

Project Report

Introduction	2
Title	2
Objectives	2
Car distance scanning	2
Getting commands from access point	2
Material and Methods	3
Analog diagram	3
UDP	3
H-Bridge	3
NodeMCU	3
Arduino code	4
PWM	6
Ubuntu server	6
Results	7
Car Motor	7
Current	8
Voltage	9
UDP Latency	9
Car Velocity	9
Conclusion	9
References	9
Acknowledgements	10

Introduction

As young engineers that travel daily with cars we realised that our current system is slowly degrading and becoming outdated. We realised that the current growth of population and means of transportation only leads to more and more problems on the road. To solve this problem we thought of a way to take away control of the humans driving. Not only could this solve the problem of human error but it could also solve problems like traffic jams. We created cars that are capable of braking if they get too close to another object which in our case will be another car. The cars will be able to output all their data over Wifi to a nearby server. They will also be able to receive data from servers so they can be forced to brake even though they have no incentive of their own.

Goal

We built a miniaturized version of what we have in mind for the real world. To make this possible in the real world we will have to equip every car with communication capabilities. The best type of service for this would be satellite communication to cover more area at once. We can use sonar in combination with LIDAR to cover short range and long range distances around the car.

Let's take for example the E19, this highway is frequently jammed because of too many cars driving home. Our servers will detect this type of jamming and will create ways for cars to reach their destination as soon as possible. This doesn't mean that there won't be jams anymore. It simply means that they will be home as soon as possible.

That is one of the many advantages a system like ours can have. Another advantage is the fact that there is no human in control anymore. Every year nearly 1.3 million people die every year by car accidents. This will all be solved by simply letting software control the car.

Objectives

Car distance scanning

The car should be able to scan the distance between himself and the vehicle in front so that it can adjust his speed.

Getting commands from access point

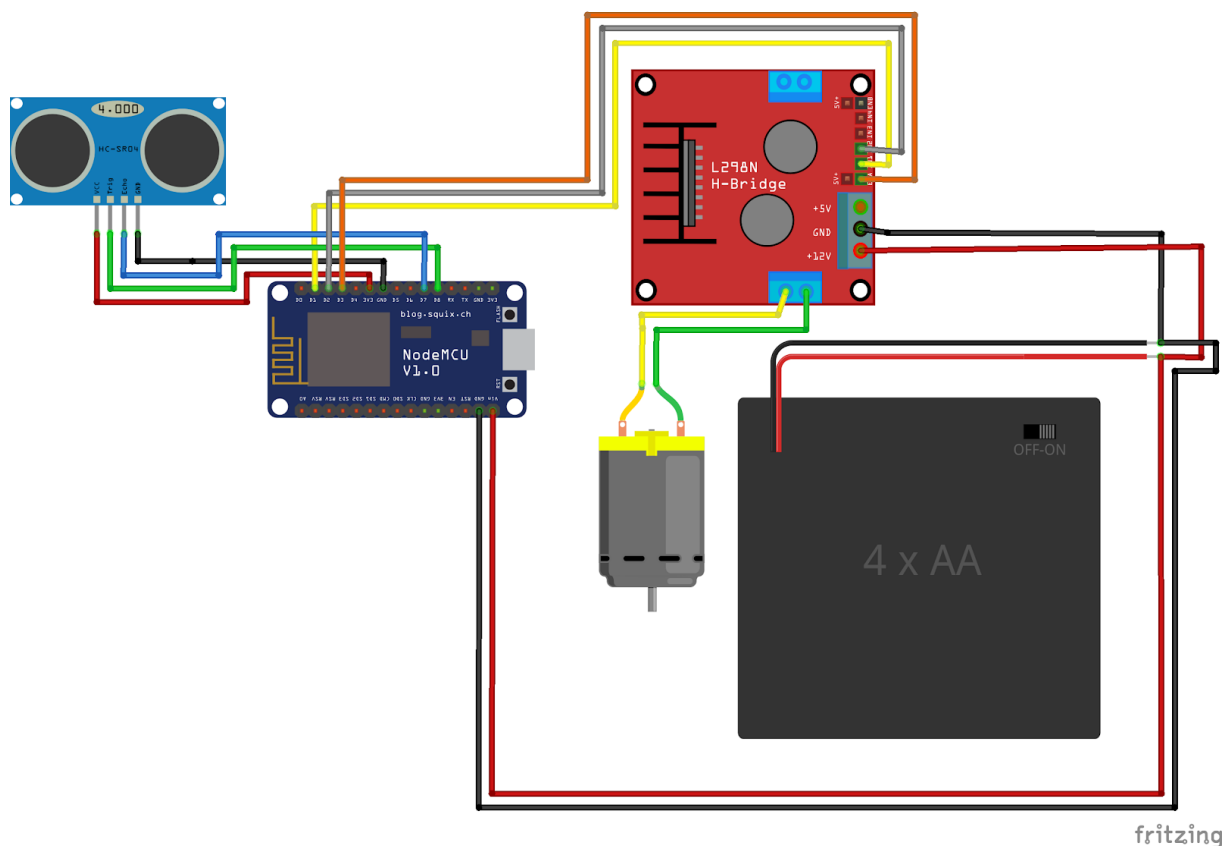
Material and Methods

Analog diagram

The analog diagram is a representation of the entire circuit, it contains all components implemented in the circuit. This diagram can visualize what we are actually doing and it is universal understandable, other engineers that don't understand this language can still figure out what we made by looking at the analog diagram.

The analog diagram contains different components of which the working will be discussed below. The components are:

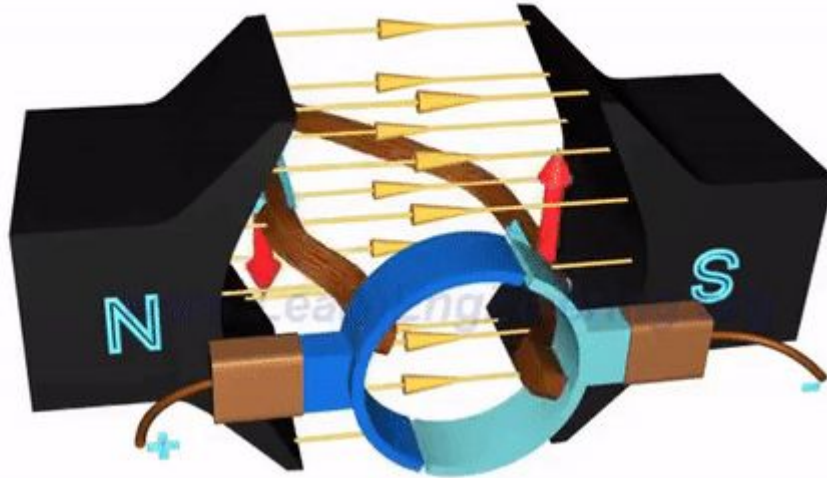
- The NodeMCU: small microcontroller with ESP8266 wifi module implemented to send/receive packets through network
- L298N: H-Bridge/Motorshield that connects microcontroller and DC-motor to correctly drive the DC-motor
- Battery pack: contains 4 x AA 1.5V batteries with a on/off switch to make a DC-Voltage source of 6V
- HC SR04: distance sensor that sends a wave and catches up the reflected wave so it can calculate the distance



DC-Motor

A DC-motor is a motor that can deliver power from a DC-Voltage source. The motor contains several parts: The stator, the rotor, commutator, DC-Voltage source. The stator creates a constant magnetic field inside the motor, this can be done by permanent magnets or by a electromagnet that is created from the same DC-Voltage source. The rotor contains coils that are inside the magnetic field of the stator so that according to the Lorentz law, a force will be applied on the coil which results in

rotation. The coils are bound to the commutator rings. As the coils rotate, the commutator rings connect to the power source of opposite polarity so that the coils will spin in the same directions. By adding more coils the motion of the rotor will be more smooth.



UDP

UDP technology is known for its faster speed of transmission when compared to TCP. However there are some limitations: with UDP it is not guaranteed that the packets will be received by the receiver. For our project we have concluded that speed of transmission is more significant than assuring 100 % of packet delivery.

But after the choice was made there were some experiments needed to be done i.e. how many percentage of the packets are actually received on the receiver?

For this experiment there are two options: one-way transmission and two-way transmission.

