# Team name: Eastern Invaders

**Members:**

Adam Zilizi

Alexander Mitkov Dimitrov

Ali Akeel Ibrahim Mochlif Al-Benaian

Kion Valther Christensen

# Project 2 - The Robocar Project

Project Report

Business Academy Aarhus ITEK-E19a
2019.12.13. 1st semester project

# Table of contents

# Introduction

For our first semester project our assignment was to build a robotic car that is able to follow a black line, detect and push out boxes from an arena and have a web interface through which it can be remotely controlled. The car was to be controlled by a Raspberry Pi using Python programs and an HTTP web interface. All the motors, sensors cables and body parts were provided to us by our teachers, but it was not forbidden to add our own improvements.

The teams for this first project were picked randomly. We have filled out and signed a group contract (Appendix A), which we were supposed to follow during the duration of this project.

This report is about how we chose the project management type, planned the project according to it, designed and built our robotic car from the parts that we received, setup the Raspberry Pi and finally created three software modules to meet the functionality requirements.

The main purpose of this project was to test our skills and knowledge that we have obtained during the first semester. It explored how we were able to cooperate and organize our work in a slim time period. For many of us it has also showed our deficiencies and made us realize that teamwork is a process where communication and simple enthusiasm has vital importance for the ultimate success of the project.

# 1 Project planning and management

The first part of this report is about how we chose a project management type and our reasoning behind the decision. It is then followed by our activities creating a project plan, breaking down tasks, estimating required time for the tasks, assigning responsibilities and creating a testing plan.

During the whole duration of the project we were also supposed to use Trello.

## 1.1 Selecting Project management type

When we discussed what type of project management we should follow during this project, the decision was very simple. Due to the short duration of the project and notoriety of our team members not attending classes, we have agreed that following an agile method like SCRUM would bring no benefit.

The project is too short and in SCRUM terms it would only fall within a single sprint. Many aspects of SCRUM, like delivering value early, having no definite end-state, holding demos with presentable products at the end of sprints, could not be applied.

Another deciding factor was regarding how SCRUM teams function – there are clearly set sprint days which have an organized structure. It is important for team members to show up, perform a Daily standup meeting and start sprinting. Our team had members which usually did not show up on daily classes, were late, or promised to show up but failed to do so.

Fortunately, we already know about this team dynamic before the start of the project and could choose the right project management type – waterfall.

## 1.2 Opportunity identification

### 1.2.1 Hierarchy of objectives

Mainly based on our previous project management report, we have defined a hierarchy of objectives. This time we have expanded on our project's deliverables and objectives.

**Purpose of project**

- Apply all the skills that we have gathered during our first semester studies by creating a robotic car
- Improve our practical skills and gather more experience
- To successfully finish the first semester

**The project's product**

- The product is a robotic car that is controlled by a Raspberry Pi using Python programs and an HTTP web interface

**The project's deliverables**

- A robotic car that is able to follow a line and do that as fast as possible
- The car being able to push boxes out of a circle
- A web interface that is able to control the car

**Project objectives**

- The robotic car meeting all required specifications – being able to finish the "race track" and being among the fastest cars.
- The car being able to push all the boxes out of the arena without any problems
- The web interface being fast and responsive

Creating this hierarchy helped us to get a clear picture of the most basic objectives of our project.

### 1.2.2 Rich picture

The rich picture has remained unchanged since our last report.

In our drawing we tried to capture all the relevant entities (people, objects) and visualise the processes. Between the individual people working on the project we have created relationships and visualised possible conflicts. We have also captured the cycle of coding and testing, which played an important role during the development of the project.

After the creation of the rich picture, the team members had an image of the processes and relationships between the different entities of the project. This understanding helped us with the creation of the work breakdown structure.

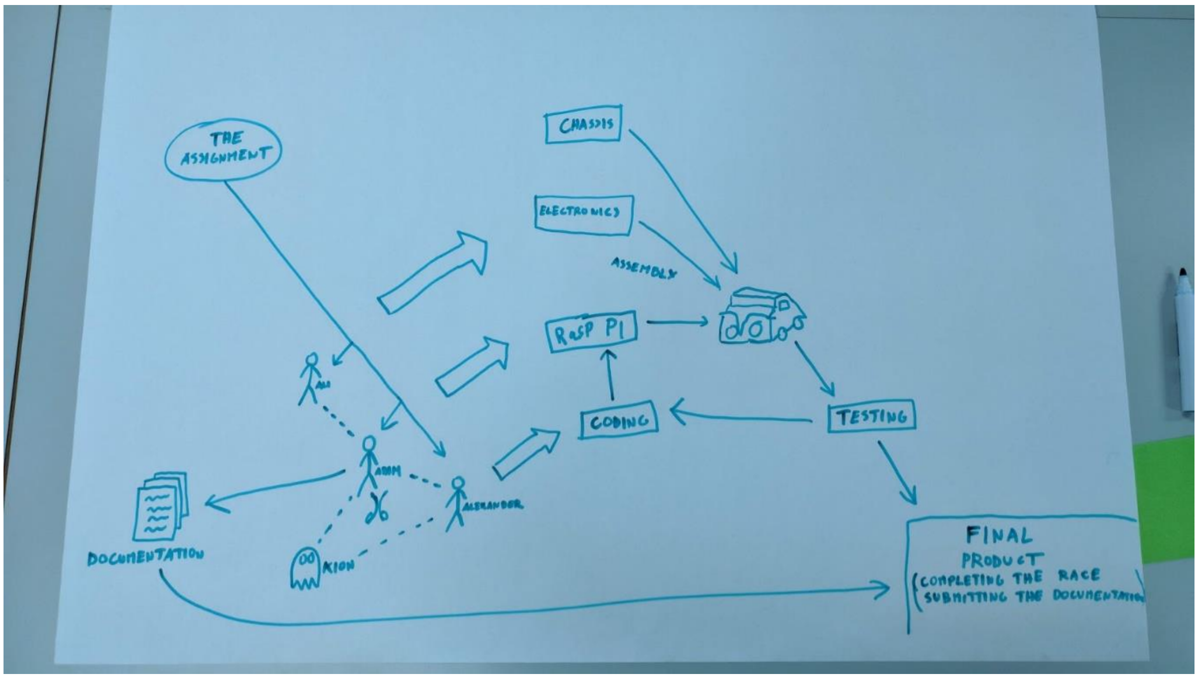


*Figure 1.2.1 – Rich picture*

### 1.2.3 WBS – work breakdown structure

The purpose of the work breakdown structure is to give an overview of everything that needs to be done, while also putting these tasks in chronological order.

We have reused the WBS that we created for the previous project management report. The coding part has been further expanded on, due to now knowing the exact three software modules which had to be programmed.

Our WBS showed that we had to assemble prototypes, thoroughly test them, determine what was going to work and what was not. After the prototyping was done, then the whole car could be built and tested – both hardware and software-wise.

Once we had a working robot we could finish the documentation and create a little movie. In the end there was nothing left other than showing off the robot at the presentation and submitting the documentation.

The product of this activity was not only a breakdown of the tasks, but also their chronological order, which we fully utilized when creating a list of activities.

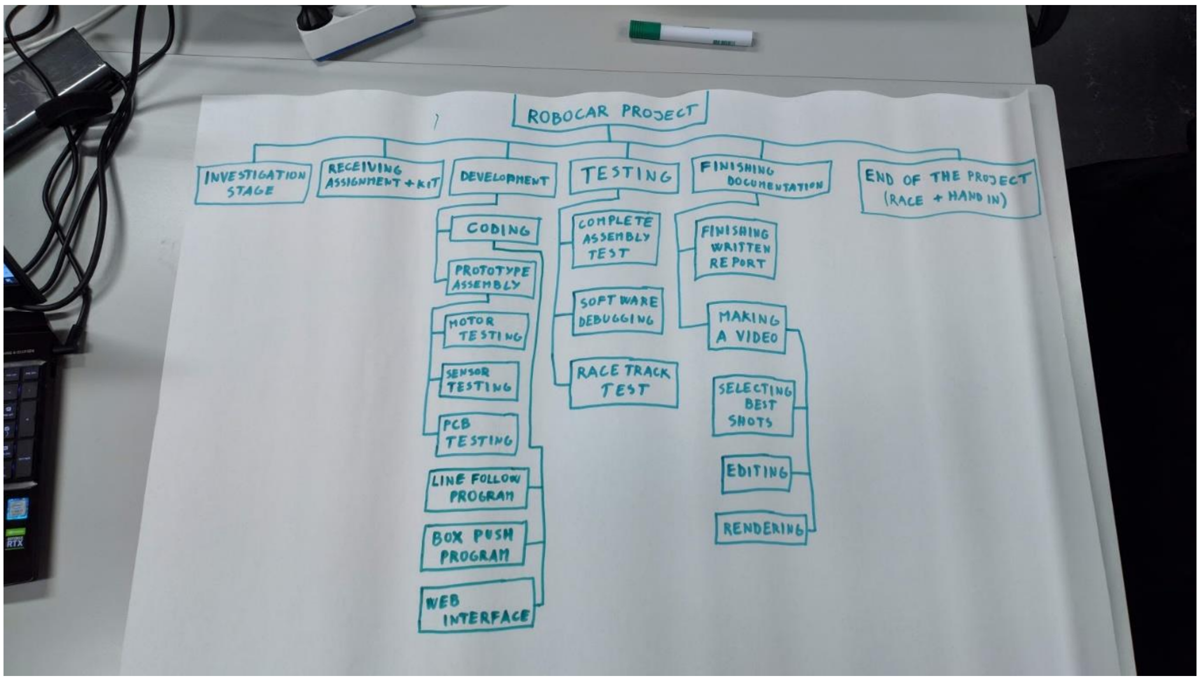


*Figure 1.2.2 – WBS – Work Breakdown Structure*

## 1.3 Project Time Management

Now that we knew the exact time frame for the project, we could rework our time estimations so that they would be more realistic.

In this stage of the project planning, we have focused on creating a list of activities, making logical relations between them, estimating the duration of each activity and performing successive calculations on the riskiest one. After obtaining all this information our objective was to create a critical path.

### 1.3.1 Time Estimation

First, using our WBS, we have created a list of all activities.

After that we assigned logical dependencies to each of them. These logical dependencies meant that certain tasks had to be completed before work could be started on the next task. This type of a logical relationship between tasks is known as “FS” (finish to start).

Lastly, we have estimated how long each activity would take. We made these estimations with the full project timeframe in mind.

| Number | Activity | Logical relationship | Duration (hours) |
|---|---|---|---|
| 1 | Investigation stage | | 3 |
| 2 | Receiving assignment | 1 | 0,5 |
| 3 | Coding | 2 | 10 |
| 4 | Prototype assembly | 2 | 4 |
| 5 | Testing | 3, 4 | 8 |
| 6 | Finishing written report | 5 | 6 |
| 7 | Making a video | 5 | 3 |
| 8 | End of project | 5, 6, 7 | 0,5 |

### 1.3.2 Successive calculations

Some of the estimated duration values are not as easy to predict as others – some have a higher uncertainty. That is the case in every project. After a discussion we have selected two of the most uncertain activities: coding and testing. We put them in a table, assigned pessimistic ($V_P$), most probable ($V_S$) and optimistic values ($V_O$):

| Activity | $V_P$ | $V_S$ | $V_O$ |
|---|---|---|---|
| Coding | 16 | 10 | 8 |
| Testing | 12 | 8 | 6 |

The following were the calculations of three values – mean value ($V_M$), standard deviation ($S_V$) and variance ($S_V^2$):

**Coding:**

Mean value: $V_M = \frac{V_P+3*V_S+V_O}{5} = \frac{16+3*10+8}{5} = 10{,}8$ hours

Standard deviation: $S_V = \frac{V_P-V_O}{5} = \frac{16-8}{5} = 1{,}6$

Variance: $S_V^2 = 1{,}6^2 = 2{,}56$

**Testing:**

Mean value: $V_M = \frac{V_P+3*V_S+V_O}{5} = \frac{12+3*8+6}{5} = 8{,}4$ hours

Standard deviation: $S_V = \frac{V_P-V_O}{5} = \frac{12-6}{5} = 1{,}2$

Variance: $S_V^2 = 1{,}2^2 = 1{,}44$

From these results we could see that the variance of coding was higher than the variance of testing. This meant that testing had a higher uncertainly and thus we had to break it down into smaller pieces, calculate the mean values of those smaller pieces and add them up again. The result was a more accurate estimation of the duration of this activity.

We broke coding down into three activities (utilizing our WBS for assistance):

- Line follow program
- Box push program
- Web interface

We have created the following table:

| **Activity** | **V$_P$** | **V$_S$** | **V$_O$** |
|---|---|---|---|
| Line follow program | 6 | 4 | 3 |
| Box push program | 4 | 3 | 2 |
| Web interface | 5 | 4 | 3 |

and calculated the following mean values:

Line follow program mean value: $V_{M1} = \frac{V_P+3*V_S+V_O}{5} = \frac{6+3*4+3}{5} = 4{,}2$ hours

Box push program mean value: $V_{M2} = \frac{V_P+3*V_S+V_O}{5} = \frac{4+3*3+2}{5} = 3$ hours

Web interface mean value: $V_{M3} = \frac{V_P+3*V_S+V_O}{5} = \frac{5+3*4+3}{5} = 4$ hours

The total mean value was $V_M = V_{M1} + V_{M2} + V_{M3} = 4{,}2 + 3 + 4 = 11{,}2$ hours

Now we could update the table with this value.

### 1.3.3 Assigning responsibilities / Final form of the list

With successive calculations done, we had a somewhat reliable time estimation for the project. It was time to assign responsibilities to the individual team members. The reason we had not assigned responsibilities yet, was that without logical dependencies and time estimations we might have put too much load on certain members and too little load on others. Another unfavourable situation would have been that one person would be responsible for two activities that ran in parallel – that would have put too much unnecessary pressure on him and produce unsatisfactory results.

| Number | Activity | Logical relationship | Duration (hours) | Responsibility |
|---|---|---|---|---|
| 1 | Investigation stage | | 3 | Ali |
| 2 | Receiving assignment | 1 | 0,5 | Kion |
| 3 | Coding | 2 | 11,2 | Adam |
| 4 | Prototype assembly | 2 | 4 | Alexander |
| 5 | Testing | 3, 4 | 8 | Kion |
| 6 | Finishing written report | 5 | 6 | Ali |
| 7 | Making a video | 5 | 3 | Alexander |
| 8 | End of project | 5, 6, 7 | 0,5 | Kion |

Workload on individual members:

| | |
|---|---|
| Adam: | 11,2 hours |
| Alexander: | 7 hours |
| Ali: | 9 hours |
| Kion: | 9 hours |

We have managed to split the workload almost evenly between the group members, this was supposed to prevent either of the members from being overworked.

### 1.3.4 Critical path

The very last step of time management was to create a critical path. Usually projects are not fully linear, some activities are running in parallel, this means that there are multiple paths from “Start” to “Finish”. One of these paths is called the critical path, and it determines the minimum time it will take to perform the project. If just one of the tasks on the critical path is delayed, the entire project will be delayed. That is why it is important to determine what the critical path is.

We had drawn the critical path and thus learned which activities must not have been delayed, and which ones had some slack – the ones that could have been delayed.

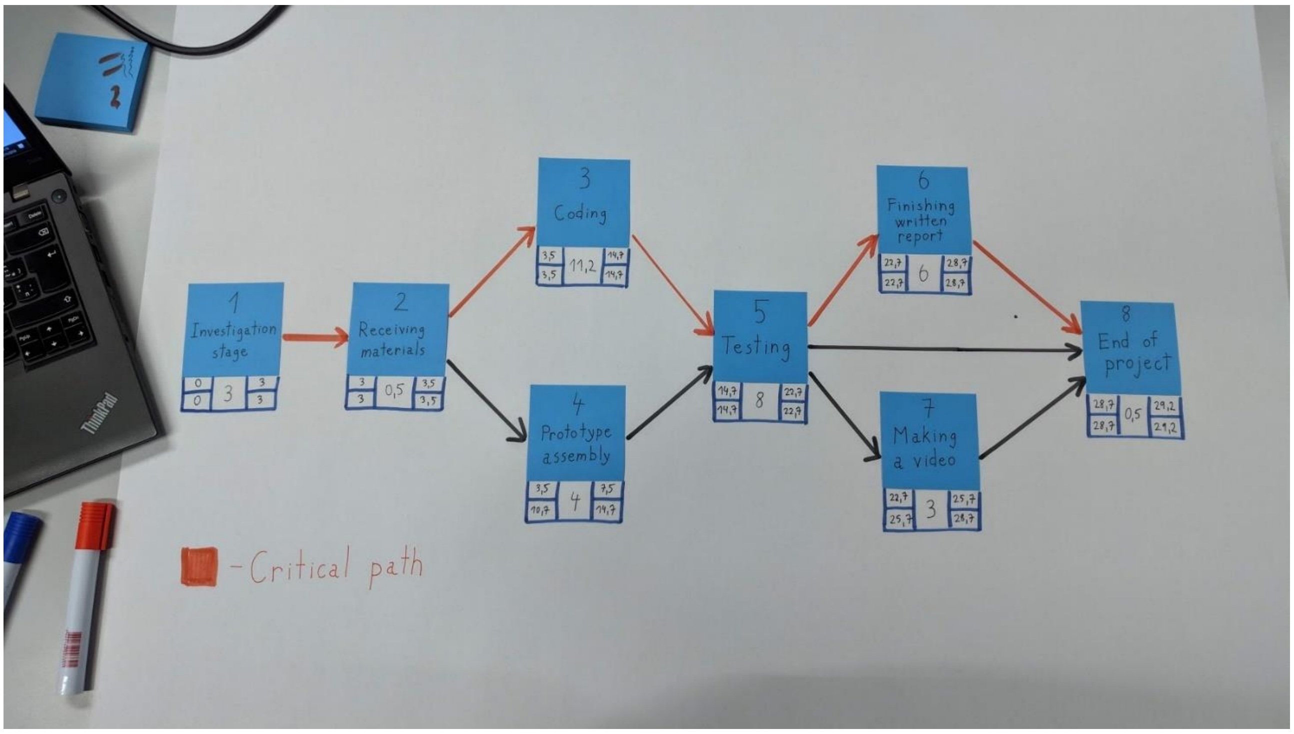


*Figure 1.3.1 – Critical path*

## 1.4 Project Quality Management

### 1.4.1 Testing

Having a proper test plan and following it, is vital to the success of the whole project. Testing ensures that the product works exactly as intended.

