 2021:29
// MF133X Group 11
// Axel Hedvall, Filip Rydén
// 2021-05-09
// Task:
//    Calculate which arcs are crossing, in a background thread
// Hardware:
//    A computer with a web browser

self.addEventListener('message', function(e) {
    const r = 1000; // N.o. pieces
    const obst = e.data[0];
    var obstacles = e.data[1];
    const ind = e.data[2];
    // Remove arcs that cannot cross
    for(var i = 0; i < obstacles.length; i++) {
        const element = obstacles[i];
        const dx = obst.position.x - element.position.x;
        const dy = obst.position.y - element.position.y;
        const sqrD = dx*dx+dy*dy;
        if(sqrD > (obst.radius + element.radius)*(obst.radius +
        ↪  element.radius)) {
            // If they are too far apart
            obstacles[i] = undefined;
```

```
        } else if (Math.max(element.radius, obst.radius) >
        ↪  Math.sqrt(sqrD) + Math.min(element.radius, obst.radius)
        ↪  || Math.max(element.radius, obst.radius) <
        ↪  Math.sqrt(sqrD) - Math.min(element.radius, obst.radius))
        ↪  {
            // If they are inside of eachother
            obstacles[i] = undefined;
        }
    }

    for(var i = 0; i < r; i++) { // Divide the new arc into pieces
        var x = obst.position.x + obst.radius *
        ↪  Math.cos(obst.angle[0] +
        ↪  (obst.angle[1]-obst.angle[0])*(i/r));
        var y = obst.position.y + obst.radius *
        ↪  Math.sin(obst.angle[0] +
        ↪  (obst.angle[1]-obst.angle[0])*(i/r));
        for(var k = 0; k < obstacles.length; k++) { // For every
        ↪  saved arc
            if (obstacles[k] == undefined) { // If theh cannot cross
                continue;
            }
            const elmt = obstacles[k];
            for(var j =0; j < r; j++) { // Devide into pieces
                var x1 = elmt.position.x + elmt.radius *
                ↪  Math.cos(elmt.angle[0] +
                ↪  (elmt.angle[1]-elmt.angle[0])*(i/r));
                var y1 = elmt.position.y + elmt.radius *
                ↪  Math.sin(elmt.angle[0] +
                ↪  (elmt.angle[1]-elmt.angle[0])*(i/r));
                if(Math.abs(x - x1) < 0.1 && Math.abs(y - y1) < 0.1)
                ↪  { // If the pieces are close enough
                    // Send a hit to the main thread
                    self.postMessage(k);
                    self.postMessage(ind);
                }
            }
        }
    }
    self.close();
}, false);
```

TRITA ITM-EX 2021:29