One-way transmission

For this mechanism to work both the sender and receiver need to be synchronised to a highly precise clock source such as atomic clock, high frequency radio clock etc. Therefore this method is a little inconvenient than the other one.

two-way transmission

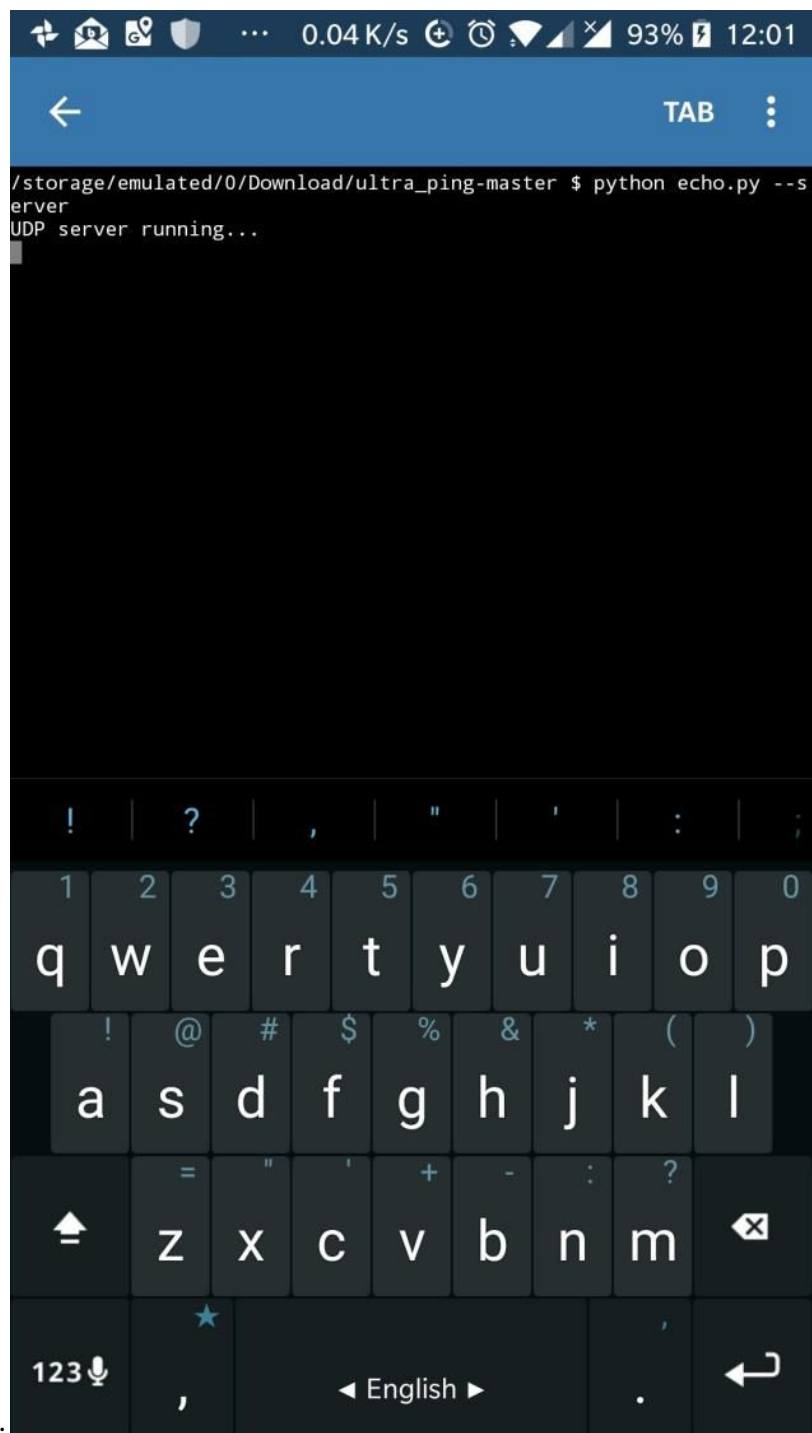
In this way of transmission the sender tracks the time of sending a packet e.g. t_0 . The packet is received by the other side and a similar packet is sent backwards to the sender. The sender notes the time of arriving as t_1 . The two-way transmission time would be then $t_1 - t_0$.

For this experiment we have found a great source of code to do exactly what is mentioned in the two-way transmission method. Here is the link to the code:

https://github.com/mrahtz/ultra_ping

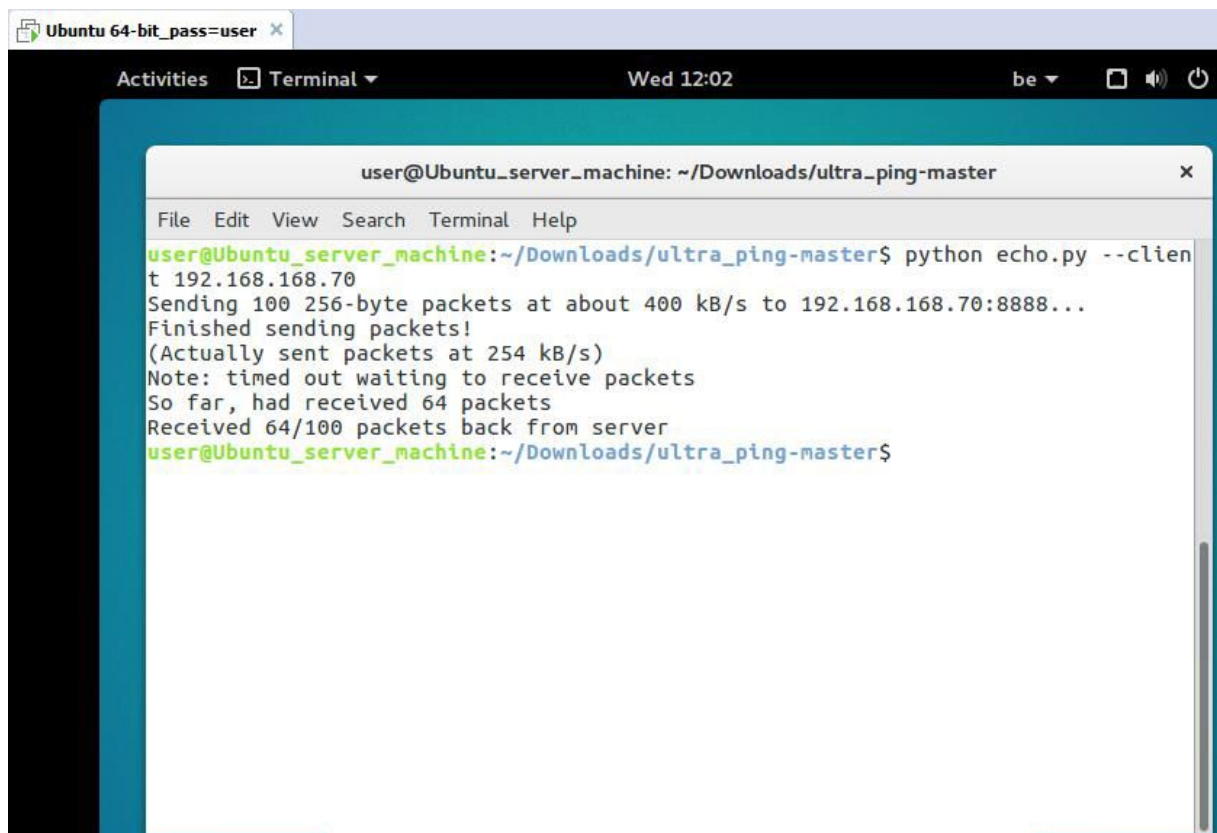
However this code needs to be run on Python programming language which is already available on Ubuntu server operating system. In case of our nodeMCU we have to integrate Python into the firmware which is not suitable for a light-weight hardware chip. Therefore we put the repository on

an android device where an app called [Pydroid](#) was installed. The code is run on both Ubuntu server and android device while being on a same Wifi network.



Client-side:

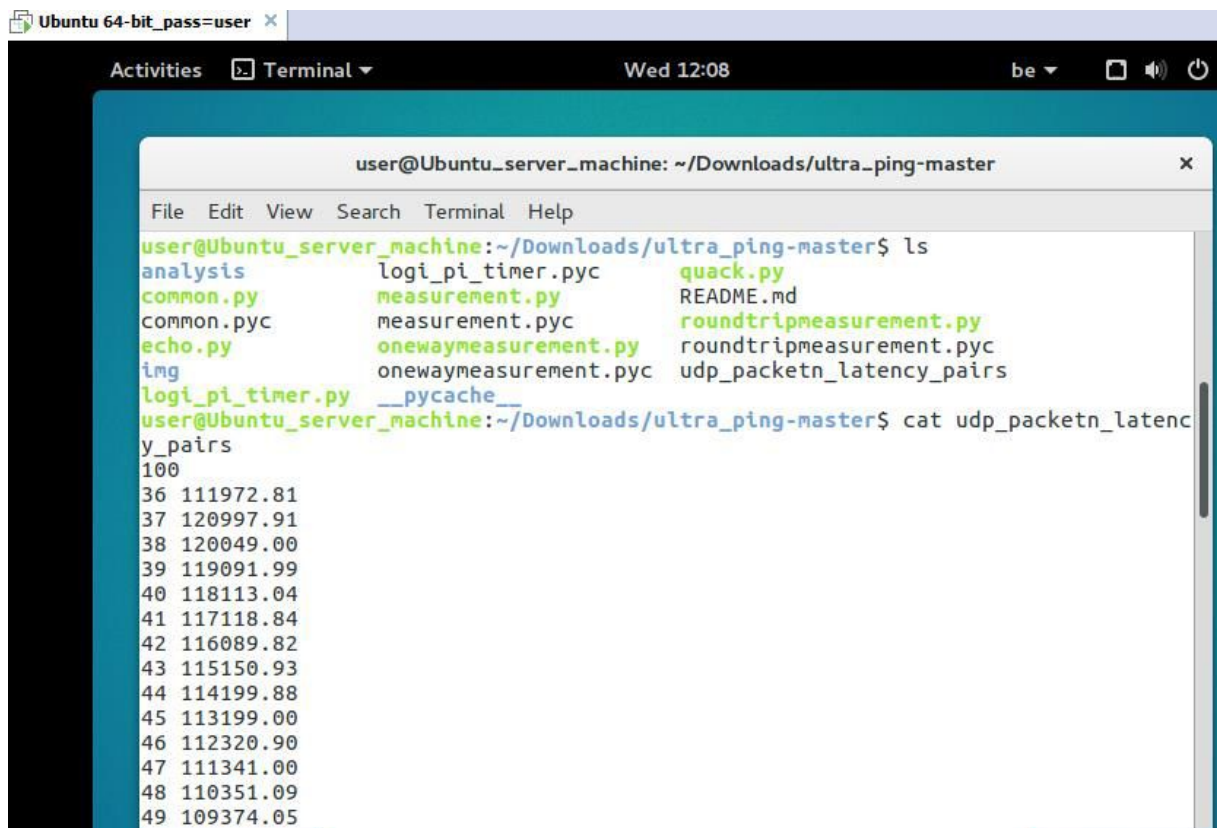
Server-side:



A terminal window titled "user@Ubuntu_server_machine: ~/Downloads/ultra_ping-master". The window shows the execution of a Python script named "echo.py" with the command "python echo.py --client 192.168.168.70". The output of the script is as follows:

```
user@Ubuntu_server_machine:~/Downloads/ultra_ping-master$ python echo.py --client 192.168.168.70
Sending 100 256-byte packets at about 400 kB/s to 192.168.168.70:8888...
Finished sending packets!
(Actually sent packets at 254 kB/s)
Note: timed out waiting to receive packets
So far, had received 64 packets
Received 64/100 packets back from server
user@Ubuntu_server_machine:~/Downloads/ultra_ping-master$
```

The result is shown as below:



A terminal window titled "user@Ubuntu_server_machine: ~/Downloads/ultra_ping-master". The window shows the execution of the "ls" command, listing the contents of the directory. The output is as follows:

```
user@Ubuntu_server_machine:~/Downloads/ultra_ping-master$ ls
analysis          logi_pi_timer.pyc      quack.py
common.py         measurement.py         README.md
common.pyc        measurement.pyc        roundtripmeasurement.py
echo.py           onewaymeasurement.py  roundtripmeasurement.pyc
img              onewaymeasurement.pyc  udp_packetn_latency_pairs
logi_pi_timer.py  __pycache__
user@Ubuntu_server_machine:~/Downloads/ultra_ping-master$ cat udp_packetn_latency_pairs
100
36 111972.81
37 120997.91
38 120049.00
39 119091.99
40 118113.04
41 117118.84
42 116089.82
43 115150.93
44 114199.88
45 113199.00
46 112320.90
47 111341.00
48 110351.09
49 109374.05
```

...

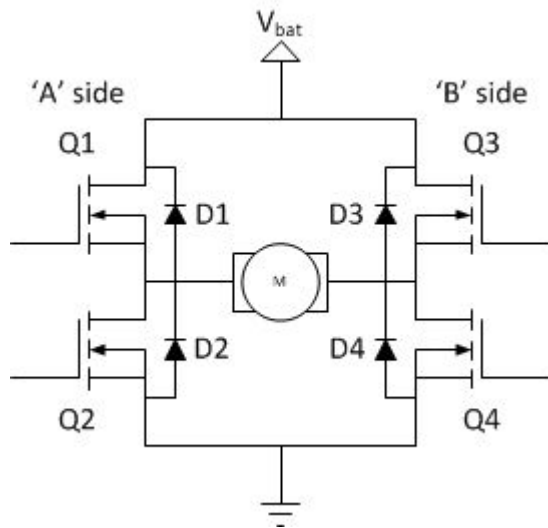
The first column shows the index of the packet which passed the test. The second column shows the transmission time in microseconds. Of the 100 packets sent simultaneously only 64 of them successfully travelled between the server to client and vice versa. This might seem significant but our nodeMCU will send a single packet in a given time interval. Therefore this method seems to be working as expected.

The average one-way travel speed was calculated to be 53 milliseconds.

H-Bridge

In general an H-bridge is a rather simple circuit, containing four switching element, with the load at the center, in an H-like configuration:

The switching elements (Q1..Q4) are usually bi-polar or FET transistors, in some high-voltage applications IGBTs. Integrated solutions also exist but whether the switching elements are integrated with their control circuits or not is not relevant for the most part for this discussion. The diodes (D1..D4) are called catch diodes and are usually of a Schottky type.

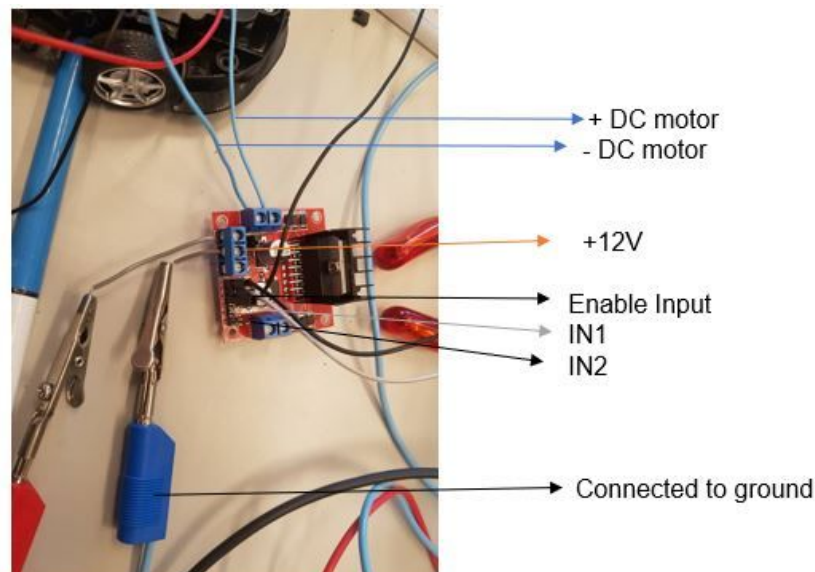
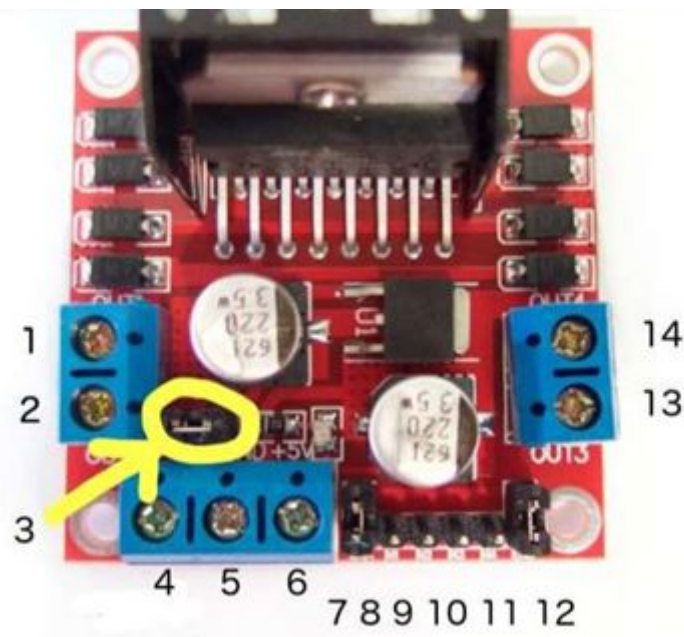


The top-end of the bridge is connected to a power supply (battery for example) and the bottom-end is grounded.

In general all four switching elements can be turned on and off independently, though there are some obvious restrictions.

Though the load can in theory be anything you want, by far the most pervasive application of H-bridges is with a brushed DC or bipolar stepper motor (steppers need two H-bridges per motor) load. In the following I will concentrate on applications as a brushed DC motor driver.

Our H-Bridge The L298N



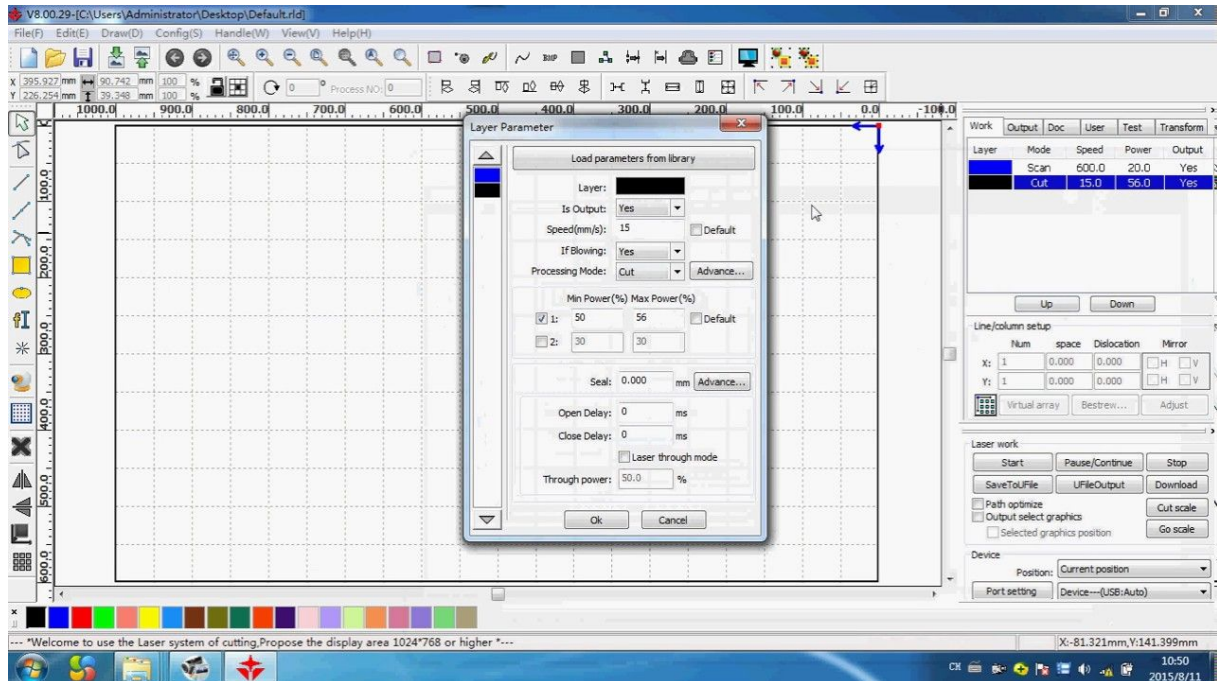
1. DC motor 1 "+" or stepper motor A+
2. DC motor 1 "-" or stepper motor A3.
3. 12V jumper - remove this if using a supply voltage greater than 12V DC.
This enables power to the onboard 5V regulator
4. Connect your motor supply voltage here, maximum of 35V DC.
Remove 12V jumper if >12V DC
5. GND
6. 5V output if 12V jumper in place, ideal for powering your Arduino (etc)
7. DC motor 1 enable jumper. Leave this in place when using a stepper motor.
Connect to PWM output for DC motor speed control.
8. IN1
9. IN2
10. IN3
11. IN4
12. DC motor 2 enable jumper. Leave this in place when using a stepper motor.
Connect to PWM output for DC motor speed control