In this project we were following the V model of testing, which highlighted the need to plan and prepare for testing early in the development process. A large benefit of this model was that the testing phases were given the same level of attention as the corresponding development phases, thus minimizing the number of missed problems and bugs.

For this project we have determined the following testing methods and objectives:

**Unit testing**

- Every individual electronic component (sensors, motors, cables) and short pieces of code are tested
  - Do the sensors properly detect the black line?
  - Do the motors spin up when provided power from the battery?
  - Are these cables we use not damaged?
  - Does this for loop break after the expected amount of iterations?

- Testing performed by the developers
- Fully Whitebox testing – the internal structure is known to the developers
- Positive testing – the developers need to make sure the hardware pieces and lines of code work the way they expect them to

### Integration testing

- Testing of the interoperation of hardware (motors, sensors) and software
    - Do the motor forward and backward functions spin the wheels in the proper direction?
    - Does the Raspberry read the value of the sensor properly?
    - Does this if-statement properly signal the motor to stop when the sensor detects the black line?
    - Does the car respond to the input through the web interface?
- Testing performed by the developers but also with present customers (teachers)
- Whitebox, Graybox and Blackbox testing
- Both positive and negative testing – the developers should now try to break the code, put the smaller co-operating systems in unexpected circumstances

### System testing

- Testing of the assembled car if it is working the way it should
    - Is the car following the line?
    - Does the car not spin out due to too high speed?
    - Does the car not fall over when making turns?
    - Is the Raspberry not overheating?
    - Can we control the car through the web interface?
- Testing by developers and customers
- Blackbox testing
- Both positive and negative testing – developers and customers (teachers) are making sure the car is behaving according to the design and they also try to make the car malfunction.

### Acceptance testing

- Testing if the car is able to finish the race track
    - Does the car make the run without any hiccups?
    - Is the car (at least looking like) making the run at maximum possible speed?
    - Is the car faster than the other groups' cars?
    - Does the car push out all the boxes?
    - Is it possible to play football through the web interface?
- Testing by the customer (teacher)
- Blackbox testing
- Positive testing (presumably) – the customer (teacher) will evaluate if the car is according to the assignment he had given us. He will most likely not try to break it on purpose.

## 1.5 Reflection on the project plan

The project plan will be only as good as the team following it.

Due to the general disinterest shown towards the project by the team, the whole plan practically fell apart. Team members were not showing up for classes and have not communicated between each other about the status of the project.

The work on the project was started by a single person and 90% of all work has also been done by that single person.

The rest of the team had no clue about the status of the project, because they did not ask. The person working on everything was so busy with doing all the work, that he simply did not have the time to write daily reports about what he had done to the silent graveyard called the Facebook group chat with all members. With the members not asking, the idea never even occurred in his head.

# 2 Building the Robocar

## 2.1 Receiving the parts

We did not start building our robotic car out of thin air. Our teachers have prepared each team a small box with all the required parts.

The list of all parts is:

- 3pcs IR Reflective Line Sensors with QRE1113
- 1pcs Ultrasonic Distance Sensor with HCSR04
- 1pcs Smart Car Chassis Kit
- 4pcs 3-6V DC Motors
- 1pcs Raspberry Pi 3 A+
- 1pcs H-Bridge with L9110
- 1pcs 7,4V Li-Po Battery Pack
- 1pcs 5V USB Power Bank

### 2.1.1 Reflective Line Sensor

The line sensor that we received was a breakout board manufactured by Sparkfun. They are selling two types of these sensors – analogue and digital. We have received the analogue version, which relies on the Raspberry's UTL (Upper Trigger Level) and LTL (Lower Trigger Level), to correctly read if the surface under the sensor is reflective or not.

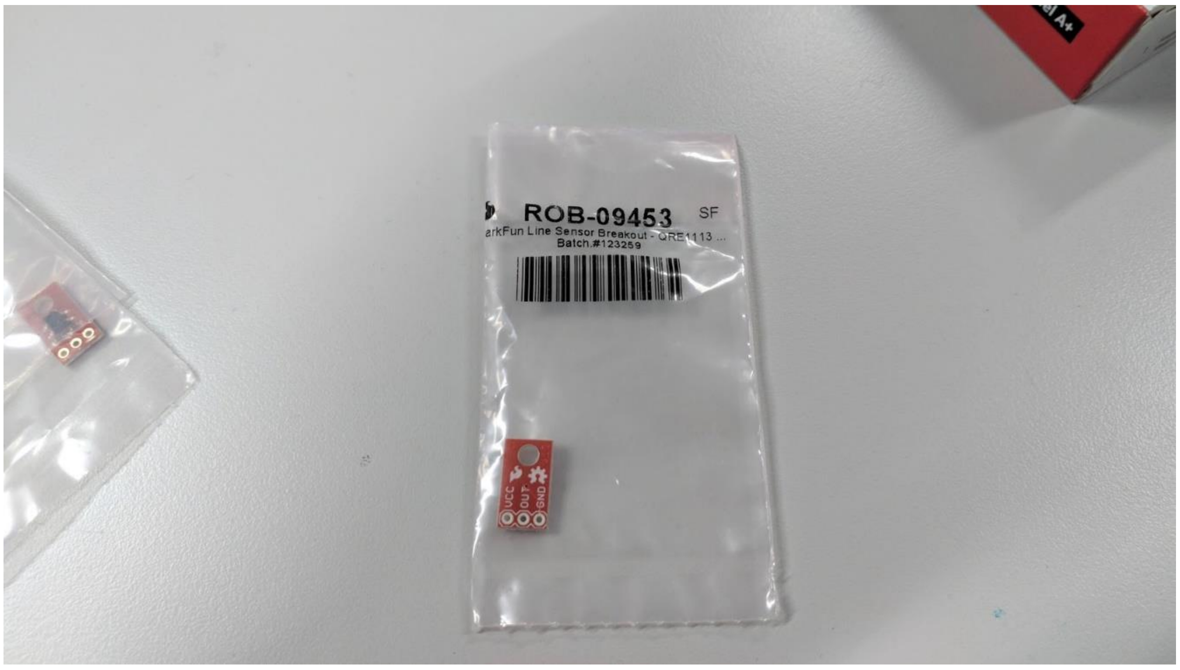


*Figure 2.1.1 – The Line Sensor*

**Upper Trigger Level and Lower Trigger Level of the Raspberry Pi**

During our studies we were introduced to the concept of Upper Trigger Level and Lower Trigger Level. What do they mean? The UTL value is a voltage at which the digital read value changes from logic 0 to a logic 1. The LTL value is the voltage at which the digital value changes from logic 1 to a logic 0. These two values differ slightly in the way, that a higher voltage is needed to go from 0 to 1 than going from 1 to 0. This is done by a Schmitt Trigger and when looking at the curve we can see the hysteresis effect.

The exact values of UTL and LTL on our Raspberry were:

UTL – 1,27 Volts

LTL – 1,15 Volts

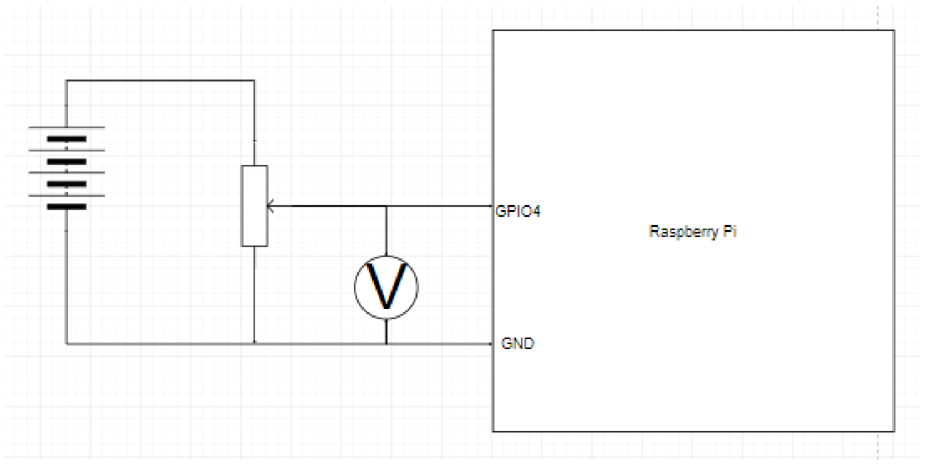


*Figure 2.1.2 – UTL and LTL measuring circuit*

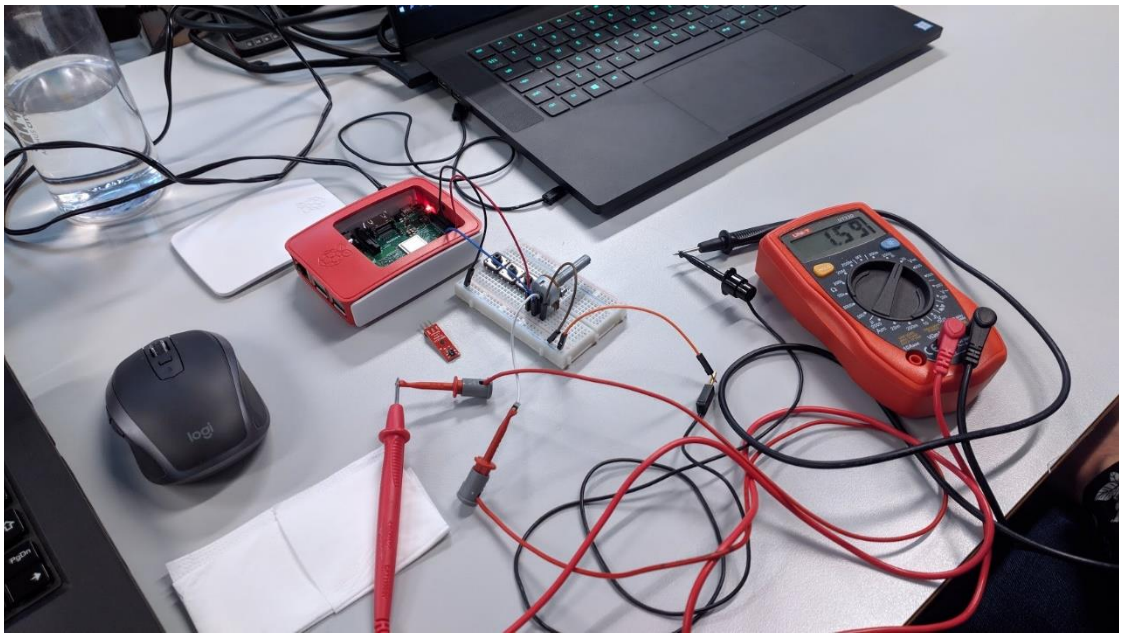


*Figure 2.1.3 – UTL and LTL measuring*

**The sensor's function in depth**

The IR reflectance sensor consists of two parts – and LED that is emitting IR (Infrared) light and a phototransistor that is sensitive to IR light. When power is applied to the VCC and GND pins the IR LED inside the sensor illuminates. A 100Ω resistor is connected in series with the LED to limit current. The second resistor on the board has a value of 10kΩ and its purpose is to pull up the output pin to the value of VCC. When the light from the LED is reflected back to the phototransistor, it will start to open, and the output voltage will begin to fall. The more IR light the phototransistor receives, the lower will be the output voltage.

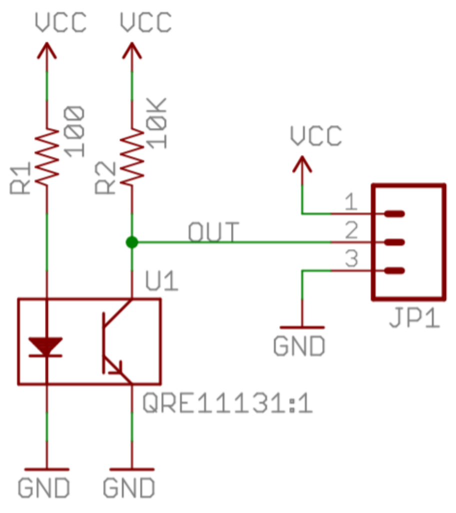


*Figure 2.1.4 – Line Sensor schematic*

These sensors are perfect for building line following robots, since white surfaces reflect more light than black, so the voltage output when the sensor is moved over a line will rise.

### 2.1.2 Ultrasonic Distance Sensor

The Ultrasonic Distance Sensor was also manufactured by Sparkfun. Its name already suggests the concept on which it is able to measure distance. Each module includes an ultrasonic transmitter (speaker) and a receiver (microphone).

The module needs a supply voltage of 5V, and is receiving input signals through the Trigger pin from the Raspberry. This signal makes the speaker emit an 8-cycle burst of ultrasound at the frequency of 40kHz and starts listening on the microphone. When the microphone receives the sound that was emitted by the speaker, the Echo output pin gets the value of logic 1. The distance can then be calculated by knowing the speed of sound (343 metres / second) and measuring the time how long it took to receive the Echo signal.

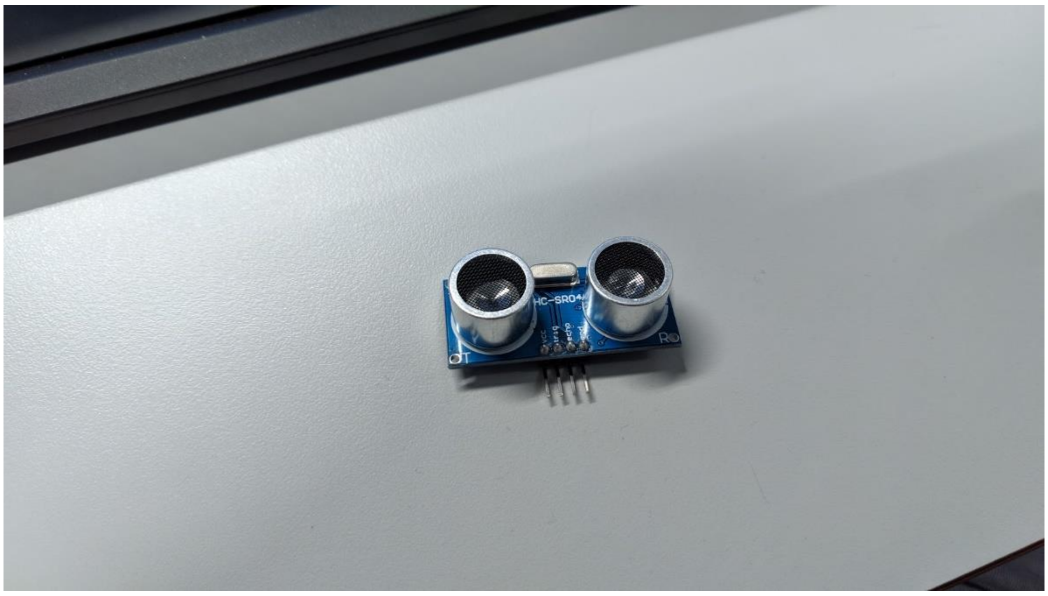


*Figure 2.1.5 – The Ultrasonic Distance Sensor*

**The Ultrasonic sensor's limitations**

The basis on which the sensor works is also its biggest flaw. Since it is emitting sound waves it relies on these sound waves returning unaffected by the environment. Sound, unlike light, can be bounced off much easier of angled surfaces. This means that the distance sensor works reliably only when the measured object has a flat plane and is angled at 90° towards the sensor. We have learned these limitations while testing the Distance Sensor with different types of objects.

