Project Report

Inhoud

[Introduction 1](#_gjdgxs)

[Title 1](#_30j0zll)

[Objectives 1](#_1fob9te)

[Material and Methods 2](#_3znysh7)

[UDP 2](#_2et92p0)

[H-Bridge 2](#_tyjcwt)

[NodeMCU 2](#_3dy6vkm)

[Results 2](#_1t3h5sf)

[Conclusion 2](#_2s8eyo1)

[References 2](#_17dp8vu)

[Acknowledgements 2](#_3rdcrjn)

# Introduction

As young engineers, we wonder add lot about the real world and the problems that we face every day. In a discussion on a topic of accidents on the road, we realised the number of accidents on the roads is so high which made us thinking about a safer way that cars could travel on the roads. We have all seen the documentary where different car brands have been brought to testing its safety of automatic brake systems. Surprisingly, not many of them came out with a positive result. What if cars can communicate with each other or a server controls the cars to reduce traffic on the roads? These questions have brought us to the idea of our project: CarToCarCommunication.

# Title

# Objectives

## Car distance scanning

## Getting commands from access point

# 

# 

# Material and Methods

## UDP

//

## H-Bridge

//

## 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 9 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.

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 code

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.

Commands  
This section will explain the functionality of the commands we use in our code.

* pinMode(pin, mode) : Defines in or output of a given pin
  + pinMode(5, OUTPUT);
* Serial.begin(baudrate) : Defines the serial connection through the micro-USB or uart
  + Serial.begin(115200);
* Serial.print(text) : sends the text to the serial connection
  + Serial.print(“Testing serial connection”);
* analogWrite(pin, duty\_cycle) : writes a PWM with given duty cycle to a output-pin
  + analogWrite(12, 200);
* digitalWrite(pin, value) : writes a high or low to a output-pin
  + digitalWrite(2, HIGH);
* pulseIn(pin, value) : start sensing for a value and times how long value stays on pin.
  + pulseIn(4, HIGH);

WiFi

* WiFi.begin(ssid, password) : Sets the mcu in access point and connect to server.
  + WiFi.begin(“ubuntu”, “12345678”);
* WiFi.status() : will return WL\_CONNECTED when connected
  + if(WiFi.status() != WL\_CONNECTED);
* WiFi.localIP() : returns the IP of the nodeMCU when connected to WiFi

UDP

* Udp.begin(port) : Setup UDP library and start sensing on port
  + Udp.begin(2000);
* Udp.parsePacket() : checks if packet is waiting and return the length
  + int packetSize = Udp.parsePacket();
* Udp.remoteIP() : Returns IP of the connected server
  + IPAddress serverIP = Udp.remoteIP();
* Udp.remotePort() : Returns Port where last packet got in (client side)
* Udp.read(destination, size) : reads ‘size’ chars of packet and write it to dest. Returns length
  + int len = Udp.read(packetBuffer, 255);
* Udp.beginPacket(IP, Port) : Start making a new packet to dest IP and dest port
  + Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
* Udp.write(content) : write the content in the packet that u started
  + Udp.write(“ACK”);
* Udp.endPacket() : stop writing the packet and send it.

Setup()  
Here we want to configure our pins, setup a serial connection and start a Wi-Fi connection with the server. Once this is done we can enter our Loop.

## PWM

//

## Ubuntu server

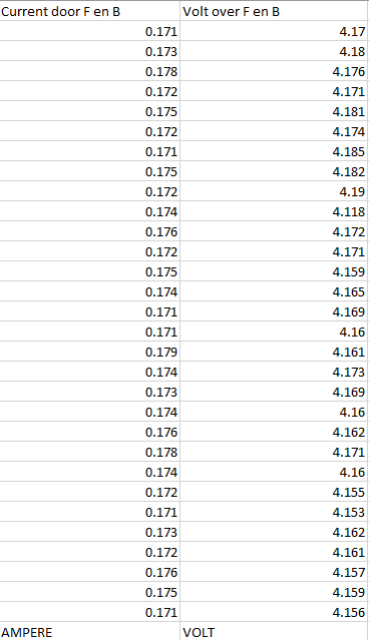
//

# 

# Results

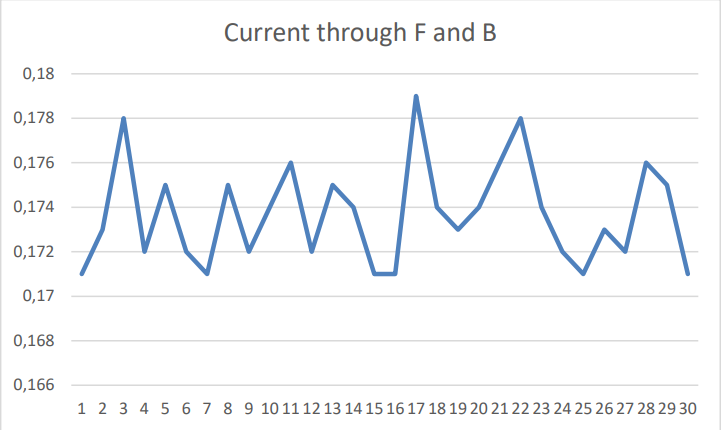
## Car Motor

For the car motor we made following measurements:

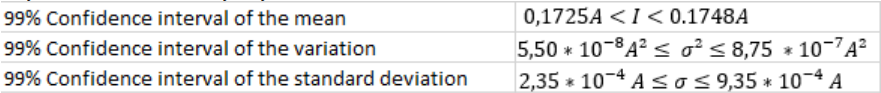


Lets discuss each part.

### Current

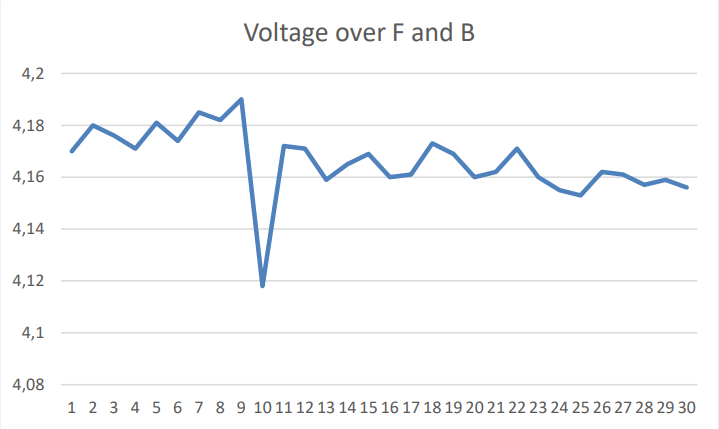




The standard deviation of this sample is 0.0023 ampere which means the values differ from each other approximately 2,3 mA. The variation also measures the spreading of the values, in this case the amperes. The value is small because the amperes are very close to each other, they only differ in a factor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

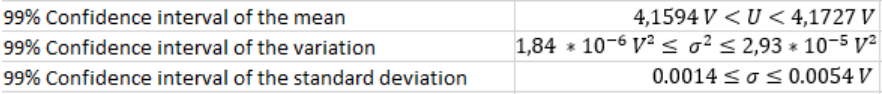
These Confidence intervals show that 99% of the values of our measurements have a mean, variation, standard deviation that is between these intervals.

### 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.



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

## UDP Latency

## Car Velocity

# Conclusion

# References

# Acknowledgements