Laser Cutting

In this chapter we will discuss how we cut our shapes from the plexiglas. First of all the reason why we use plexiglas is because it is a decent material that does not break fast and it is see through. It is a lightweight material which will not have a big impact on the car's weight.

Software

For drawing our shape to cut the material we use the software MetalCut.



In the MetalCut software you have to make a file in .rd extension , the laser cutter can only read these files. You can choose if you want to cut through the material or just engrave it into the material. That can be done by selecting different colours for different layers. Each layer can be adjusted at what speed the laser will go and how much power will be used. Distance between 2 figures can be adjusted in the toolbar above. In the left toolbar you can select different figures. When selecting a figure you can adjust the height or width of the figure. When you finished your shapes, you can simulate how the laser will print it by pressing the preview button. Always make sure the holes are big enough to fit the screws. When the drawing is complete don't forget to convert the file into the .rd extension. That can be done in the right bottom corner by pressing the save to file button. Use a USB-stick to transfer the file over to the laser cutter.

Laser cutter

Our university provides us with the “BRM laser” lasercutter. To start the machine use the main



switch which can be found at the right side of the machine. After it has boot up, insert the USB-stick.

On the laser cutter screen navigate to file -> read from USB, select the file you want to insert into memory and write it into memory. Then go back to file and select it, the preview will be shown on the screen.

Insert your material frame and adjust the position of the laser. That can be done by moving the laser with the arrow keys and pressing the origin button for changing its starting point. You can always check where the laser cutter will cut by pressing the frame button. That shows the outter lines of the preview screen.

With the pulse button you can check how strong the laser will be and at which spot it is on your material. Adjust the right height of the laser(Z-axis movement). That is important for cutting through the material. Make sure that the cooling system is activated and the ventilation is on or else there will be smoke coming out of the laser and a bad smell will fill the room. If everything is set correctly you can start by pressing the start button.

When the shapes are cut you can open the machine and take your shapes.

NodeMCU

In this chapter we will discuss which microcontroller we chose and why. We will start with a brief introduction of microcontrollers. Then we will compare key features of different microcontrollers and finally we will conclude why we chose the NodeMCU

Microcontrollers

A microcontroller is actually a tiny computer on a single integrated circuit. It contains the microprocessor, which does all the processing of the chip, and the memory where all the data is stored. It also contains the peripherals needed and the right connections to the pins of the processor to establish a more convenient way of working with a microprocessor without needing complicated setups or much brain power.

Microcontrollers are typically quite small and cheap, low power consumption components mostly used for controlling a specific part of a electronic setup, like a LED-display, using sensors,...

We will be using these to control our rc cars and make them more 'intelligent' by adding sensors and a wireless communication to a server over WiFi.

Which microcontrollers

As we are using WiFi, we need a microcontroller which supports this. The most famous MCU's are Raspberry Pi and Arduino. These do both not come with a WiFi-feature but a WiFi shield could be bought. We discovered that these shields use a 'ESP8266' wifi chip and after a bit of research we found a cheaper alternative which integrates the 'ESP8266' as the main processor, the NodeMCU.

Arduino and Raspberry pi are so famous for a couple of reasons. They are an all-round MCU with a lot of 'shields' (modules) to achieve all kinds of goals. There is also very much documentation and test projects to find. Another reason is the good stats. They have a lot of memory, fast processor, contain a lot of built-in peripherals etc..

The disadvantage is that all of this comes with a price. This is where the NodeMCU comes in

NodeMCU

The NodeMCU is a MCU which uses the ESP8266 microprocessor built by 'Espressif', a Shanghai based electronics company. It is widely used in microcontrollers because it's cheap and has built-in WiFi connectivity.

The name 'NodeMCU' stands for the microcontroller itself as well as the firmware programmed on the ESP8266. It is fully open-source on GitHub and already contains over 40 modules.

The production of the official NodeMCU microcontroller has stopped but there are a lot of spinoffs with exactly the same layout which can easily be found.

The firmware is written in the 'lua' language but other language firmwares can be flashed on the NodeMCU. We will be using Arduino IDE because of its widespread use and documentation .

Hardware

The nodeMCU can be powered with a micro-usb connector or with the Vin pins, which requires 5v up to a maximum of 10V. It drains around 80-90mA of current in normal usage mode. Each pin has a voltage of 3.3V and can drain an additional 15mA from the power source if connected and set as output. Sending packets through WiFi can drain up to an additional 150mA.

At startup the current can peak up to around 300mA but this is very short and not to worry about when using batteries.

The NodeMCU has 16 GPIO pins with 11 of them being normal digital I/O pins which we will use. To flash the NodeMCU, GPIO pin 0 has to be pulled down, which can be done by connecting it to the

ground with a resistor to limit the current.
All of these pins also have a PWM functionality.

Software

As we have programmed the NodeMCU with the Arduino IDE firmware, we can use that to program the processor. How Arduino works will be discussed in another topic.

Arduino

Arduino is the code environment we will be using to program the microprocessor. It is very easy to use and very documented on the arduino.cc website. A lot of examples can be found online to guide us into coding with Arduino.

Programming happens through a micro-usb connector and once the code is programmed on the processor, powering it on without the micro-usb will run the programmed code.

A arduino program contains two main sections; a 'setup()' method which will run only once at startup and a 'loop()' method which will continuously run while the microcontroller is turned on.

The code

Here we will briefly look into the functionality of the code. The code itself will be added in an attachment and will be commented enough to understand how it works.

The goal is to connect to an ubuntu server, configure the right pins for the used peripherals and then enter a loop which will:

- Check the distance gap to the car in front.
- Adjust the car's acceleration to keep a constant distance.
- Check if the server has sent a command
- Send the distance and the acceleration to the server.

Adjust car's acceleration

There are a couple of methods to adjust the acceleration of the car. We will briefly sum up the different methods and conclude which we picked and why.

- Feedback-Loop : The distance measured is compared to a fixed distance which the cars need to attain. The offset is multiplied by a constant and added (or subtracted if the offset is negative) to the current duty cycle.

Pro: Not required to add a fixed or reference duty cycle as the value will automatically reach an equilibrium.

Cons: The multiplier constant needs to be a balance between reaction speed and accuracy. A high constant will cause the car to better react to drastic changes like an obstacle but will fluctuate hard at a constant duty cycle. Also sensitive to errors. A low constant will give better accuracy for a constant or slow change, but will react slowly to a drastic change in distance.

- Fixed duty cycle: At different distances, the duty cycle will have different, fixed values.
e.g.: The goal distance of 100cm is equal to the duty cycle of 640. If the distance lowers to 90cm, the duty cycle will change to 576.