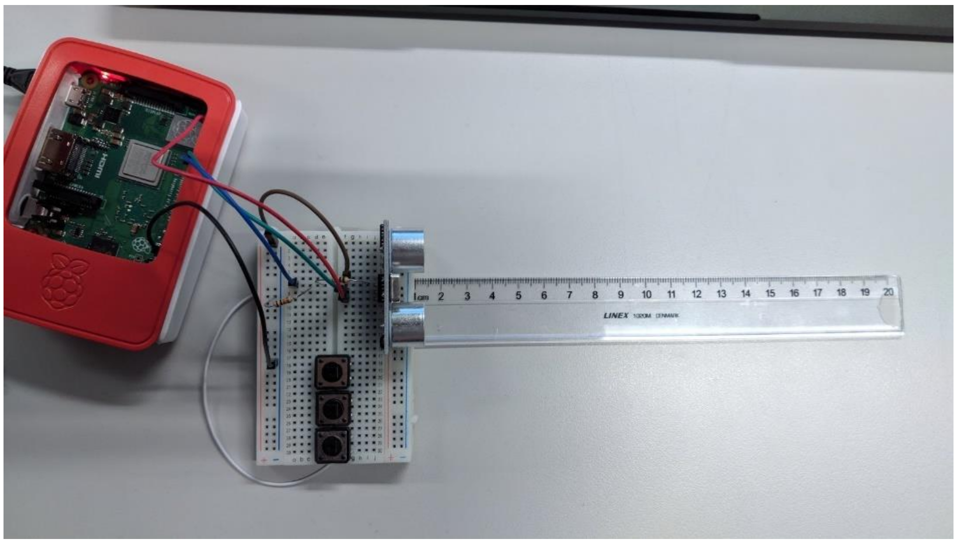


*Figure 2.1.6 – Testing the Ultrasonic Distance Sensor*

### 2.1.3 Smart Car Chassis Kit

The kit that we received consisted of:

- 2 acrylic base plates with mounting holes
- 4 wheels with the diameter of 66mm
- 8 motor holders
- 4 DC motors
- 4 speed encoders
- 6 standoffs
- 8 long screws
- 19 short screws
- 15 nuts

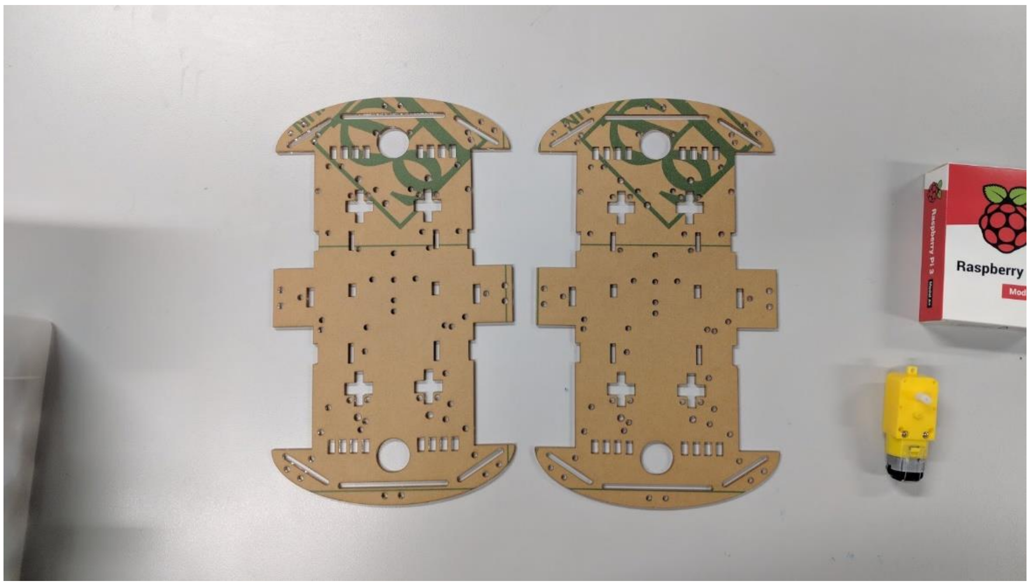


*Figure 2.1.7 – The acrylic base plates*

*Figure 2.1.8 – The wheels*

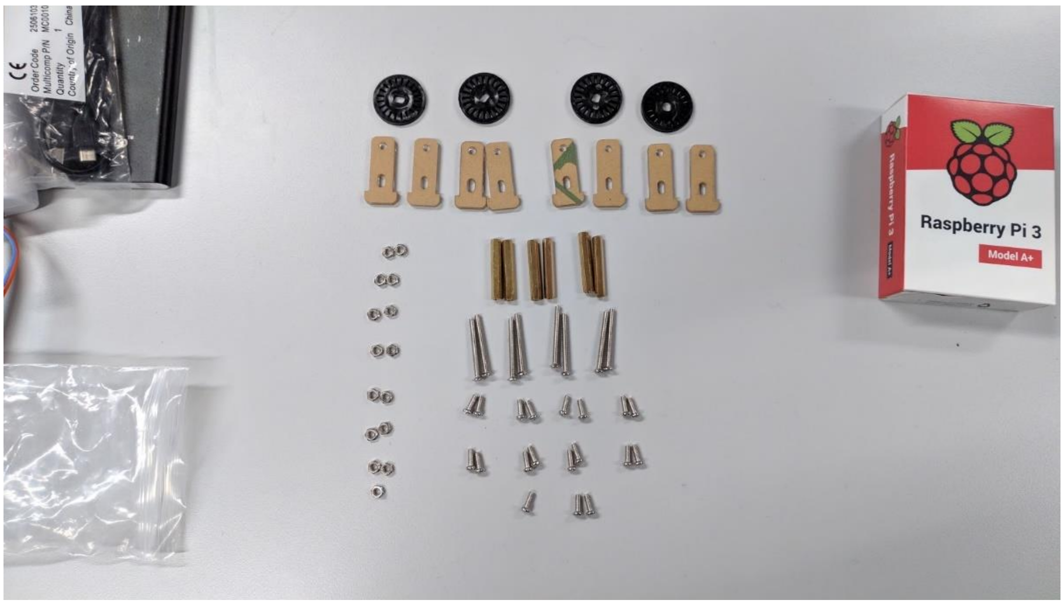


*Figure 2.1.9 – The screws, standoffs, nuts and speed encoders*

### 2.1.4 DC motors

The DC motors included in the kit were plastic gearbox motors, also known as "TT" motors, with a gear ratio of 1:48. A great feature of DC motors is that the direction of rotation is determined by the input voltage polarity. So we can make the wheels of a car spin both forward and backward just by flipping the power cables. They can be powered with voltages in the range of 3V to 6V. With a rising input voltage, the RPM (Revolutions Per Minute) of the motor also rises, resulting in a higher speed. The official specifications of these motors are:

- 3V, current 150mA, speed 90rpm
- 6V, current 200mA, speed 200rpm

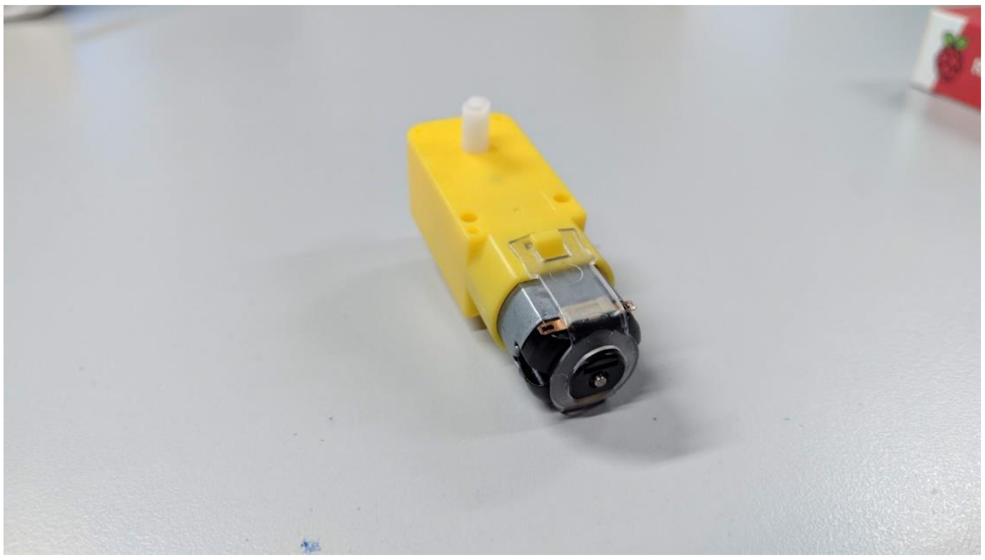


*Figure 2.1.10 – The DC motor*

These DC motors are very basic, so they don't have built-in encoders, speed control or positional feedback. There is also a 10% deviation allowance in the RPM at a given voltage, raising the possibility of some motors running faster than others at the same voltage. These concerns however are very minor and, in this project, can be deemed irrelevant.

### 2.1.5 Raspberry Pi 3 A+

The brains of our robot will be a Raspberry Pi. The Raspberry Pi is a credit-card sized computer developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools. The A+ model is a more compact and reduced-price version of the B+ model. The difference is that the A+ model has half the memory (512MB), only one USB A port and no Ethernet port as opposed to the B+ model. The full spec sheet of the Raspberry Pi 3 A+ is:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1,4 GHz
- 512MB LPDDR2 SDRAM
- IEEE 802.11 b/g/n/ac WiFi
- Bluetooth 4.2 / BLE (Bluetooth Low Energy)
- 40-pin GPIO header
- HDMI port
- USB 2.0 port
- CSI camera port
- DSI display port
- 4 pole 3,5mm Audio input / output jack
- MicroSD slot
- 5V / 2,5A DC power input (MicroUSB)

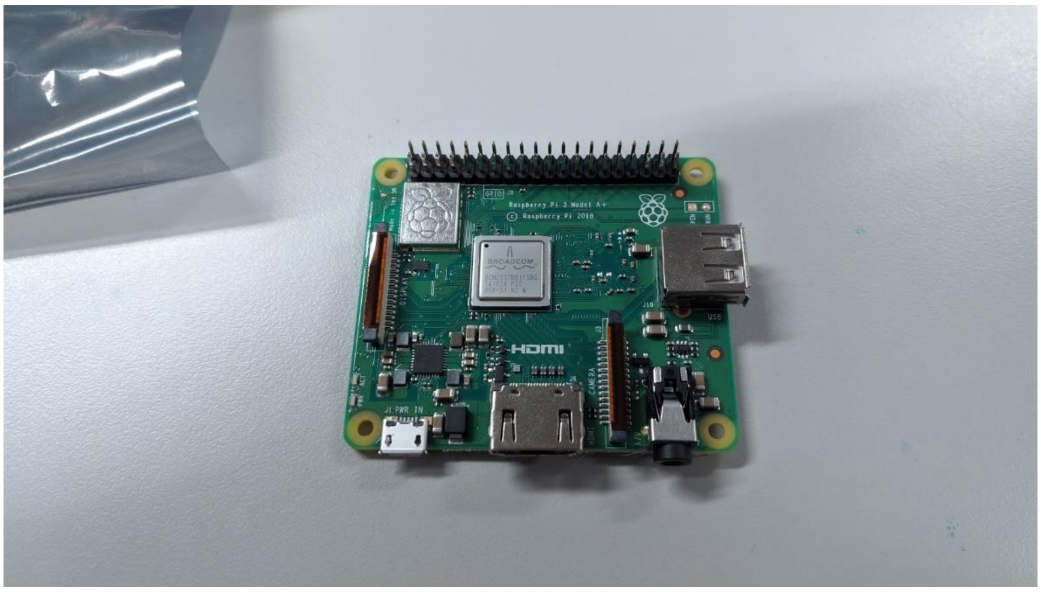


*Figure 2.1.11 – The Raspberry Pi*

### 2.1.6 H-Bridge

In order to drive the motors, we also needed an H-Bridge, since the Raspberry cannot provide enough power to drive the DC motors by itself. Our particular H-Bridge uses two L9110 motor control driver chips, one for each of the two output channels. The way it works is that the logical value of two input pins per channel determine the voltage and its polarity on the output. The L9110 chip supports a supply voltage range of 2,5V to 12V. This is the voltage which is then sent to the output. PWM is also supported by this chip, so the motor speed can be controlled through the input pins.

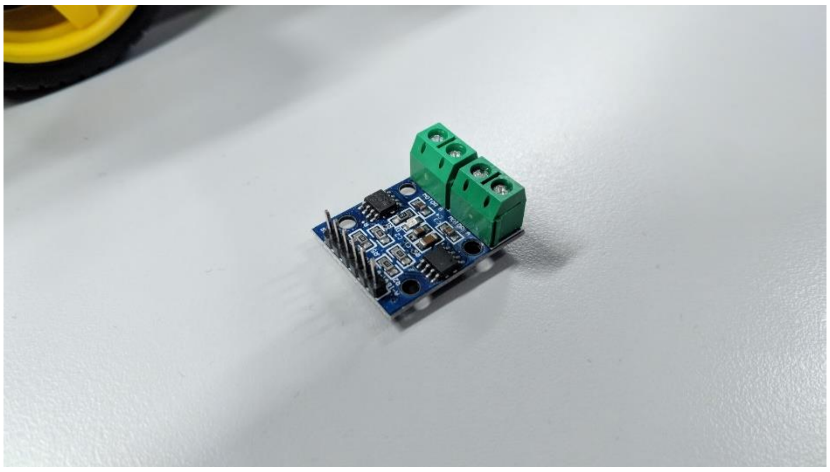
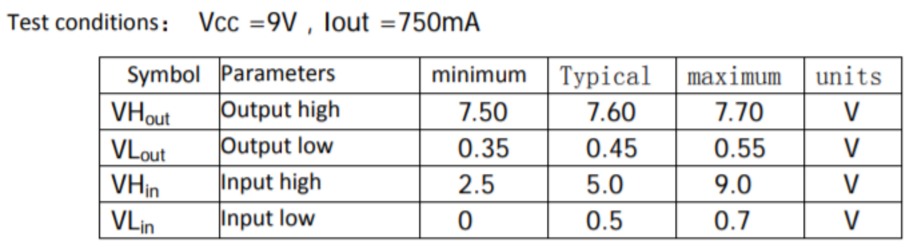


*Figure 2.1.12 – The H-Bridge breakout board*

#### A look at the L9110 datasheet

If we take a look at the datasheet of the L9110, among a lot of useful information we can find a table with the results of a lab testing. The reason this table is interesting for us because we can see that the H-Bridge does not actually supply the exact voltage that it is getting on the VCC pin, but it has a slight voltage drop. In the official testing the VCC was 9V and the measured output voltage was 7,6V. This voltage drop will play in our favour when powering the DC motors, since our battery has a voltage of 7,4V and the motors are rated for a maximum voltage of 6V.



Test conditions: Vcc =9V , Iout =750mA

| Symbol | Parameters | minimum | Typical | maximum | units |
|---|---|---|---|---|---|
| $VH_{out}$ | Output high | 7.50 | 7.60 | 7.70 | V |
| $VL_{out}$ | Output low | 0.35 | 0.45 | 0.55 | V |
| $VH_{in}$ | Input high | 2.5 | 5.0 | 9.0 | V |
| $VL_{in}$ | Input low | 0 | 0.5 | 0.7 | V |

*Figure 2.1.13 – Excerpt from the L9110 data sheet*

### 2.1.7 Li-Po Battery

The battery pack that we received was manufactured by Brainergy with a marking of 2S1P, voltage of 7,4V, a capacity of 4000mAh and a C rating of 45C.

The marking of 2S1P means that the battery has 2 cells in series and 1 cell in parallel (basically no cells in parallel). This is what gives a battery its voltage of 7,4V as well, since a single Li-Po cell has a voltage of 3,7V and when they are connected in series the voltage of both cells is added up. Our battery pack has no parallel cells but that is not a problem, because connecting cells in parallel is used just to double the capacity and also double the maximum possible current output. Neither of these are a concern because we are going to power simple DC motors that draw very little current.

The capacity of a battery is most often specified in Ah or mAh (Ampere hours or milli Ampere hours), but also sometimes in Wh (Watt hours). The capacity of 4000mAh or 4Ah means that the battery can supply 4A of current for 1 hour on a full charge, or 1A of current for 4 hours. (0,5A for 8 hours, etc.) The capacity in Wh can be easily determined by multiplying the voltage with the capacity. In our case the battery pack has a capacity of 7,4 * 4 = 29,6Wh.

The C rating of the battery is telling us the theoretical maximum current it can safely supply without blowing up. The formula through which we can calculate the current is to multiply the C rating with the capacity in Amperes. Our battery can safely supply a continuous current of 4 * 45 = 180A. Even if only for (4 / 180) * 3600 = 80 seconds.

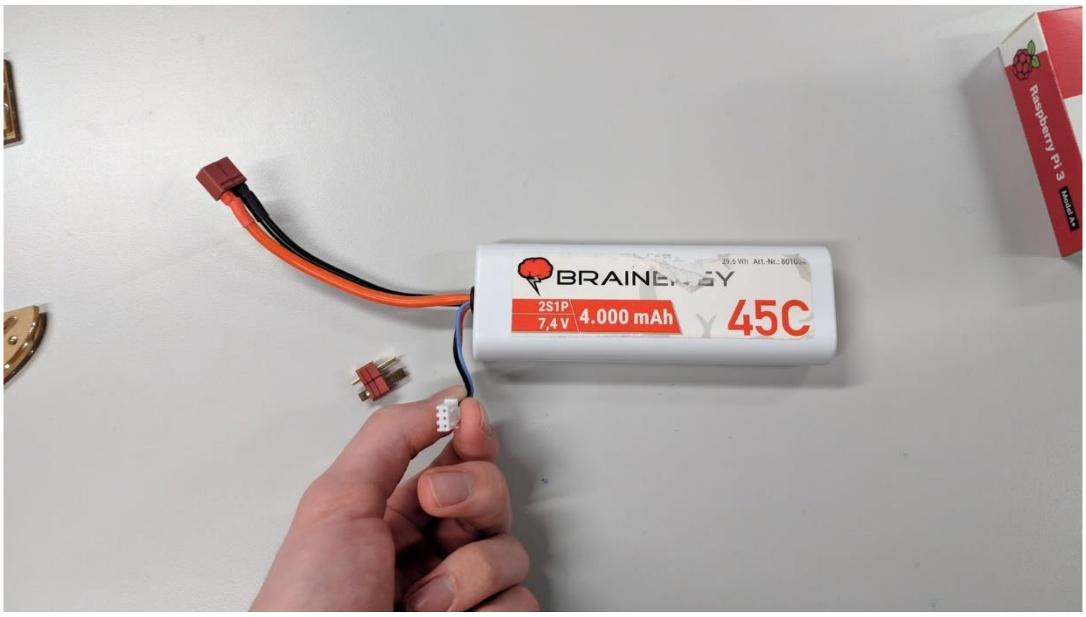


*Figure 2.1.14 – The Li-Po battery pack*

#### 2.1.8 USB Power Bank

The last piece of equipment that we received was a regular USB power bank with the usual USB output of 5V through a full-sized USB A port. It is manufactured by Poweradd and has the capacity of 5000mAh or 18.5Wh.

The power bank will be used to provide power for the Raspberry Pi. While the Raspberry Pi could have also been powered by the larger battery pack, I can see the reasoning why our teachers decided against it. That implementation would need a BEC (Battery Eliminator Circuit), in this case a SEPIC (Single-ended Primary-inductor Converter) could be used, which would add another point of failure for the people less experienced with electronics and might risk burning out some of the Raspberries.

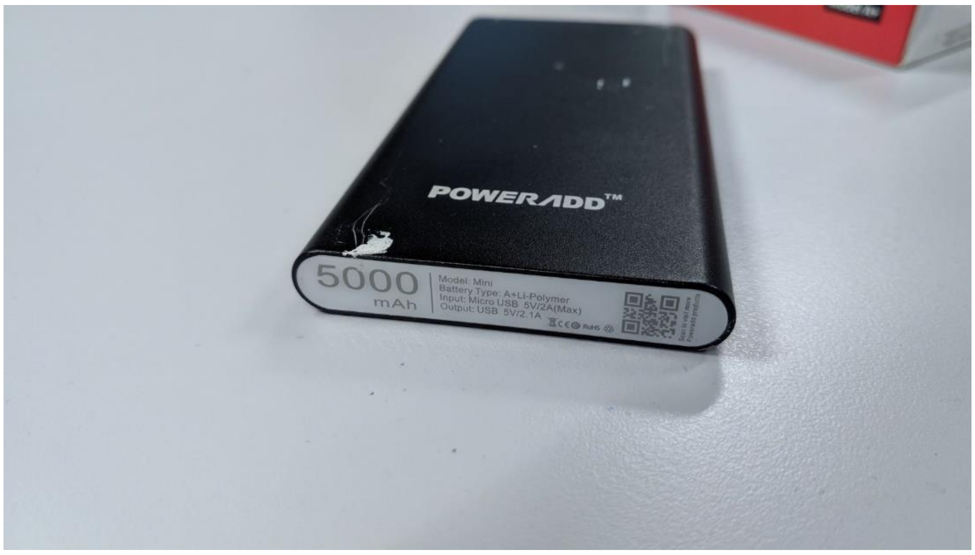


*Figure 2.1.15 – The USB power bank*

## 2.2 Block schematic and Wiring diagram

Before we jumped right onto the physical assembly, we had drawn a block schematic consisting of all the relevant functioning parts of the robot, created a GPIO pin to device assignment table and finally a wiring diagram.

#### 2.2.1 Drawing the Block schematic

A block schematic is a schematic of a system in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the individual blocks. Block schematics are drawn on a higher level than wiring diagrams, meaning that they just generally visualize what a system is built out of and how are its parts connected. Pin numbers and other electrical details are left out purposefully.

In principle a block schematic is made out of black boxes – pieces whose internals are hidden – in order to not distract the viewer with unnecessary details.

We have chosen to draw our block schematic in the free online diagram drawing software Draw.io. We have started drawing our block schematic with the center piece – the Raspberry and placed all the other components around it. The arrows between the boxes tell us which components act as inputs and which act as outputs when working together.

Some components actually have a bidirectional communication with the Raspberry, like the Distance Sensor and obviously the Cloud.

The "fat" arrow coming from the Raspberry to the H-Bridge tells us that there are multiple control pins and they work in tandem – a connection we could refer to as a bus.

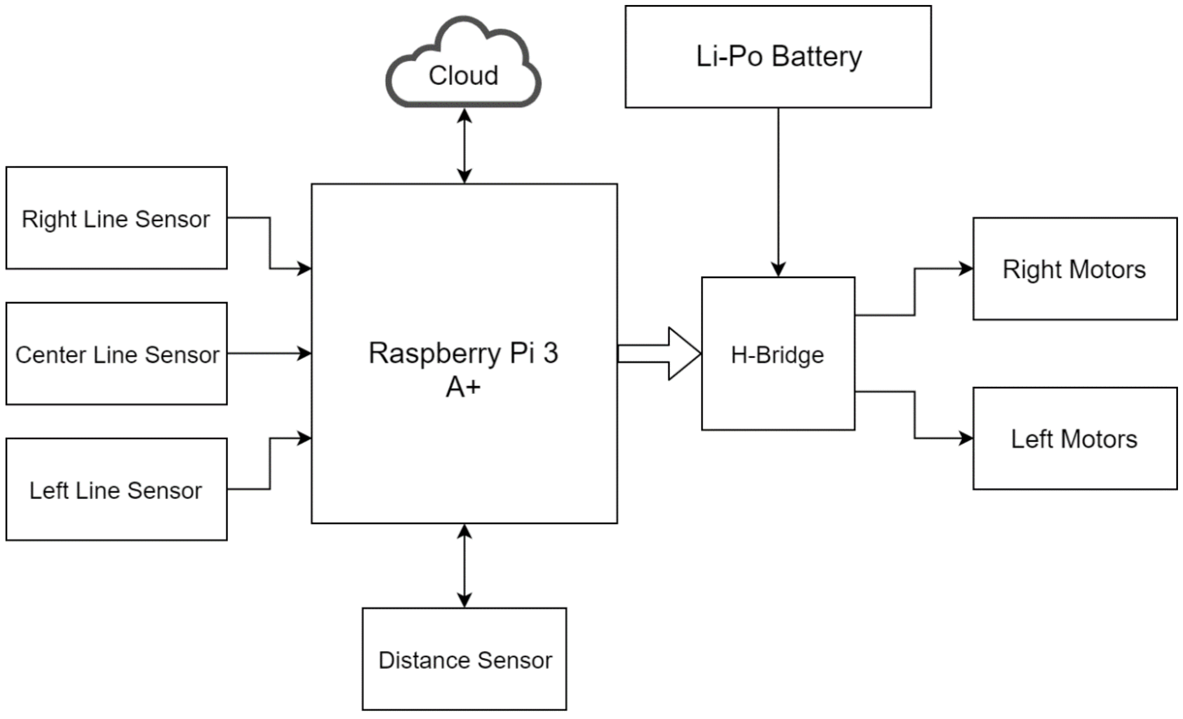


*Figure 2.2.1 – Block schematic*

### 2.2.2 Creating a GPIO pin to device assignment table

After creating a block schematic, we already had an idea in our heads how and where each of the components will be physically placed on the car's body. We knew that we want to have the H-Bridge in the center of the body – at the intersection of all four motors. The line sensors at the bottom either in the center or front of the car. A breadboard in the front of the car together with the distance sensor.

This way we were able to assign GPIO pins to these devices in a way that the cables will not cross each other unnecessarily, keeping a nice clean look of the car – so far only theoretically.

The table that we created was:

| GPIO PIN | Device |
|---|---|
| 14 | Left Line Sensor |
| 21 | Center Line Sensor |
| 20 | Right Line Sensor |
| 23 | Distance Sensor Echo |
| 24 | Distance Sensor Trigger |
| 18 | H-Bridge A1-B |
| 15 | H-Bridge A1-A |
| 12 | H-Bridge B1-B |
| 16 | H-Bridge B1-A |

### 2.2.3 Creating the wiring diagram

Now that we had both a block schematic and a GPIO pin assignment table, we were able to draw an in-depth diagram of the whole system – a wiring diagram.

Unlike a simple block schematic, a wiring diagram exactly shows the pins of all electronic components and all the wires connecting them. Other electronic components, like passive components (resistors, diodes, etc.), are also included in the circuit.

Very often – always – when a circuit is being built by an electrical engineer, he is building it according to a wiring diagram. Even if sometimes this diagram only exists in his head.

When the real circuit does not represent the wiring diagram, problems arise. A very common mistake that beginner hobbyists do is that they do not read the wiring diagram properly, connect the devices in a wrong way and in a better case just result with a non-functioning circuit. In a worse case they damage components and even suffer injuries.

The software that we used for this was Autodesk Eagle and we chose it for a very simple reason – we had previous experience with it and Autodesk is a nice company that provides educational licenses to their software for all students for free.

The finished diagram looked as follows:

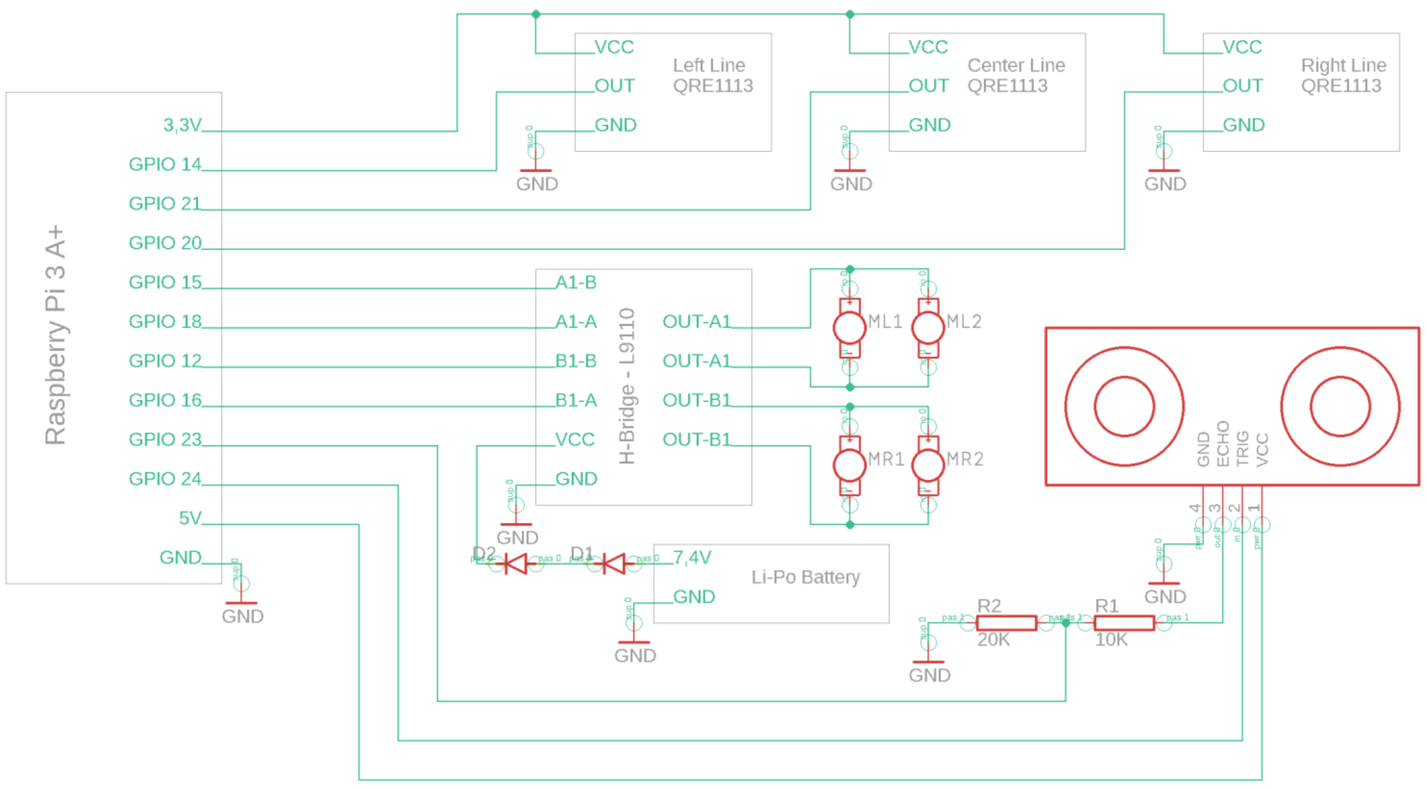


*Figure 2.2.2 – The Wiring diagram*

### The diode voltage dropping circuit

The first interesting part that might hit the eyes of the fellow reader are these two diodes in series connected between the Li-Po battery and the H-Bridge. Tracing back to the specifications of the DC motors, we can see that they are rated for a maximum voltage of 6V. Supplying it with a higher voltage would be considered overvolting which will make the motors spin faster but also raises the risk of damaging the motors.

Our Li-Po battery pack has a voltage of 7,4V. We have decided not to risk damaging our motors, so we had to figure out a way to reduce the voltage from the battery. There were multiple ways to do this:

- Voltage divider
- Diodes
- DC-DC buck converter

The voltage divider is quickly ruled out, because the motors draw a high enough power to exceed the limit of 0.25W of a regular resistor. It would be possible to connect more resistors in parallel to distribute the load, but it would still create a lot of heat, since all voltage divider circuits are in general very inefficient.

The best option would be to use a DC buck converter, since they have a high efficiency (up to 92%) and their output can be regulated. Unfortunately, we did not have enough time to buy these buck converters, so we had to go with the third, not great – not terrible, option.

Diodes are most often used for rectifying AC voltage or for creating a reverse voltage protection for DC circuits, and their nature of creating a voltage drop (due to them being semiconductors) is treated only as a side effect.

Diodes made out of different semiconducting materials have a different voltage drop. We have decided to use silicon diodes (1N4007) which have a voltage drop of 0,6V. The 1N4007 diode is rated for a constant current of 1A, so our circuit falls under this safe limit. We are using two diodes in series to generate a voltage drop of 1,2V, resulting in the VCC for the H-Bridge receiving 7,4 – 1.2 = 6,2V. Tracing back to the section where we looked at the H-Bridge's datasheet we will know that the output voltage to the motors will be slightly lower than the VCC voltage, so the motors will get a voltage below 6V, thus reaching our goal of running the motors in-spec.

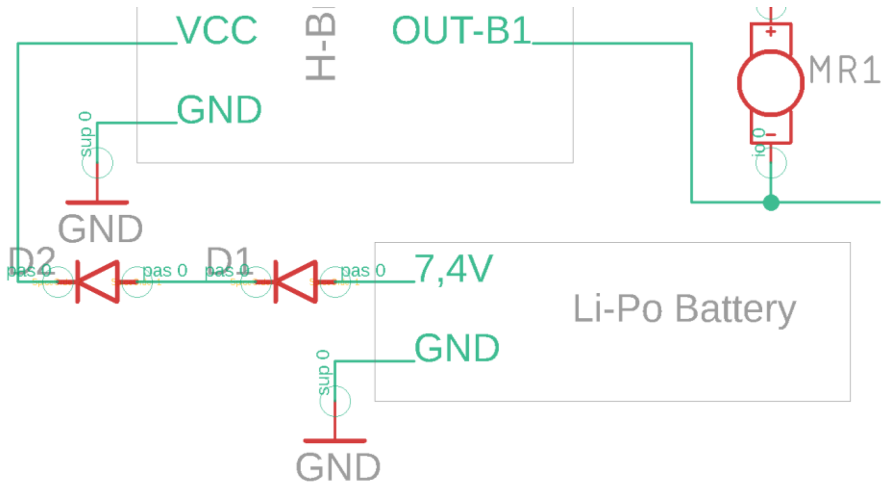


*Figure 2.2.3 – The diode voltage dropping circuit*

**The voltage divider**

The second interesting part is the voltage divider connected to the Echo pin of the Distance Sensor. We did mention that voltage dividers are very inefficient in the previous section but that was regarding a circuit that supplies power. In this case we are using a voltage divider in a digital low-power circuit, where efficiency is not a concern.