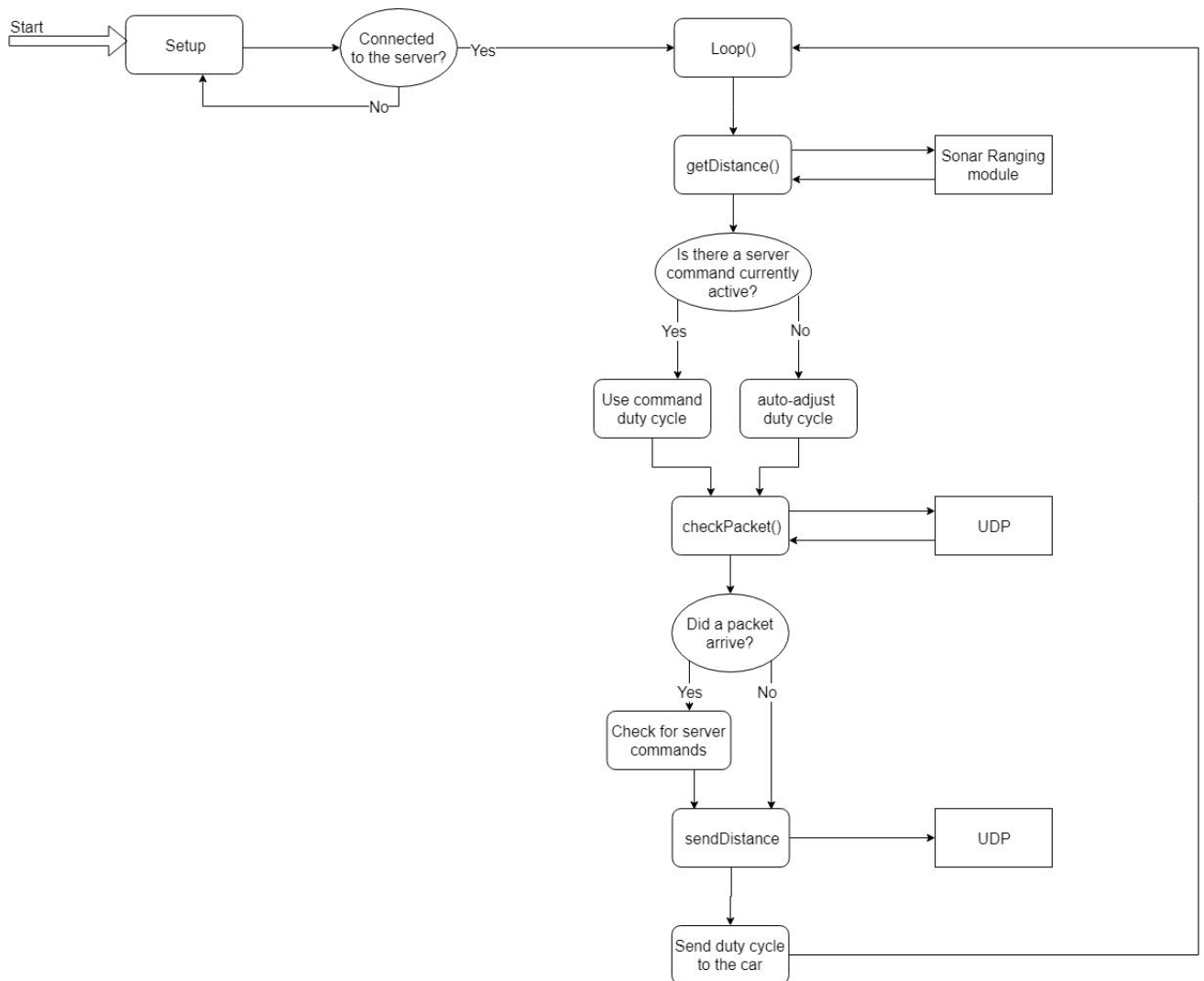
Pro: Speed and acceleration relative to a duty cycle does not need to be known. Less chance for errors,

Cons: not as accurate as the feedback loop as we work in specific intervals.

We picked the fixed duty cycles. The reason being that the feedback-loop had some problems like causing overflows when one faulty measurement of e.g. 13m was measured. Our tests showed that this method was reliable enough so we went with this.

Flowchart

How the program works is visualised with this flowchart. The ellipses are conditional statements, the rounded squares are functions or actions and the sharp-edged squares are peripherals.



PWM

We have talked about the acceleration of the car and adjusting it with a microcontroller, but how does this work?

The speed of a motor is relative to the DC-voltage it receives. The higher voltage it gets, the faster it will spin and the greater our acceleration will be.

The problem is that a microcontroller works digitally. The pins can only send out a HIGH(3.3V) or a LOW(0V).

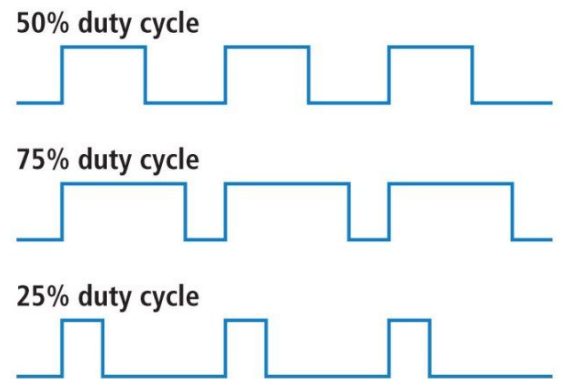
One solution to achieve a (pseudo) analog signal is to very rapidly turn a pin on and off. This results in a signal which is a square wave of HIGH and LOW, but acts like a DC signal of a voltage defined by the amount of HIGH in one period.

The percentage of HIGH on a period is called the duty cycle. A duty cycle of 10% means that the signal will be high 10% of the time and low 90% of the time, resulting in 1/10th of the HIGH voltage, or 0.33V.

This method is called Pulse Width Modulation.

As the NodeMCU has 12 bits connected to its PWM -output, it can distinguish between 1024 different duty cycles, where 1023 stands for 100%, 512 for 50% etc..

In practice we will be sending this signal to the enable of the H-bridge, which acts like a switch between the power source and the DC motor.



Distance measurement

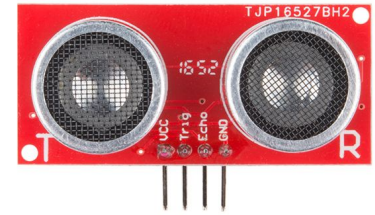
We have to determine the distance of the car in front. There are three ways to calculate this.

- Ultrasonic ranging:
 - Pro: Cheap and easy to use, very reliable up to 2m. Can be connected directly to microcontroller
 - Con: Less fool-proof. Has a big operating range which can cause interference of irrelevant objects
- Laser (LIDAR) ranging:
 - Pro: Very accurate, larger range
 - Cons: VERY expensive
- calculating the position of each car using a start position and speed measurements and sending it to the server, which sends the offset of each car to all the cars
 - Pro: The server knows the position of each car
 - Cons: Large margin for error, intensive server - mcu communication, slow reaction time

Conclusion: We went with the ultrasonic ranging module as this is perfect for this project. We will not need a large range and it is easy to control and work with. We will use the Sparkfun HC-SR04.

HC-SR04 Sonar ranging module

The HC-SR04 is a component which uses ultrasonic waves to determine the distance of the nearest object by calculating the time it took for the ultrasonic waves to bounce off that object back onto the HC-SR04.



Aside to power and ground, it has two pins, a trig and an echo pin.

When a HIGH is put on the trig-pin for 10 microseconds, it will send out 8 ultrasonic wave pulses on one side and start sensing on the other side.

Once that ultrasonic wave is reflected back to the sensing side, the echo-pin will send out the duration.

This can be converted into distance by multiplying it by the speed of sound, 340m/s or 0.034m/ μ s.

As this is the time it took for a round trip, this has to be divided by 2 to get the distance of the object.

Normally, this component needs 5V working current, which our MCU cannot provide. Experiments showed that this is not necessary and it works perfectly provided with a 3.3V current.

More information on this in the Experiments paragraph.

Ubuntu server

In order to provide a smooth flow of the traffic we have decided to use a server to maintain that. There is a great pool of servers to choose from and each one has its own features and customizations. In our case we concluded to use an Ubuntu server for the following reasons:

- It is open-source and one of the best platforms for developers and learners to work with.
- It has stable releases so the reliability is of course a necessity to consider for a project like ours.
- Ubuntu has huge community support so in case of any issues the support is there.
- It has graphical user interface which is convenient for newbie Linux users.
- It is Linux based so all the powerful tools on Linux infrastructure are available there.
- The server is lightweight, it only needs at least 1.5 GB of HDD space to install it.

How to install Ubuntu server

There are a few ways to install the server software.

1. Standalone server: In this method the Ubuntu server would be the only operating system installed on our laptops or PCs. This method should only be considered if we intended to use Ubuntu server for our personal use instead of the Windows variants that we currently have. Unless we provide an extra machine for the project other than our personal laptops.
2. Multi-booting: This mode will put the Ubuntu server besides our current Windows operating system. We could choose to start any of the two operating systems every time the laptop or PC is turned on. This option is also to be considered but since we would want to use Ubuntu server only for this project so we need to find a better alternative.
3. Hypervisor: This option is highly customizable. It makes a layer on top of our current operating system upon which the Ubuntu server would be installed. We could allocate any amount of available hardware resources that we have. For example, we can allocate more processors, RAM, network cards etc. to the Ubuntu server.

For our project we have chosen to install Ubuntu server on a freeware hypervisor software

like VMware Workstation Player. This option has a limitation however. Our laptops have only one wireless network card which is used by host operating system like Windows. The internet connection of this card is shared with our guest operating system (Ubuntu server). In order for the nodeMCU to communicate directly with Ubuntu server we have to provide an extra network card (see the figure below).

How the nodeMCUs communicate?

For the nodeMCUs to communicate directly with our server we have to make a wireless access point for the chips to connect to. The details of how to make a wireless hotspot have been written in our preparation report of 4th week.

How the server gets data from nodeMCU?

There are a few options to consider. There is a great tool in Linux called netcat. Basically it listens to incoming packets on a chosen port on TCP or UDP platform. This command is manipulated for the project so that the incoming packets from nodeMCUs are stored in a text file called "input.txt". Since we have multiple nodeMCUs, we might want to listen on different port numbers to distinguish which packet was received from which nodeMCU. This could also mean that we use different input.txt files for each nodeMCU. Each file should be stored in a different location.

To make things easier we have also configured a DHCP server on our Ubuntu operating system to provide each nodeMCU with a permanent unique IP address. Please refer to the preparation report of 9th week for more details.

How the server makes a reply?