We need this voltage divider because of the nature of how this Distance Sensor works. This sensor is designed for 5V systems (for example an Arduino) and not 3,3V systems, like our Raspberry.

The best and tidiest way to solve this problem is using a Bi-Directional Logic Level Converter, but we do not have such a component at our disposal.

The way the Distance Sensor works, is that when it senses a returning ultrasonic sound signal, it returns a signal through the Echo pin with the same amplitude (same voltage) as the VCC voltage, so 5V. The problem is that the Raspberry GPIO pins operate at 3,3V, so if an input pin received a 5V signal it might damage the Raspberry.

We can easily reduce this 5V signal to be an approximately 3,3V signal by creating a 1:2 ratio voltage divider. According to Ohm's Law, the voltage will divide on the resistors in the same ratio as are their resistances. In our case we are using a 10kΩ and a 20kΩ resistor. The voltage on the 20kΩ will be (5 / 3) * 2 = 3,3V, so this is where we will connect the input pin of our Raspberry.

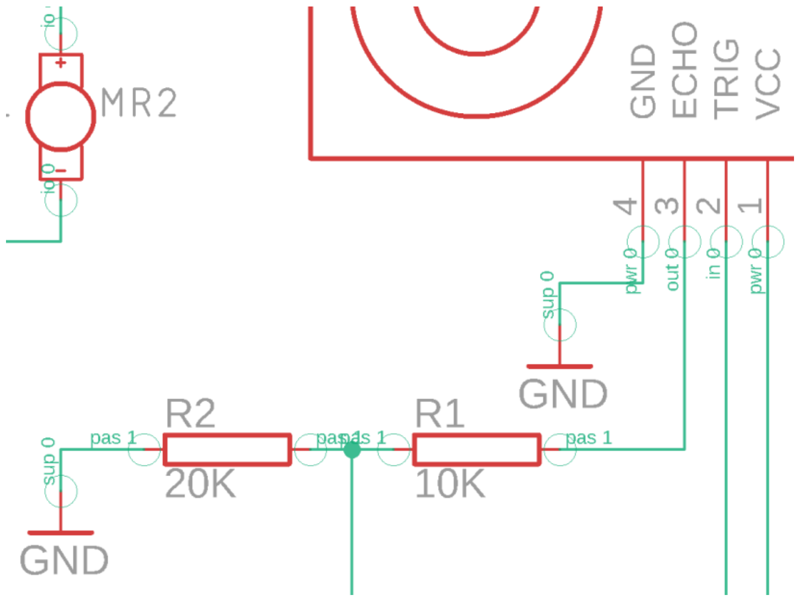


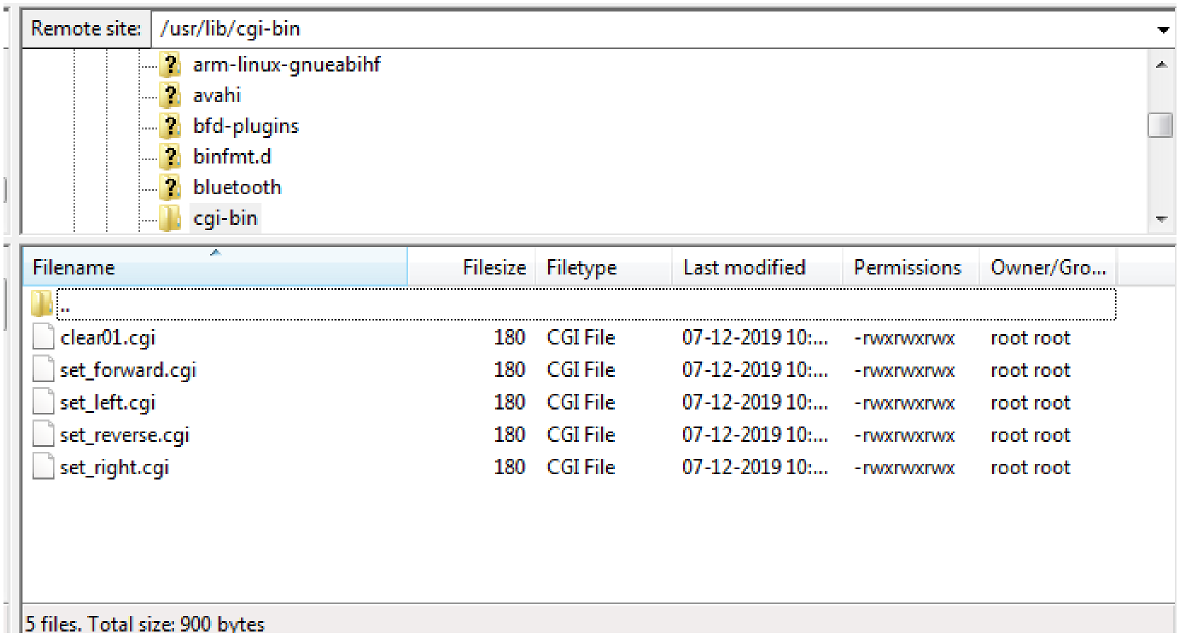
*Figure 2.2.4 – The voltage divider*

## 2.3 Setting up the Raspberry Pi

The Raspberry setup began by flashing Raspbian Buster Lite onto the SD card using BalenaEtcher, which is a very easy to use tool to flash images onto both USB sticks and SD cards.

The initial setup of the Raspberry has been done with a monitor connected to its HDMI port and a keyboard connected to the single USB port. We have created a user, set up a password, configured the keyboard layout configured the Raspberry to connect to the ITEK_1sem_1router access point and enabled SSH.

The configuration has been continued through SSH. All the repositories have been updated and then all the packages as well. This was followed by installing and configuring the Apache Web Server for future hosting of the web interface. Afterwards, Python was installed and also two libraries – gpiozero and pigpio, which we will be using to control the GPIO pins with our programs and the web interface.



Remote site: /usr/lib/cgi-bin

- arm-linux-gnueabihf
- avahi
- bfd-plugins
- binfmt.d
- bluetooth
- cgi-bin

| Filename | Filesize | Filetype | Last modified | Permissions | Owner/Gro... |
|---|---|---|---|---|---|
| .. | | | | | |
| clear01.cgi | 180 | CGI File | 07-12-2019 10:... | -rwxrwxrwx | root root |
| set_forward.cgi | 180 | CGI File | 07-12-2019 10:... | -rwxrwxrwx | root root |
| set_left.cgi | 180 | CGI File | 07-12-2019 10:... | -rwxrwxrwx | root root |
| set_reverse.cgi | 180 | CGI File | 07-12-2019 10:... | -rwxrwxrwx | root root |
| set_right.cgi | 180 | CGI File | 07-12-2019 10:... | -rwxrwxrwx | root root |

5 files. Total size: 900 bytes

*Figure 2.3.1 – Connected to the Raspberry through SFTP*

## 2.4 Physical assembly

After getting familiar with all the parts and creating both a block schematic and a wiring diagram, work was commenced on the physical assembly of the robot.

### 2.4.1 Figuring out the placement of components

This was something that we were already visualising in our heads when we were creating the GPIO pin to device assignment table. This time, we were taking each of the electrical components and thinking about how and where we would mount them on the base acrylic plate.

For the motors, we have found a YouTube video on how to secure them onto the chassis. The rest of the components were to be held by screws and duct tape.

It was also decided, that for interconnecting every component with the Raspberry we would use a breadboard, since it is easy to work with and we are not dealing with high currents, so its weakness of not creating perfect connections is irrelevant.

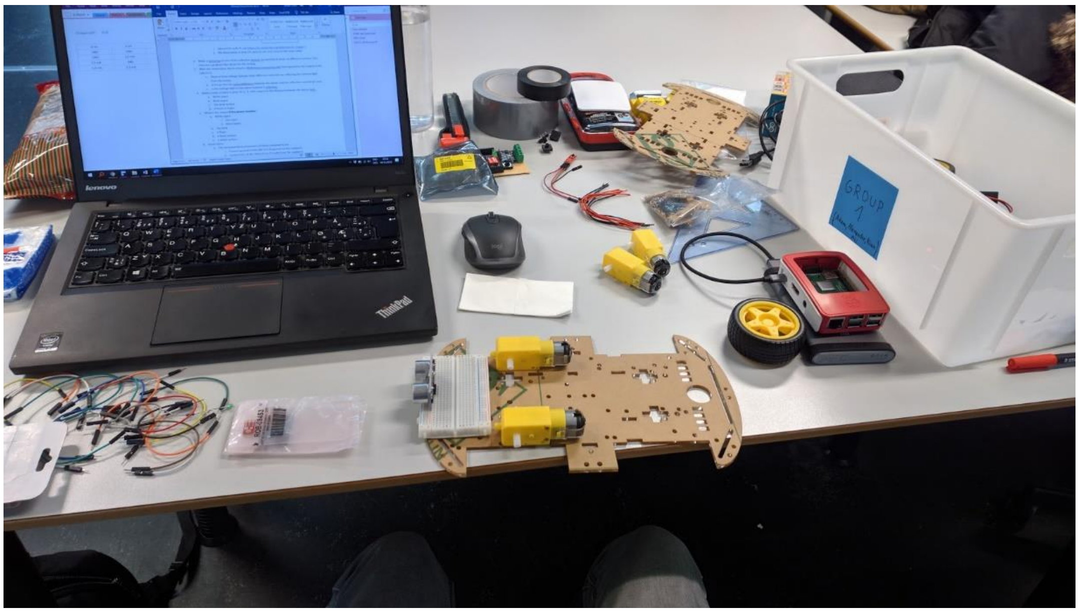


*Figure 2.4.1 – Placing components on the base plate without mounting them*

### 2.4.2 Soldering wires to devices

Some devices did not have pin headers or connectors, only solder terminals, so we had to solder wires onto them in order for us to be able to connect them to the breadboard or another device.

First of these devices were the DC motors. Luckily, we were provided with wires, so we have just soldered them onto the terminals of the motors. We have used the soldering station set to 360°C, helping hands and leaded solder.

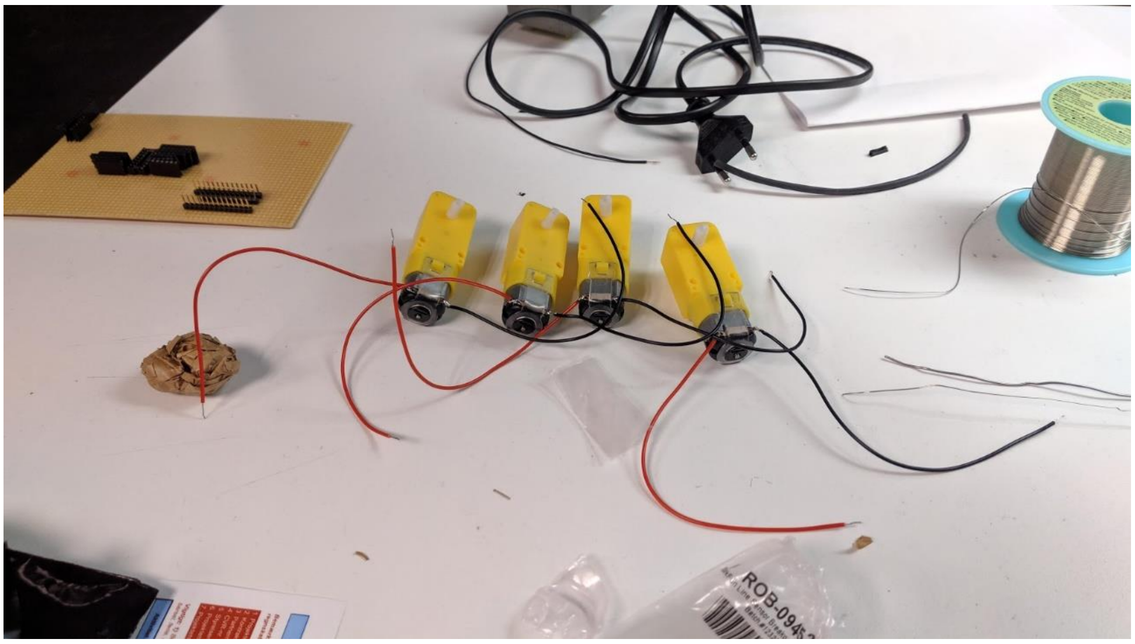


*Figure 2.4.2 – The DC motors with soldered wires*

After soldering wires to the motors, we have soldered male to male jumper wires to the Line Sensors. The equipment and techniques used were the same.

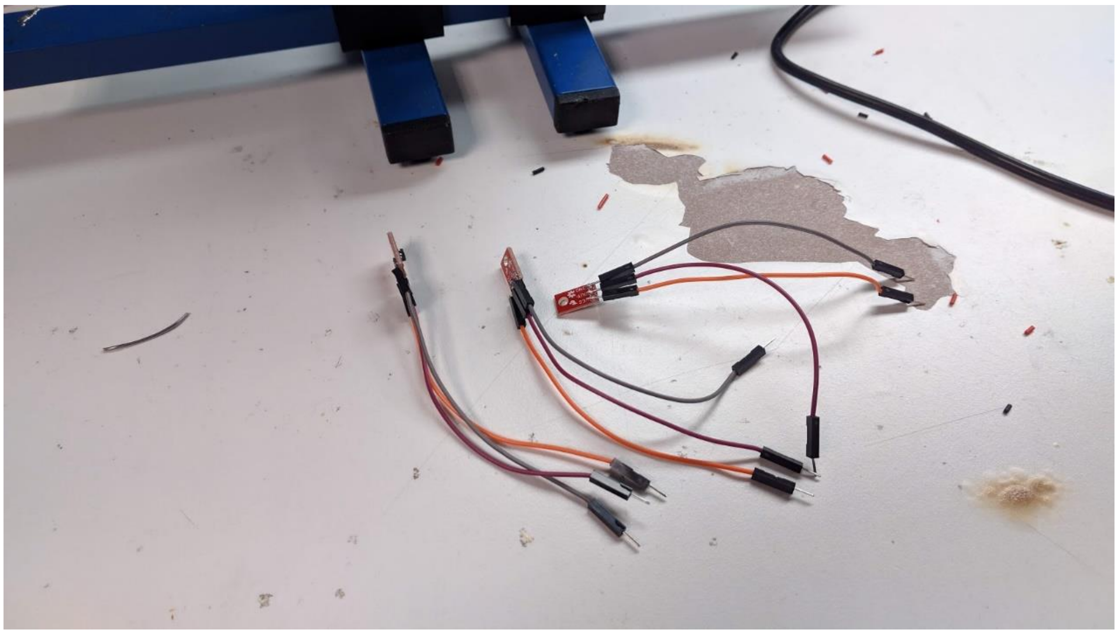


*Figure 2.4.3 – The Line Sensors with soldered wires*

### 2.4.3 Testing the motors

After soldering wires to the DC motors, we could finally mount them on the chassis. The mounting was followed by testing whether the motors are working properly. To do that we have prepared a very simple code that spins the motors forward, stops them, spins them backward, stops them and the cycle repeats.

The code is included in Appendix B.

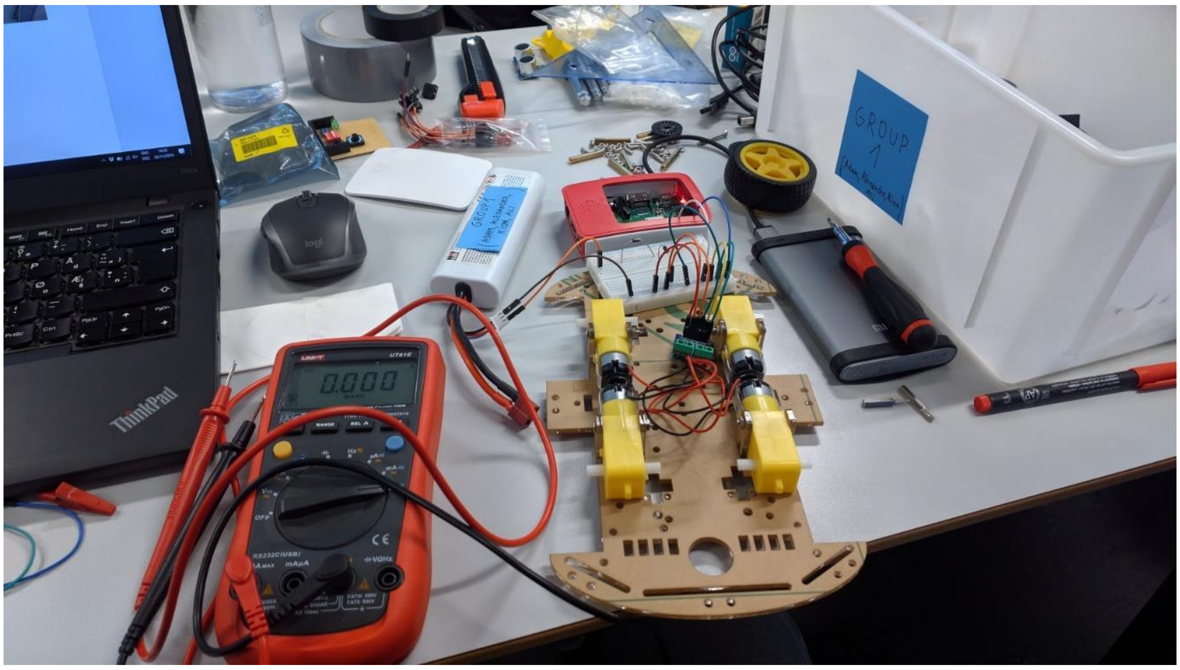


*Figure 2.4.4 – Testing the motors*

### 2.4.4 Mounting the Line Sensors

With the motors working as expected, we continued with mounting the Line Sensors. From previous experience with Line Sensors on lessons of Embedded Systems, we knew that the sensors must be very close to the ground. To achieve that we used standoffs.