Based upon the data the server receives in the input.txt file. The server analyses the data and makes a suitable reply. The analysing part is being discussed in the preparation report of 9th week. For this part we have used some java code in combination with the Linux commands to work.

Results

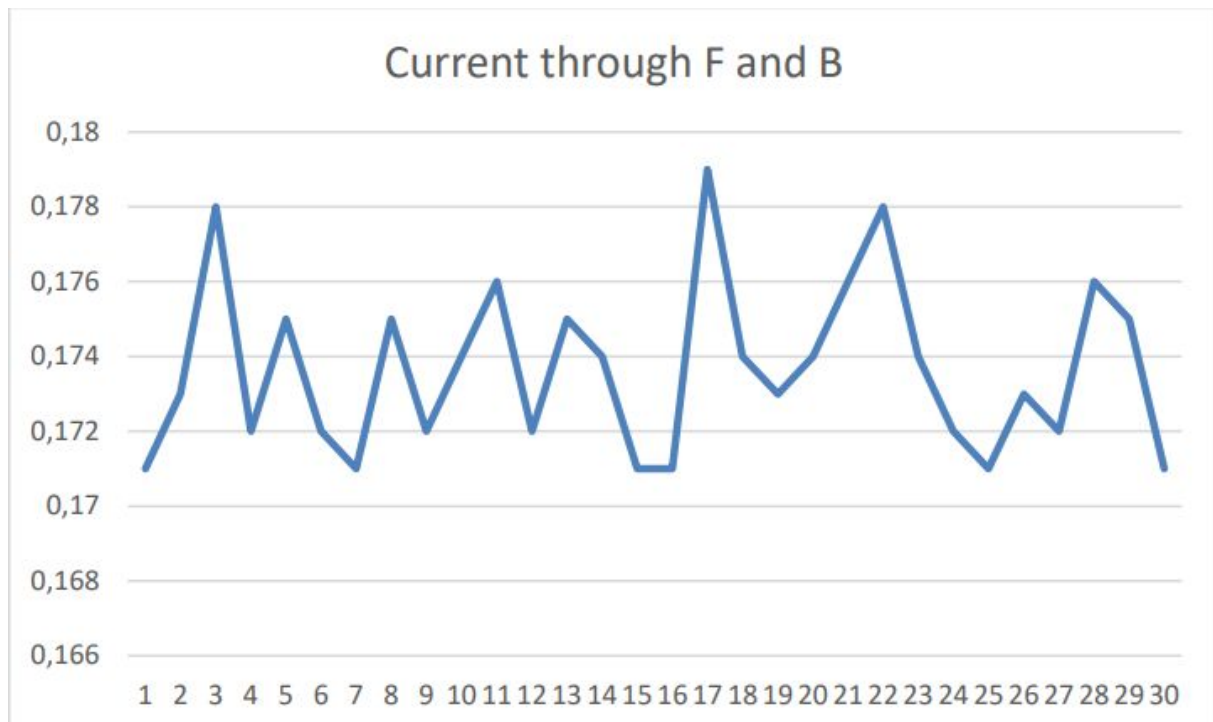
Car Motor

For the car motor we made following measurements:

Current door F en B	Volt over F en B
0.171	4.17
0.173	4.18
0.178	4.176
0.172	4.171
0.175	4.181
0.172	4.174
0.171	4.185
0.175	4.182
0.172	4.19
0.174	4.118
0.176	4.172
0.172	4.171
0.175	4.159
0.174	4.165
0.171	4.169
0.171	4.16
0.179	4.161
0.174	4.173
0.173	4.169
0.174	4.16
0.176	4.162
0.178	4.171
0.174	4.16
0.172	4.155
0.171	4.153
0.173	4.162
0.172	4.161
0.176	4.157
0.175	4.159
0.171	4.156
AMPERE	VOLT

Lets discuss each part.

Current



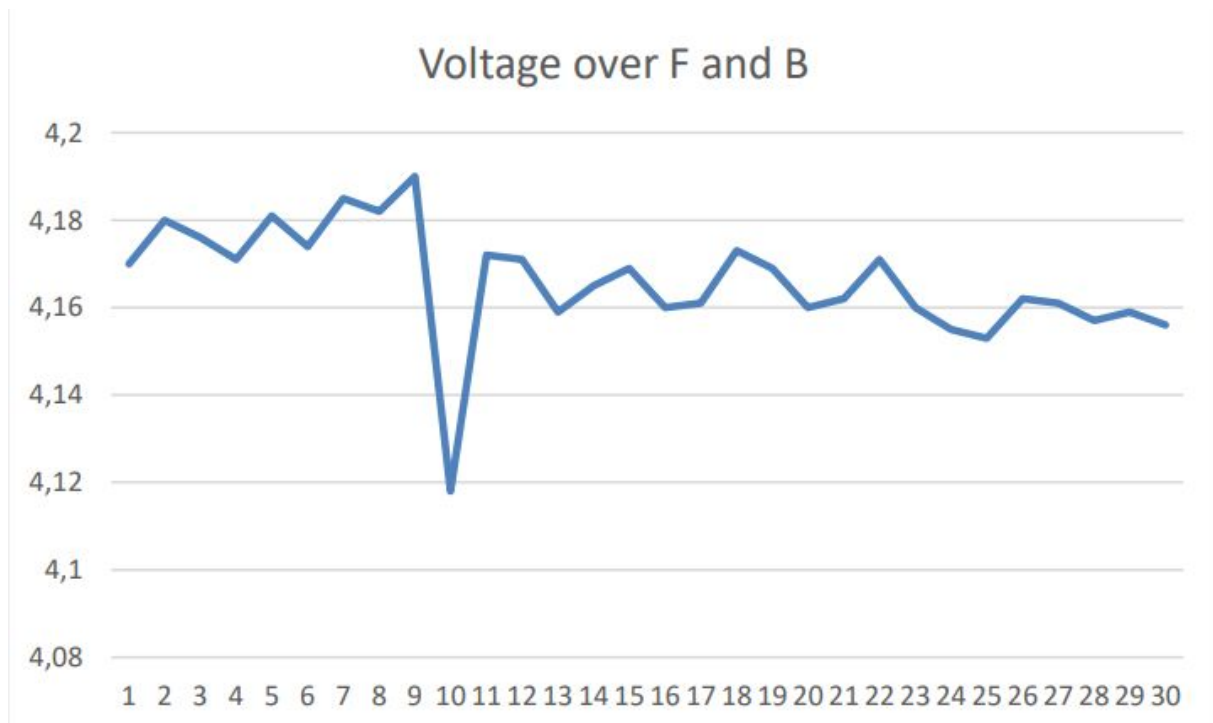
Standard deviation of the sample	0.0023
Mean of the sample	0.1737
Variation of the sample	0.0000051954

The standard deviation of this sample is 0.0023 ampere which means the values differ from each other approximately 2,3 mA. The variation also measures the spreading of the values, in this case the amperes. The value is small because the amperes are very close to each other, they only differ in a factor 10^{-3} or few mA.

99% Confidence interval of the mean	$0,1725A < I < 0.1748A$
99% Confidence interval of the variation	$5,50 * 10^{-8} A^2 \leq \sigma^2 \leq 8,75 * 10^{-7} A^2$
99% Confidence interval of the standard deviation	$2,35 * 10^{-4} A \leq \sigma \leq 9,35 * 10^{-4} A$

These Confidence intervals show that 99% of the values of our measurements have a mean, variation, standard deviation that is between these intervals.

Voltage



In this graph we can see there is an outlier voltage at the 10th measurement, that might have been caused by a fault during the measurement.

Standard deviation of the sample	0.0132
Mean of the sample	4,1661
Variation of the sample	$1,7365 \cdot 10^{-4}$

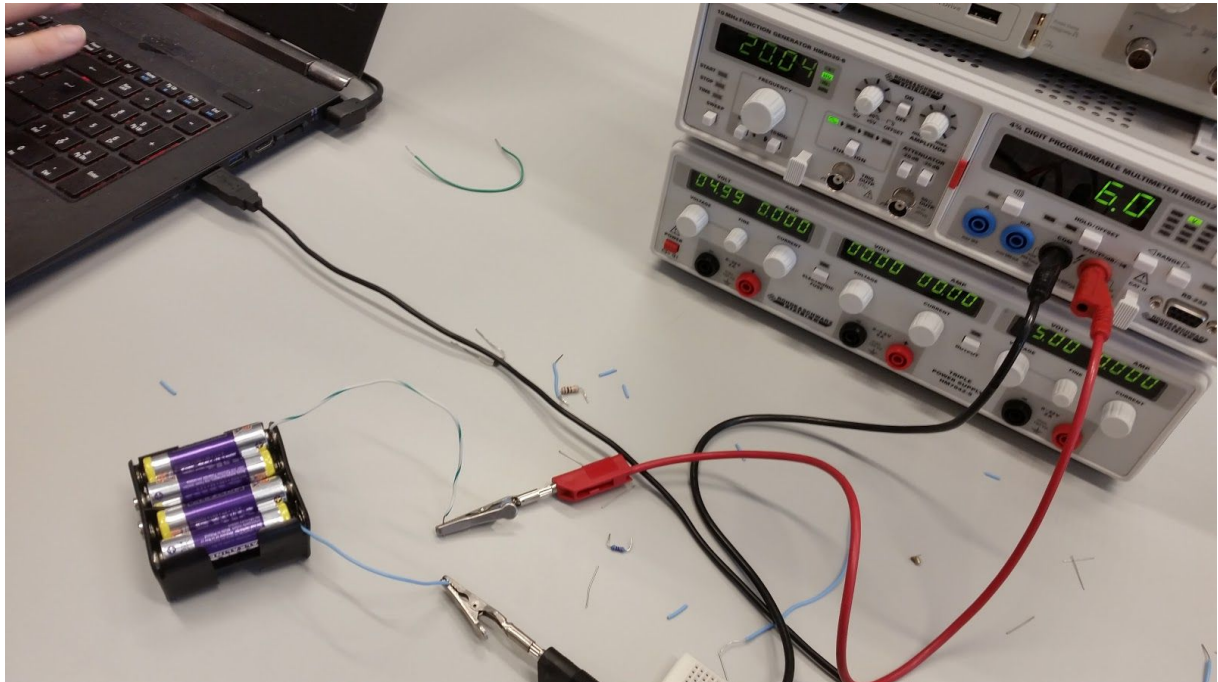
The standard deviation is 0.0132V, so the values measured are spread with approximately 0.0132V. The variation again measures as well the spreading of the measurements.