First, we have installed the sensors to the center of the car. This design however, was later regretted. The issue was that when the sensors were in the center of the car, they were also in the center of rotation. This resulted in oversteering when the sensors detected a line. Because of this the car was moving forward very slow.

The sensor placement was later changed. They were mounted to the front of the car and also the third – the center senor – was mounted. The sensors were originally also working intermittently. They have sometimes detected a line even when there was no line.

That was fixed by putting the sensor even closer to the ground, achieved by using rubber bands as washers under the screws.

The sensors have been tested with a small code that we have created. It is included in Appendix C.

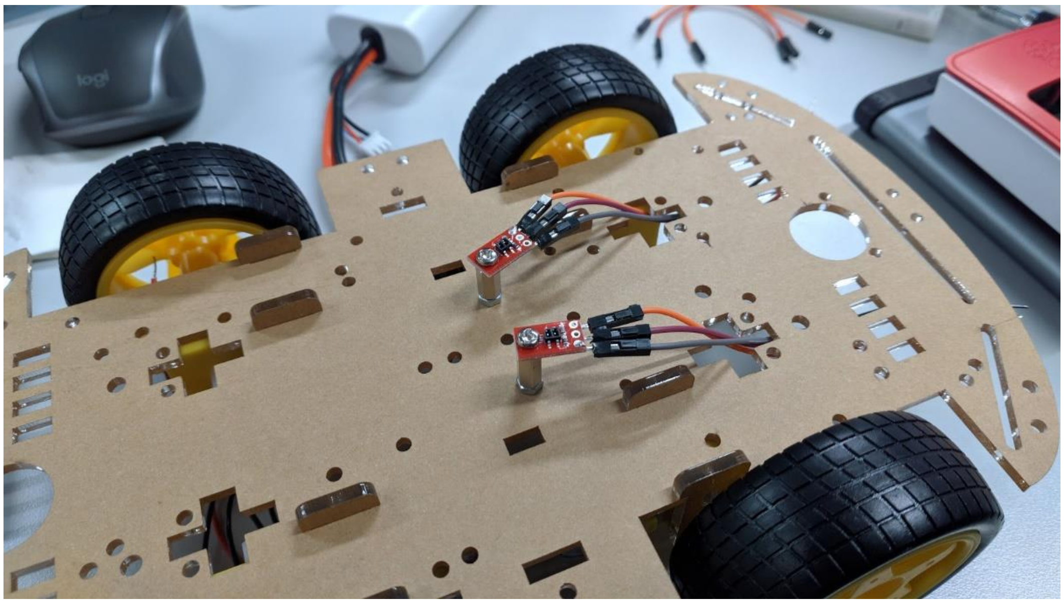


*Figure 2.4.5 – Line Sensors mounted in the center of the car*

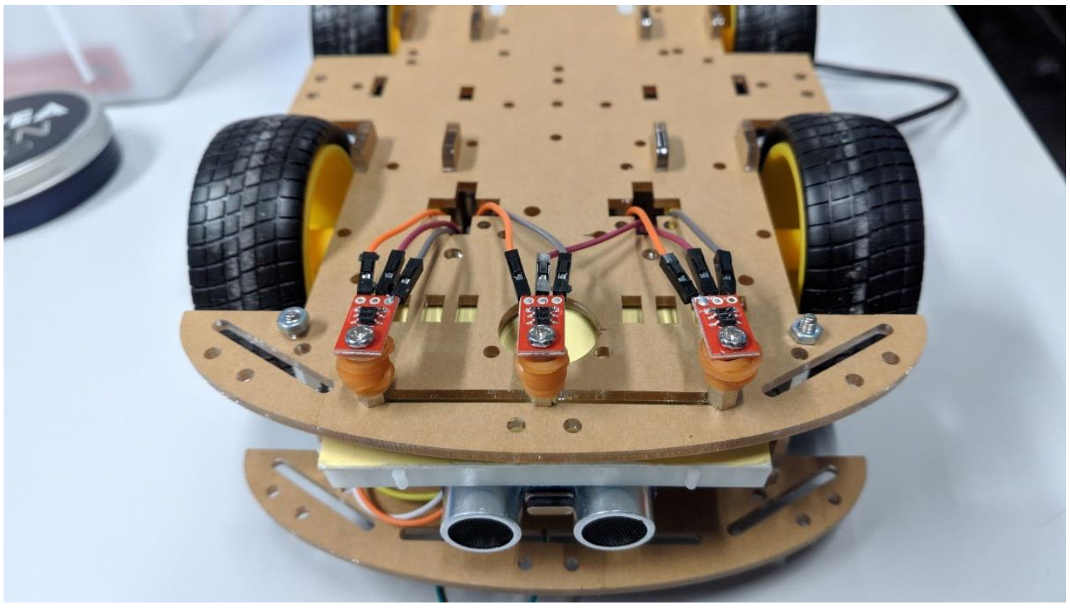


*Figure 2.4.6 – Line Sensors at the front of the car*

### 2.4.5 Finishing the lower body assembly

The next thing that we did was that we have connected the Distance Sensor to the breadboard and also all the other components – the H-Bridge, Line Sensors and we have also created our diode voltage dropping circuit and voltage divider for the Echo pin of the Distance Sensor. We have then connected wires that will be connected to the Raspberry and mounted the top plate.

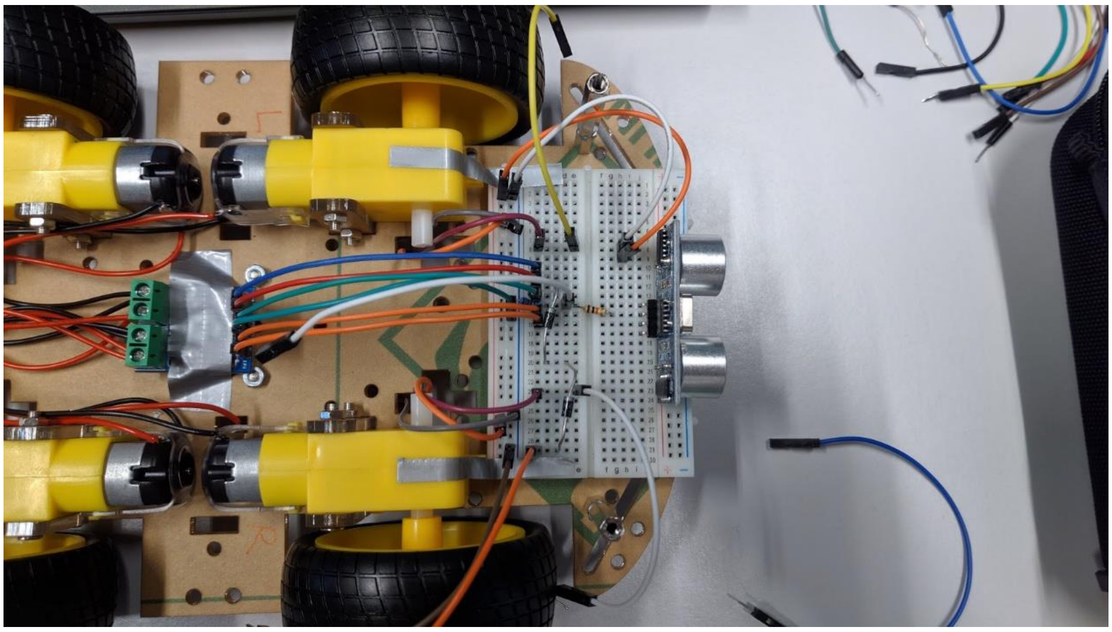


*Figure 2.4.7 – Top view of the breadboard circuitry*

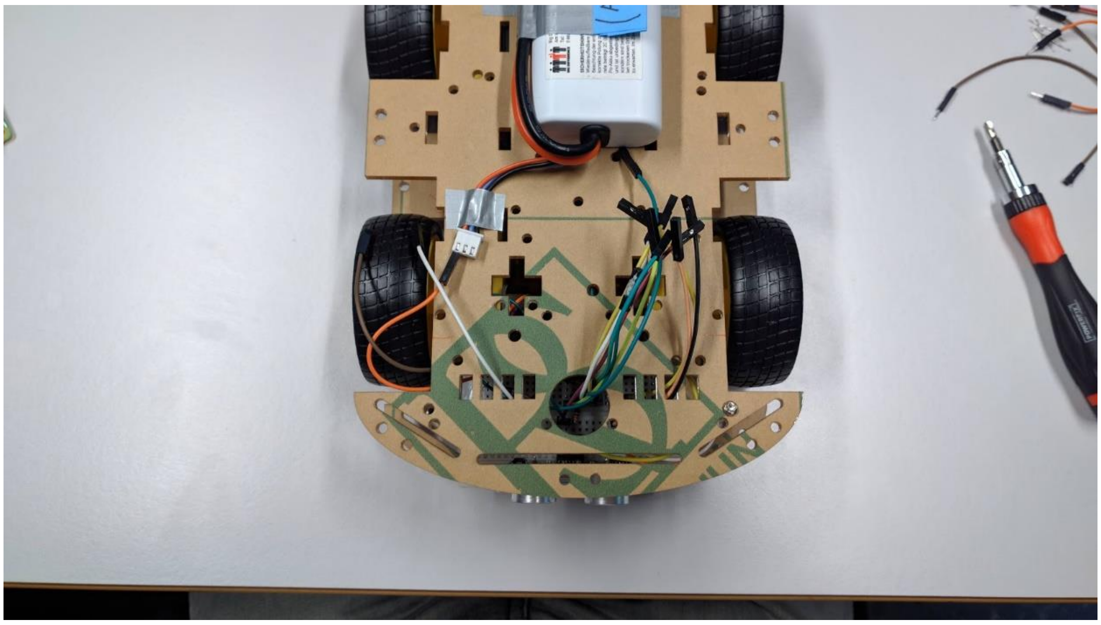


*Figure 2.4.8 – Top view with the top plate installed*

### 2.4.6 Connecting the Raspberry Pi

The last step to complete the assembly of the car was to wire up the Raspberry Pi on the top of the chassis. Thanks to the GPIO pin to device assignment table that we have created, doing this task was not a problem at all.

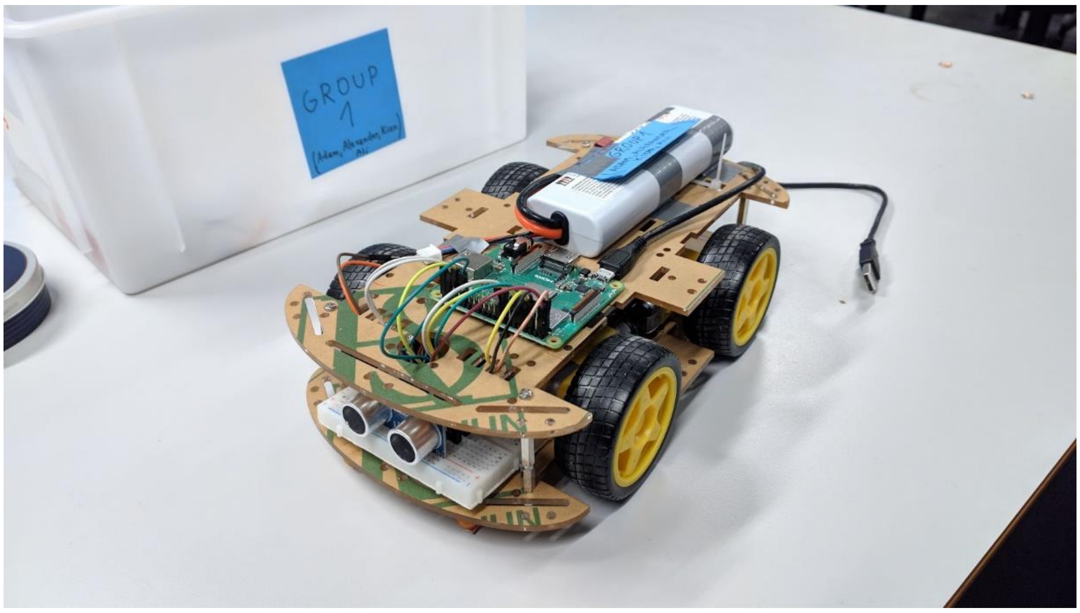


*Figure 2.4.9. – Robocar fully assembled*

### 2.4.7 Testing the Distance Sensor

The final component that we have not tested yet was the distance sensor. We have also created a small program for testing it. It is included in Appendix D.

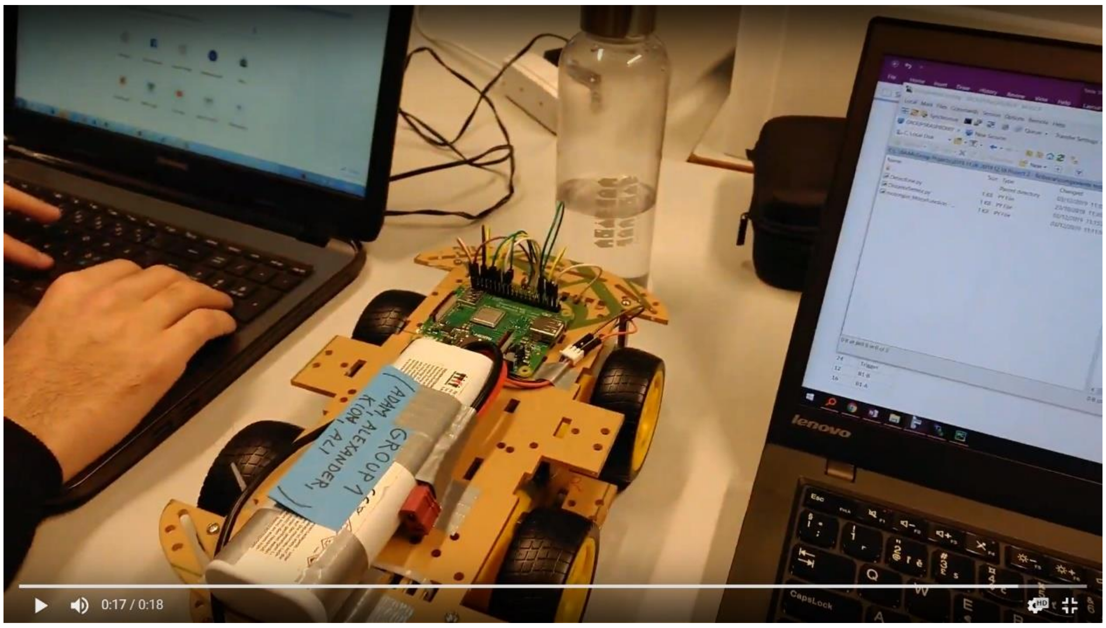


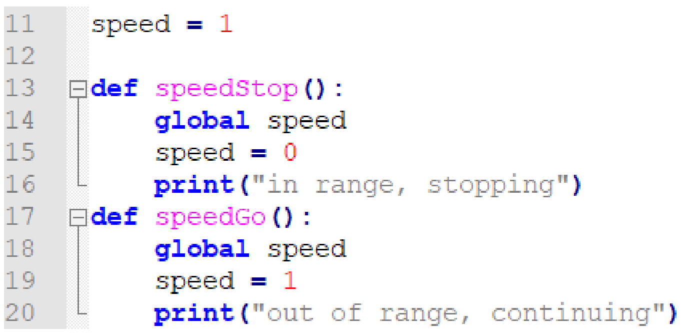
*Figure 2.4.10 – Testing the Distance Sensor*

## 2.5 Writing the programs

Having the assembly completed, we have moved onto the next task – programming. For the first two programs, line following and box pushing, we have used the Python programming language. The webpage was created in HTML and the remote controlling features were implemented with JavaScript.

### 2.5.1 Line Follow program

Before the main loop of the car starts, we have defined two functions called speedStop() and speedGo(), which alter the global variable called speed. The two possible states that the variable can be in after calling one of these functions is 1 or 0.

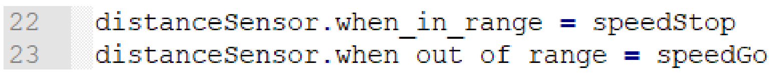


```python
speed = 1

def speedStop():
    global speed
    speed = 0
    print("in range, stopping")
def speedGo():
    global speed
    speed = 1
    print("out of range, continuing")
```

*Figure 2.5.1 – the speedStop() and speedGo() functions*

Below these functions we are using two callback functions from the DistanceSensor() class. Namely when_in_range and when_out_of_range. These functions trigger when the Distance Sensor detects a change of distance going below or above the distance defined in the threshold_distance parameter in the DistanceSensor() constructor.

The when_in_range function will call the speedStop() function and when_out_of_range will call the speedGo() function. What this does, is that when the Distance Sensor detects and object in front of it, it will set the speed variable to 0 – stopping the car to prevent a collision. When the object is removed, the speed variable is set to 1 and the car continues moving.



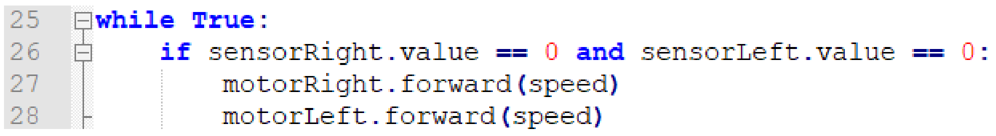
```python
distanceSensor.when_in_range = speedStop
distanceSensor.when_out_of_range = speedGo
```

*Figure 2.5.2 – the callback functions*

Next, the program enters its main loop where the line following logic is programmed. It is programmed with a four case if, elif and else stack. To detect the line, the left and right sensors are used, that is also the reason why there are four cases – two sensors with two possible states create four possible combinations.

The first scenario is that both left and right line sensors do not see a line, then motors on both sides of the car are set to go forward with the speed that is held in the speed global variable.

When looking at the description of the forward() method of the Motor() class, we will see that it accepts a floating point number between 0 and 1 as a parameter. In the background this method is doing a PWM (Pulse Width Modulation) duty cycle control with which it changes the speed of the motor.

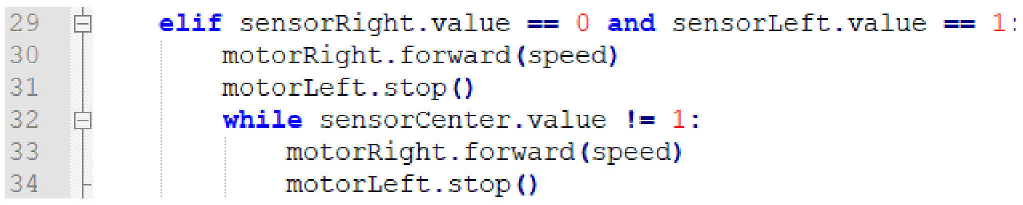


```
while True:
    if sensorRight.value == 0 and sensorLeft.value == 0:
        motorRight.forward(speed)
        motorLeft.forward(speed)
```

*Figure 2.5.3 – Robocar behaviour when no lines are detected*

Next possible scenario is that the left sensor detects a line while the right sensor does not. Then, the left motors stop while the right motors continue spinning. This way the robot is practically turning left.

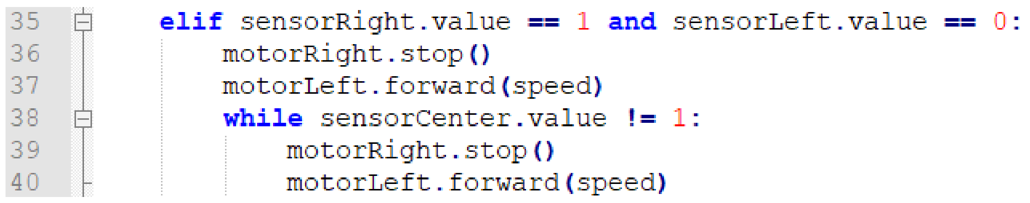
Here we have implemented another feature, the reason for it was that in sharp turns the car would not react quickly enough and run over the line with both sensors, then continue straight and run off the track. We have created a while loop in which the program gets trapped until the center sensor detects a line. Inside this while loop we have a continued spinning of the right motors, regardless of the state of the left and right sensors. This will make the robot return to the line, even if it ran slightly over in a sharp turn. When the robot returns to the line the main loop is resumed and turning is determined by the states of the left and right sensors.



```
    elif sensorRight.value == 0 and sensorLeft.value == 1:
        motorRight.forward(speed)
        motorLeft.stop()
        while sensorCenter.value != 1:
            motorRight.forward(speed)
            motorLeft.stop()
```

*Figure 2.5.4 – Robocar behaviour when the left sensor detects a line*

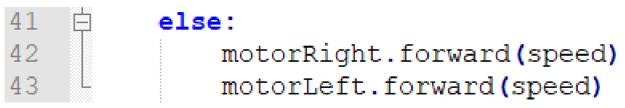
We have an identical case for the right sensor detecting a line. Only that there the right motors are stopped and the left motors continue running. This behaviour also implemented in the center sensor returning to line feature.



```
    elif sensorRight.value == 1 and sensorLeft.value == 0:
        motorRight.stop()
        motorLeft.forward(speed)
        while sensorCenter.value != 1:
            motorRight.stop()
            motorLeft.forward(speed)
```

*Figure 2.5.5 – Robocar behaviour when the right sensor detects a line*

The final case is that the left and right sensors both detect a line. This can happen at the intersection that we have on the race track. When this happens, the car is told to continue moving forward.



```
    else:
        motorRight.forward(speed)
        motorLeft.forward(speed)
```

*Figure 2.5.6 – Robocar behaviour when both sensors detect a line*

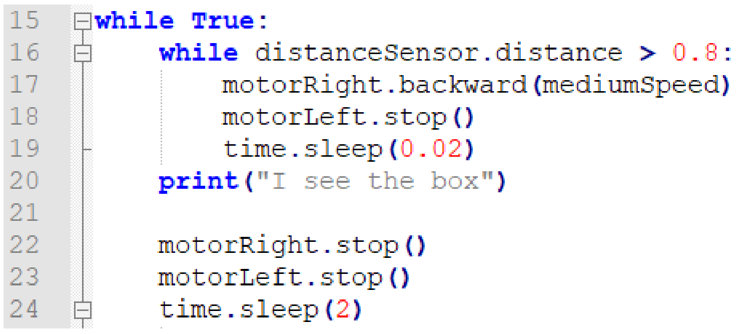
This main loop continues until it is stopped from the console. The line sensors are constantly being checked and the motors controlled accordingly. In the background the callback functions of the Distance Sensors are also constantly running, checking whether there is an object in front of the car, and stopping the car before a possible collision.

The full source code is in Appendix E.

### 2.5.2 Box Pushing program

In this program we have not defined any functions, all the program logic happens inside the main loop. Before the main loop we have only defined three global variables – fullSpeed, mediumSpeed and lowSpeed.

The main loop starts with a while loop that continues running until the sensor detects and object that is closer than 0,8 meters or 80 centimetres. While the program is inside this loop, the right motors are spinning backwards and the left motors are stopped – thus making the robot spin. There is also a 20ms sleep period defined before running the next iteration of the loop. The reasons for this are the physical limitations of the Distance Sensor. If there was no sleep period, then the Raspberry would be sending Trigger signals to the Distance Sensor faster than the sound could travel out of the sensor, reflect from an object, return to the sensor and thus raise the Echo pin. This would result in a lot of random and incorrect readings, making the Distance Sensor only a paperweight. The while loop is exited when the Distance Sensor detects an object closer than 80cm. In that moment the motors are stopped.

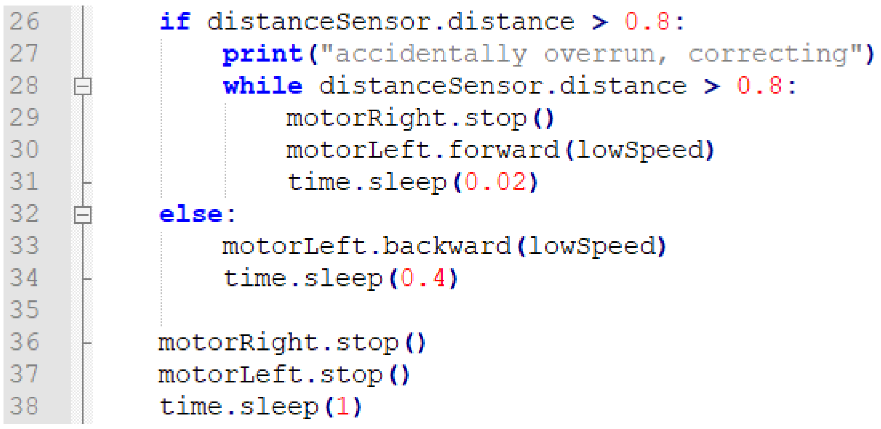


```python
while True:
    while distanceSensor.distance > 0.8:
        motorRight.backward(mediumSpeed)
        motorLeft.stop()
        time.sleep(0.02)
    print("I see the box")

    motorRight.stop()
    motorLeft.stop()
    time.sleep(2)
```

*Figure 2.5.7 – Robocar spinning until it sees an object*

Next, we have programmed a small corrective measure. Due to the motors taking a split second to completely stop, the robot always turns a bit further than it should, not fully facing the box.

In case the robot went over as much that the Distance Sensor does not see the box anymore, the robot will start spinning the other direction until it sees the box again. If the robot still sees the box, then it will spin the left motors backwards for 0,4 seconds. This number was determined by trial-and-error and seems to work reliably.



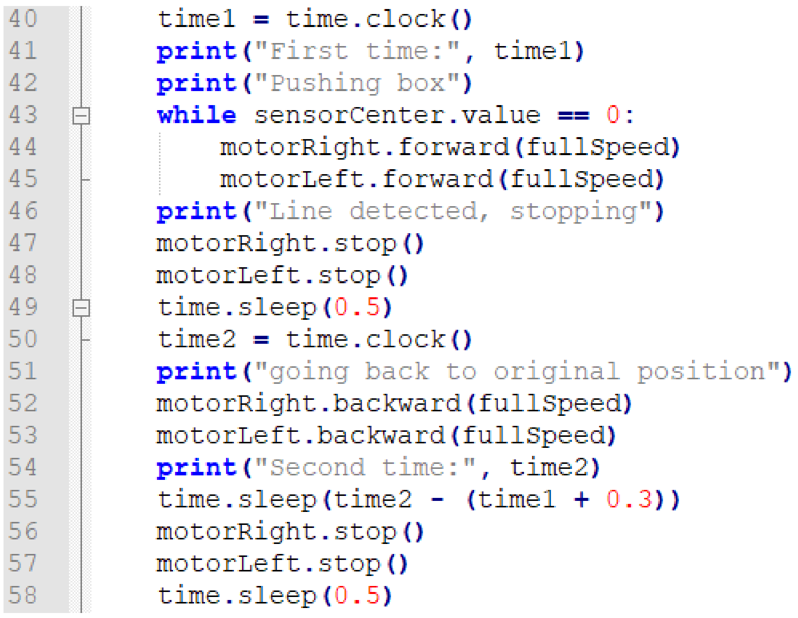
```python
if distanceSensor.distance > 0.8:
    print("accidentally overrun, correcting")
    while distanceSensor.distance > 0.8:
        motorRight.stop()
        motorLeft.forward(lowSpeed)
        time.sleep(0.02)
else:
    motorLeft.backward(lowSpeed)
    time.sleep(0.4)

motorRight.stop()
motorLeft.stop()
time.sleep(1)
```

*Figure 2.5.8 – Robocar making a corrective movement*

Now that the robot is standing still and facing the box, a first measurement of time is made using the clock() function from the time library. The clock() function returns the time in seconds since the program was started. The first measurement is assigned to the time1 variable.

Right after that, the program enter a while loop and the car starts moving forward. The program is stuck inside the while loop until the center sensor detects the line. After exiting the loop all the motors are stopped. Another measurement of time is made and saved inside the time2 variable.

All motors start spinning backwards. They are doing it as long as is the difference between time1 and time2 (plus a small correction of 300ms). The difference between time1 and time2 is actually the time it took to push out the box. By reversing for the same amount of time, the robot arrives back at the center of the board and the motors are stopped.

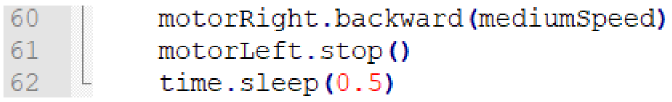


```
time1 = time.clock()
print("First time:", time1)
print("Pushing box")
while sensorCenter.value == 0:
    motorRight.forward(fullSpeed)
    motorLeft.forward(fullSpeed)
print("Line detected, stopping")
motorRight.stop()
motorLeft.stop()
time.sleep(0.5)
time2 = time.clock()
print("going back to original position")
motorRight.backward(fullSpeed)
motorLeft.backward(fullSpeed)
print("Second time:", time2)
time.sleep(time2 - (time1 + 0.3))
motorRight.stop()
motorLeft.stop()
time.sleep(0.5)
```

*Figure 2.5.9 – Code responsible for pushing out the box and then returning to the original position*

At the end of the main loop, the right motors start spinning backwards for at least 0,5 seconds and the main loop starts from the beginning. The reason for adding the 0,5 second delay is that the car should not detect the box which it had already pushed out. After 0,5 seconds the Distance Sensor is no longer pointing at the box and the program can resume looking for close objects again.

The full source code is in Appendix F.



```
motorRight.backward(mediumSpeed)
motorLeft.stop()
time.sleep(0.5)
```

*Figure 2.5.10 – The car turning away from the box it had already pushed out*

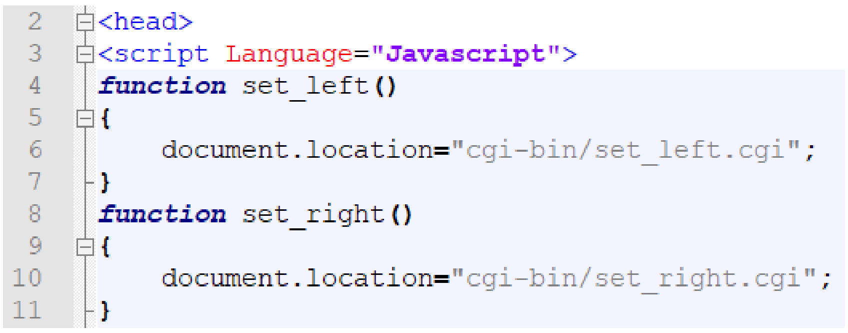
### 2.5.3 Web interface

For the web interface we have found inspiration online. At school we have learned how to execute bash commands with PHP. The problem with that for the Robocar was to make it turn reliably. If we had made it with PHP, then we would have either turned to robot by a small amount with every click or made it spin continuously until a stop button was pressed.

We have decided to use JavaScript, because it could execute CGI (Common Gateway Interface) scripts on both pressing and releasing a button. Making the script run only as long as the button is held down, afterwards executing another script.

**Webpage**

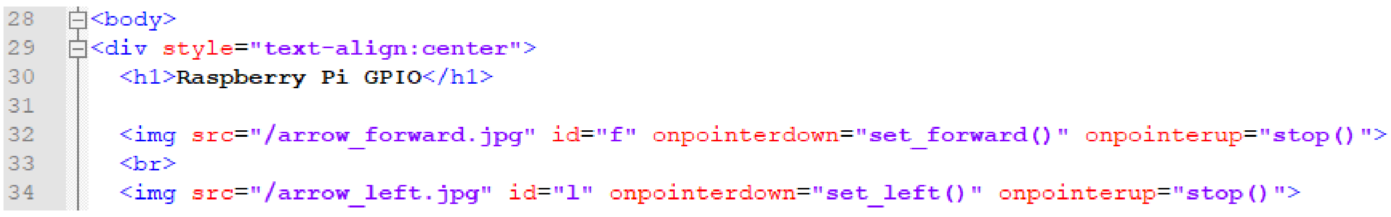
The HTML file starts with the JavaScript functions defined in the head. Each of these functions executes one of the CGI scripts. There are five functions in total: turning left, turning right, moving forward, moving backward and stopping.



```
<head>
<script Language="Javascript">
function set_left()
{
    document.location="cgi-bin/set_left.cgi";
}
function set_right()
{
    document.location="cgi-bin/set_right.cgi";
}
```

*Figure 2.5.11 – Two out of the five JavaScript functions*

Inside the HTML body, we have decided to use images as the buttons. Each of the images has two event triggers – onpointerdown and onpointerup. When an image is clicked, the function attached to onpointerdown is called. These functions are the ones we have defined previously. As soon as the mouse click is released, the function attached to onpointerup is called. The onpointerup function is the same for all images – stop() and it executes the stop.cgi script.



```
<body>
<div style="text-align:center">
  <h1>Raspberry Pi GPIO</h1>

  <img src="/arrow_forward.jpg" id="f" onpointerdown="set_forward()" onpointerup="stop()">
  <br>
  <img src="/arrow_left.jpg" id="l" onpointerdown="set_left()" onpointerup="stop()">
```

*Figure 2.5.12 – Two out of the five images and their event triggers*

## CGI scripts

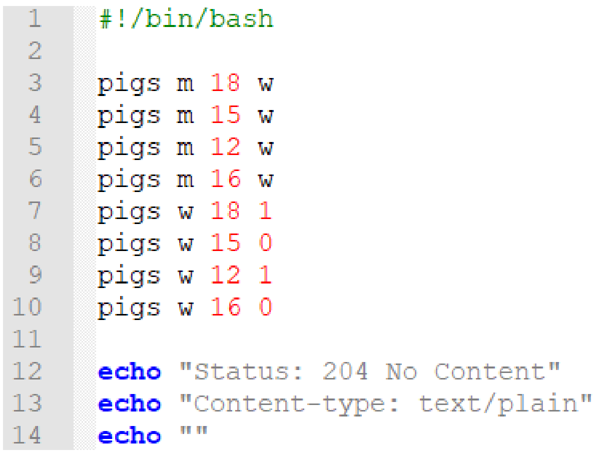
Each of the JavaScript functions execute a CGI script. CGI scripts are very powerful and can theoretically execute any kind of software module. We are using them to execute bash commands. The bash commands that we execute are provided by the pigpio library. In order for these commands to work, the pigpio daemon must be running on the Raspberry.