99% Confidence interval of the mean	$4,1594 V < U < 4,1727 V$
99% Confidence interval of the variation	$1,84 \cdot 10^{-6} V^2 \leq \sigma^2 \leq 2,93 \cdot 10^{-5} V^2$
99% Confidence interval of the standard deviation	$0.0014 \leq \sigma \leq 0.0054 V$

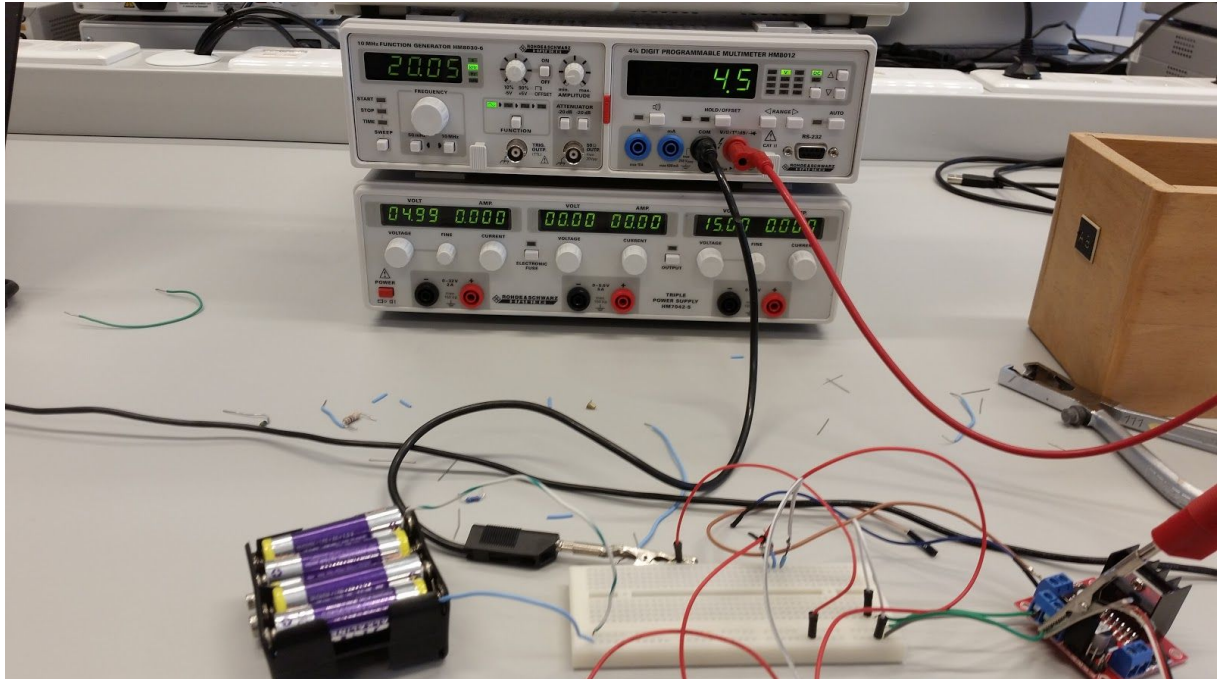
These confidence intervals tell us that 99% of the measurements have a mean, variation, standard deviation shown by these intervals.

Battery

The battery pack on the car consists of 4 x AA 1.5V batteries which adds up to 6V in total.

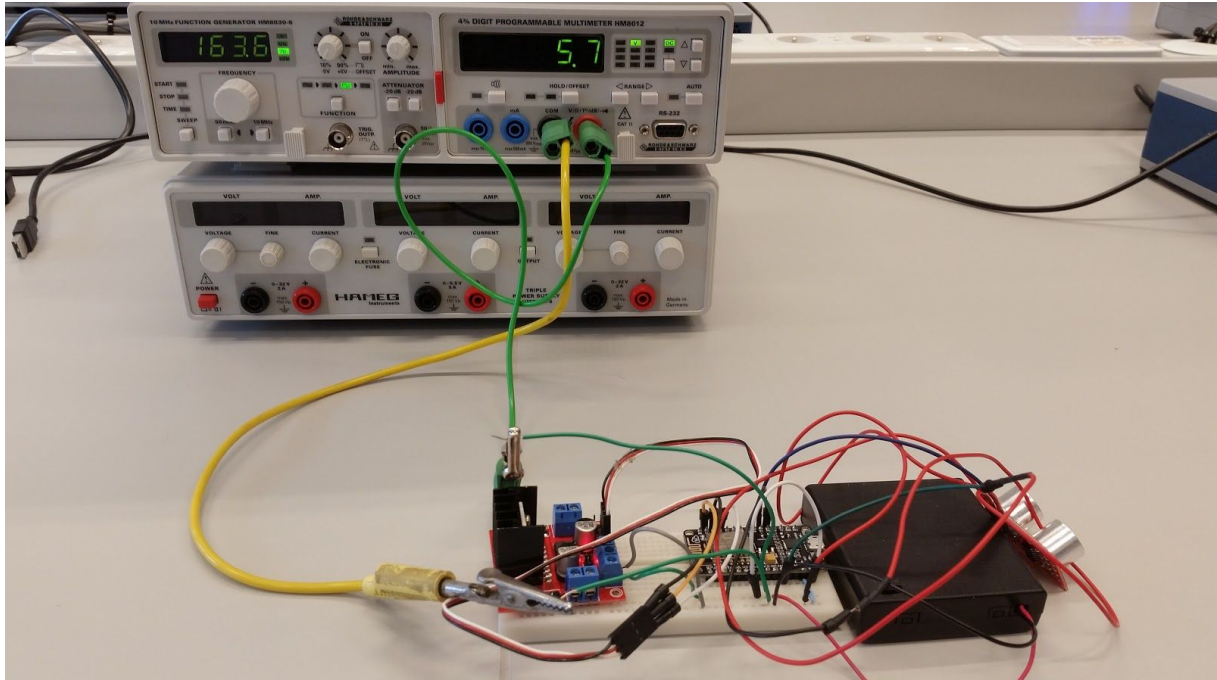


We can power the NodeMCU in different ways. The first way we tried was to use the Vcc output from the H-bridge which gives us 4.5V, and it can go up to a 5V when applying more source voltage on the H-Bridge.