In order to turn on the motors, we must know how the H-Bridge works. During the motor testing we have created a table with different control pin combinations and the corresponding motor behaviour.

| Left Motors | | Right Motors | | Observation |
|---|---|---|---|---|
| **A-1A** | **A-1B** | **B-1A** | **B-1B** | |
| GND | GND | GND | GND | Motors not spinning |
| GND | 3,3V | GND | 3,3V | Motors spinning forward |
| 3,3V | GND | 3,3V | GND | Motors spinning backward |
| 3,3V | 3,3V | 3,3V | 3,3V | Motors not spinning |

With the help of the GPIO pin to device assignment table we knew which pins have to be set to a logic 0 and which pins to a logic 1 in order to achieve the action that we want.

The example below is the set_forward.cgi script. The first line specifies that the below commands should be executed with bash. It is followed by pigpio commands with values set according to the above table. The last three echo commands are meant for the browser. They contain a message that this script is not producing any content, so it should stay on the current page. If these commands were not used, the browser would display a blank page.



```
#!/bin/bash

pigs m 18 w
pigs m 15 w
pigs m 12 w
pigs m 16 w
pigs w 18 1
pigs w 15 0
pigs w 12 1
pigs w 16 0

echo "Status: 204 No Content"
echo "Content-type: text/plain"
echo ""
```

*Figure 2.5.13 – The set_foward.cgi script*

All the CGI scripts have the same structure, only differing in the values that motor pins are assigned.

The full webpage source and all CGI scripts are in Appendix G.

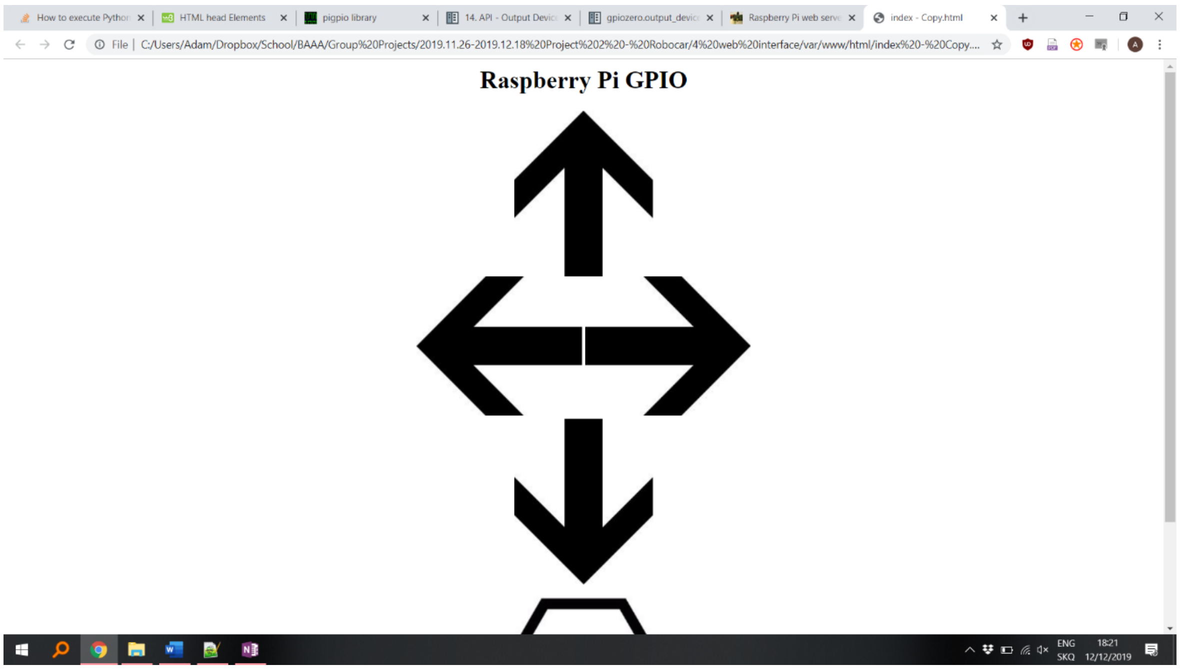


Figure 2.5.14 – Screenshot of the web inteface

## Reflections on the project

During the project the problems faced were not technical, but social. The team members were actively letting each other down.

Two days before the project deadline, the team members have finally agreed that there was a large issue with the team. Each member has agreed to write down their honest opinion on the process.

### Adam’s reflections

“There was a huge problem with attendance in our group. During this two-and-a-half-week project, some of the team members showed up only once a week. This obviously resulted with them being completely clueless about the status of the project. There was almost zero communication in the Facebook chat group that has been created. The robot for 90% was designed, built, programmed and documented by a single person.

More than 95% of this report was also written by a single person.”

### Ali’s reflections

“During this project, I have learned to create web GUI’s, and how to create scripts, that sets the backbone for the GUI that controls the car. I have not been as active as I would have been, but when active I have made my contribution to the group, or at least I feel so. We could have been better at planning work dates, and updating our to-do list, but all in all its been a okay experience.”

### Alexander’s reflections

“As reflection of the whole project I don’t think I got the most of it. Overall I am not a team player and I have never been, so coping with other people’s opinions is not my strength. Of course as a team we had some issues, but I think that went almost well at the end. We could have been more organized with the group meetings.”

### Kion’s reflections

“Group Dynamic has been bad to say the very least. And shortly we are all to blame in different aspects, myself and Ali, needs to learn that school starts at 8:30, Monday – Friday. Alexander needs to take more initiative when he is here instead of doing unrelated stuff, if you don't have anything to do, either find something or ask. We all four need to communicate better, Adam who have pulled the load off this assignment needs to change a couple of things ironically. First off, if he gets in as bad a group, I would argue he can see it from the get go, and he should have complained to a teacher since he does deserve better, it does not matter we would have been worse off, that is our problem in the end. Secondly, communication, I do understand the frustration he must have felt with us, but when our Friday agreement fell through (mostly due to my lack of communicating) he should either asked us to come Saturday or waited for Monday to continue, since him stepping in and finishing the build on his own only further deteriorates the group mentality and coherency, I do understand why I did so but it unfortunately does not help in the long run. And finally, Adam should be more harsh with the people he works with, if you’re doing good while the rest of the group is not pulling their load, you should stop it...”

# Conclusion

The main problems that this project has originally intended to create were of technical nature: Getting familiar with all the available components (Ch. 2.1), the challenge to design a wiring diagram (Ch. 2.2.3), assembling the robot (Ch. 2.4) and finally writing programs to achieve the project's three main deliverables. (Ch. 2.5)

The real challenges faced were created by the team members. The lack of communication and general interest in the project resulted in putting all the workload on a single team member. Luckily this team member was expecting this scenario to happen and did not succumb.

The project was in the end finished on time, with all the deliverables completed.

# Appendixes

List of appendixes:

# Appendix A: Group contract

## Group Contract

## for group: Eastern Invaders

## Project: Robocar project

## Period: 26.11.2019 - 17.12.2019

## Group members:

Adam Zilizi

Alexander Mitkov Dimitrov

Ali Akeel Ibrahim Mochlif Al-Benaian

Kion Valther Christensen

### 1. Problem description

Our first assignment is to create a robotic car that is able to follow a black line.

Second assignment is to make the car able to push boxes out of a circle.

Third assignment is to create a web interface through which the robot can be controlled.

Optional assignments are to make our robot go the fastest.

### 2. Documentation – process and project

**• During the course of the project period we will document the process and project with the help of:**

Microsoft Word for writing the Project Management report

Microsoft PowerPoint for creating a presentation

Phone for photos and videos

**• We will document and present the final project in the following way:**

We will create a PowerPoint presentation and make a video.

• **The group's documentation will be stored in the following location:**

Dropbox shared folder:

2019.11.26-2019.12.18 Project 2 - Robocar

## 3. Group meetings – frequency, length, agenda, and minutes

Twice a week for 1-2 hours

We are going to discuss the current plans (30 mins), difficulties (30 mins), ideas (30 mins)

## 4. Procedure in the case of a member's absence

Depends on the number of absent members:

- In case if high absence the meeting has to be rescheduled
- In case of only one person missing, the meeting can still be held and he will be given a briefing

## 5. Procedure regarding updating of the contract:

We discuss what we want to change on a meeting.

If everyone agrees we update the contract.

## 6. Procedure in the case of failure to comply with the contract:

Depends on the severity of the case:

- Serious problems - we need to take a step back and re-evaluate, re-discuss
- Minor problems - if they don't have a major impact we just adapt, try to work around it

## 7. Signatures

The following agree to this contract and will participate actively in the projects solution

| | |
|---|---|
| Adam Zilizi | Zilizi |
| Alexander Mitkov Dimitrov | [illegible] |
| Ali Akeel Ibrahim Mochlif Al-Benaian | [illegible] |
| Kion Valther Christensen | Kion Christensen |

## Appendix B: Motor test program

```
import gpiozero
import time

motorRight = gpiozero.Motor(forward=18, backward=15)
motorLeft = gpiozero.Motor(forward=12, backward=16)

while True:
    print("Motor spinning forward.")
    motorRight.forward()
    motorLeft.forward()
    time.sleep(5)
    motorRight.stop()
    motorLeft.stop()
    time.sleep(3)
    print("Motor spinning backward.")
    motorRight.backward()
    motorLeft.backward()
    time.sleep(5)
    motorRight.stop()
    motorLeft.stop()
    time.sleep(3)
```

## Appendix C: Line detect test program

```
from gpiozero import LineSensor
from signal import pause

sensorRight = LineSensor(20)
sensorLeft = LineSensor(14)
sensorRight.when_line = lambda: print('right Line detected')
sensorRight.when_no_line = lambda: print('No right line detected')
sensorLeft.when_line = lambda: print('left Line detected')
sensorLeft.when_no_line = lambda: print('No left line detected')
pause()
```

## Appendix D: Distance sensor test program

```
from gpiozero import DistanceSensor
from time import sleep

sensor = DistanceSensor(echo=23, trigger=24, max_distance=5)

while True:
    print('Distance: ', sensor.distance * 100)
    sleep(1)
```

# Appendix E: Line follow program

```
import gpiozero
import time

motorLeft = gpiozero.Motor(forward=18, backward=15)
motorRight = gpiozero.Motor(forward=12, backward=16)
sensorRight = gpiozero.LineSensor(20)
sensorLeft = gpiozero.LineSensor(14)
sensorCenter = gpiozero.LineSensor(21)
distanceSensor = gpiozero.DistanceSensor(echo=23, trigger=24, max_distance=5, threshold_distance=0.4)

speed = 1

def speedStop():
    global speed
    speed = 0
    print("in range, stopping")
def speedGo():
    global speed
    speed = 1
    print("out of range, continuing")

distanceSensor.when_in_range = speedStop
distanceSensor.when_out_of_range = speedGo

while True:
    if sensorRight.value == 0 and sensorLeft.value == 0:
        motorRight.forward(speed)
        motorLeft.forward(speed)
    elif sensorRight.value == 0 and sensorLeft.value == 1:
        motorRight.forward(speed)
        motorLeft.stop()
        while sensorCenter.value != 1:
            motorRight.forward(speed)
            motorLeft.stop()
    elif sensorRight.value == 1 and sensorLeft.value == 0:
        motorRight.stop()
        motorLeft.forward(speed)
        while sensorCenter.value != 1:
            motorRight.stop()
            motorLeft.forward(speed)
    else:
        motorRight.forward(speed)
        motorLeft.forward(speed)
```

# Appendix F: Box push program

```
import gpiozero
import time

motorLeft = gpiozero.Motor(forward=18, backward=15)
motorRight = gpiozero.Motor(forward=12, backward=16)
sensorRight = gpiozero.LineSensor(20)
sensorLeft = gpiozero.LineSensor(14)
sensorCenter = gpiozero.LineSensor(21)
distanceSensor = gpiozero.DistanceSensor(echo=23, trigger=24, max_distance=5)

fullSpeed = 1
mediumSpeed = 0.7
lowSpeed = 0.5

while True:
    while distanceSensor.distance > 0.8:
        motorRight.backward(mediumSpeed)
        motorLeft.stop()
        time.sleep(0.02)
    print("I see the box")

    motorRight.stop()
    motorLeft.stop()
    time.sleep(2)

    if distanceSensor.distance > 0.8:
        print("accidentally overrun, correcting")
        while distanceSensor.distance > 0.8:
            motorRight.stop()
            motorLeft.forward(lowSpeed)
            time.sleep(0.02)
    else:
        motorLeft.backward(lowSpeed)
        time.sleep(0.4)

    motorRight.stop()
    motorLeft.stop()
    time.sleep(1)

    time1 = time.clock()
    print("First time:", time1)
    print("Pushing box")
```

```
while sensorCenter.value == 0:
    motorRight.forward(fullSpeed)
    motorLeft.forward(fullSpeed)
print("Line detected, stopping")
motorRight.stop()
motorLeft.stop()
time.sleep(0.5)
time2 = time.clock()
print("going back to original position")
motorRight.backward(fullSpeed)
motorLeft.backward(fullSpeed)
print("Second time:", time2)
time.sleep(time2 - (time1 + 0.3))
motorRight.stop()
motorLeft.stop()
time.sleep(0.5)

motorRight.backward(mediumSpeed)
motorLeft.stop()
time.sleep(0.5)
```

# Appendix G: Webpage and CGI scripts

## Website (index.html)

```
<html>
<head>
<script Language="Javascript">
function set_left()
{
    document.location="cgi-bin/set_left.cgi";
}
function set_right()
{
    document.location="cgi-bin/set_right.cgi";
}
function set_forward()
{
    document.location="cgi-bin/set_forward.cgi";
}
function set_reverse()
{
    document.location="cgi-bin/set_reverse.cgi";
}
function stop(event)
{
    document.location="cgi-bin/stop.cgi";
}
</script>
</head>


<body>
<div style="text-align:center">
  <h1>Raspberry Pi GPIO</h1>

  <img src="/arrow_forward.jpg" id="f" onpointerdown="set_forward()" onpointerup="stop()">
  <br>
  <img src="/arrow_left.jpg" id="l" onpointerdown="set_left()" onpointerup="stop()">
  <img src="/arrow_right.jpg" id="r" onpointerdown="set_right()" onpointerup="stop()">
  <br>
  <img src="/arrow_reverse.jpg" id="r" onpointerdown="set_reverse()" onpointerup="stop()">
  <br>
  <img src="/stop.jpg" id="s" onpointerdown="stop()" onpointerup="stop()">
  </div>
  </body>
</html>
```

## CGI script: set_left.cgi

```
#!/bin/bash

pigs m 18 w
pigs m 15 w
pigs m 12 w
pigs m 16 w
pigs w 18 0
pigs w 15 1
pigs w 12 1
pigs w 16 0

echo "Status: 204 No Content"
echo "Content-type: text/plain"
echo ""
```

## CGI script: set_right.cgi

```
#!/bin/bash

pigs m 18 w
pigs m 15 w
pigs m 12 w
pigs m 16 w
pigs w 18 1
pigs w 15 0
pigs w 12 0
pigs w 16 1

echo "Status: 204 No Content"
echo "Content-type: text/plain"
echo ""
```

## CGI script: set_forward.cgi

```
#!/bin/bash

pigs m 18 w
pigs m 15 w
pigs m 12 w
pigs m 16 w
pigs w 18 1
pigs w 15 0
pigs w 12 1
pigs w 16 0

echo "Status: 204 No Content"
echo "Content-type: text/plain"
echo ""
```

## CGI script: set_reverse.cgi

```
#!/bin/bash

pigs m 18 w
pigs m 15 w
pigs m 12 w
pigs m 16 w
pigs w 18 0
pigs w 15 1
pigs w 12 0
pigs w 16 1

echo "Status: 204 No Content"
echo "Content-type: text/plain"
echo ""
```

## CGI script: stop.cgi

```
#!/bin/bash

pigs m 18 w
pigs m 15 w
pigs m 12 w
pigs m 16 w
pigs w 18 0
pigs w 15 0
pigs w 12 0
pigs w 16 0

echo "Status: 204 No Content"
echo "Content-type: text/plain"
echo ""
```