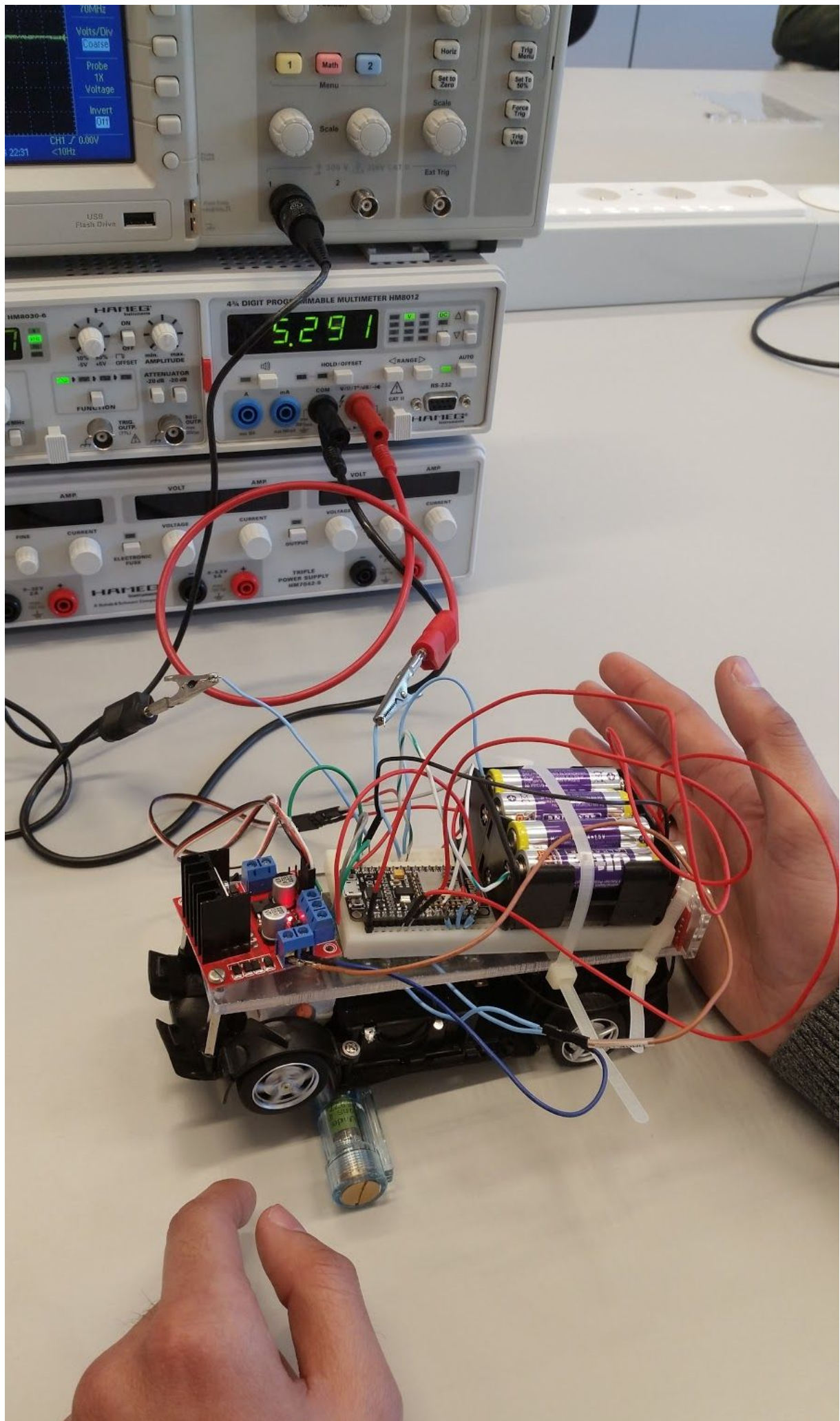


After implementing the 4.5V on the NodeMCU, when the circuit was turned on the voltage dropped to a 4.1V which was insufficient voltage to generate a good pwm enable signal for the H-bridge. It

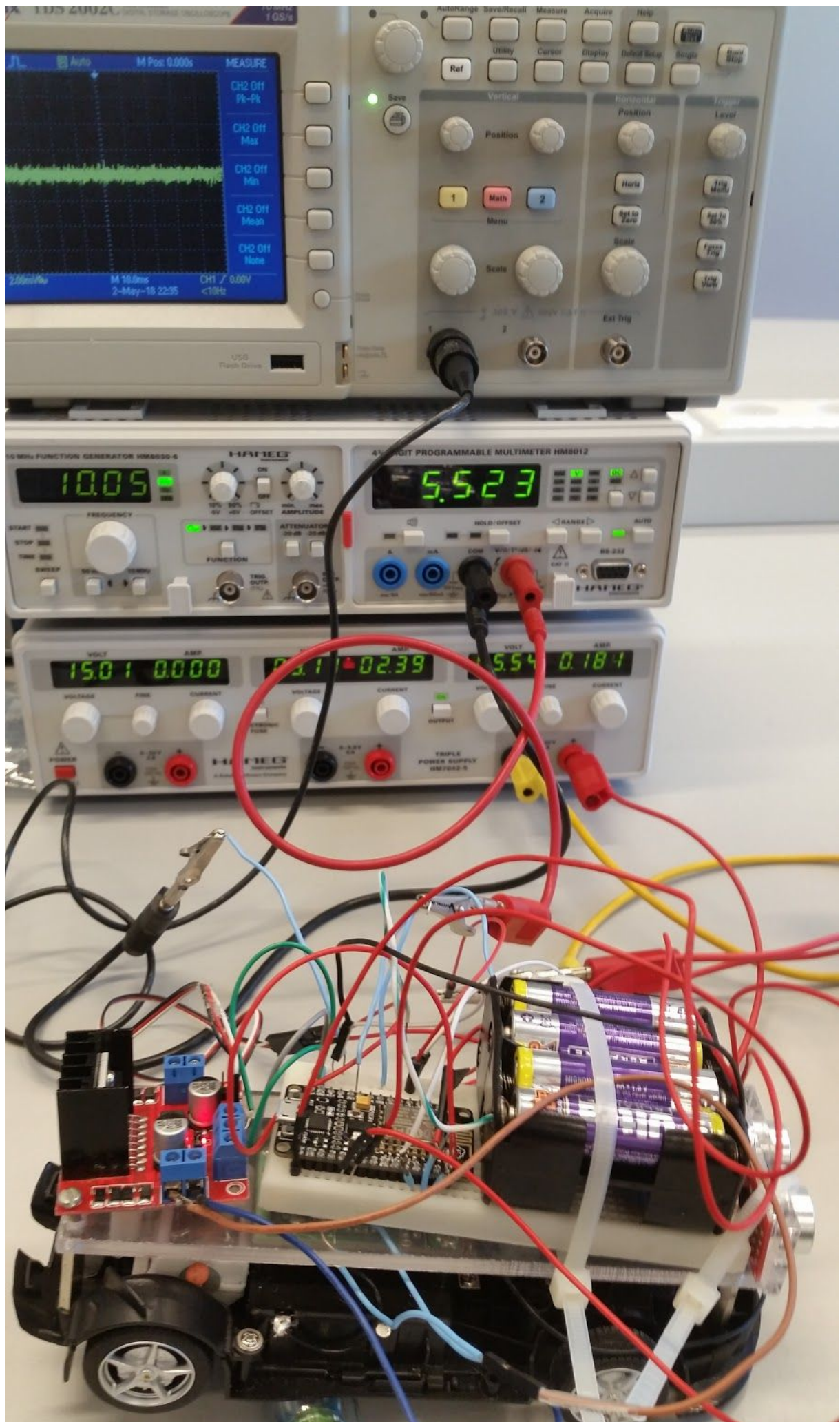
forced us to use 5.7V directly from the battery.



When the motor is running, the voltage of the battery drops to around 5,291V.



When the motor is not running but the circuit is powered then the voltage is around 5,523V.



UDP Latency

Distance measurement

Here we will test the reliability of the distance measurement component; HC-SR04. We will first look at the datasheet for some information. Here we find:

- Voltage: 5V
- Current: 15mA
- Range: 2 - 400cm
- Angle: 15°

The first problem arises with the voltage requirements of 5V as our NodeMCU can only deliver 3.3V.

This is supposedly very important but we will see in our results that it works fine just using 3.3V.

We will now test the accuracy of the module using the following test

Setup

two cars are positioned 89.1cm apart in a 21cm wide tunnel replicating the final setup. (See right)

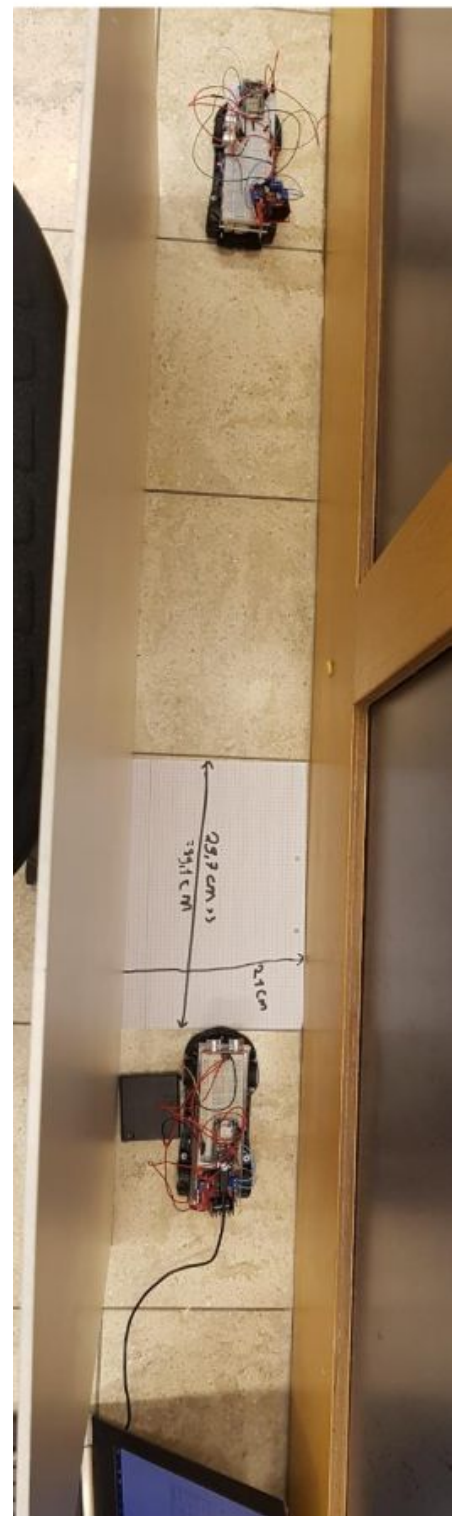
Both cars are stationary and the motors are turned off.

The bottom car is programmed to send its distance through a serial connection to the laptop, which displays these values on the serial monitor.

Code

```
long getDistance() {  
    digitalWrite(trigPin, HIGH);  
    delayMicroseconds(10);  
    digitalWrite(trigPin, LOW);  
    travelTime = pulseIn(echoPin, HIGH);  
    return (travelTime*0.034/2);  
}
```

As explained in a previous paragraph, the distance is equal to the traveltime (μs) * speed of sound ($0.034\text{m}/\mu\text{s}$) divided by two as the sound makes a round trip.



[illegible]

Car Velocity

Conclusion

References

Acknowledgements