Registration number 100134028

2018

Car engine diagnostics using a Raspberry Pi

Supervised by Dr. Ben Milner

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

Many parameters are to be taken into consideration when investigating the health of a car engine.These include the compression in each cylinder, battery voltage and current, cranking speed and many others. Usually in modern cars On-Board Diagnostics(OBD) are used where a portable specialised computer system is plugged to the car and can read fault codes.
This project will implement a non-intrusive alternative where the battery voltage before, during and after the ignition is measured and can provide information that can be used to determine the health of the engine. This will be implemented by using the combination of a Raspberry Pi and a Custard Pi.

Acknowledgements

I would like to thank my project supervisor, Dr Ben Milner for his help and support whenever I needed him throughout all the stages of the project.
I would also like to thank Peter Trollope for his technical assistance throughout the project.

Contents

List of Figures

1 Introduction

1.1 Motivation

Year by year automobiles become smarter and equipped with more sensors to observe health of the engine. But appart from the common error signs in the dashboard that maybe provide you with a warning about low levels of engine oil and if a door is open they can't tell you much more. Especially older model cars that weren't eqquiped with OBD it meant that the health of the car and finding what the problem is, relied solely to the skills of your mechanic and the amount of work they put in. High amount of working hours means higher fees and the problem could still not be detected. But even in newer cars that they are already equipped with OBD it means that you still rely solely to your mechanic as the OBD readers are expensive pieces of equipment that usually only mechanics own and know how to operate.

1.2 Project aims

The aim of this project is to create a cheap alternative to the OBD and investigate how the measurements taken from a car battery can help us derive conclusions about the engine health. Then use a suitable device to perform these measurements and create appropriate reports that demonstrate the findings.

1.3 Description and understanding of the problem

First of all, to understand how we are going to make the measurements and what those represent we need to understand how an engine works.

All modern auto-mobiles engines are of the in-cylinder combustion(internal-combustion engine, 2016) type. This means that the combustion that gives the motion on the wheels happens inside the engine cylinder. How this is done is by the piston completing four cycles (internal-combustion engine, 2017) , also known as four-stroke engines. The four cycles are the intake stroke, the compression stroke, the combustion or power stroke and the exhaust stroke. After the mixture of fuel and air is inserted in the cylinder it is compressed and while being compressed the ignition happens because of the spark plug on

the top of the cylinder. For the spark plug to ignite the fuel-air mixture it needs to get some energy. This is provided by the battery ignition system. (ignition, 2017)

A battery ignition system has a 12-volt battery charged by an engine-driven generator to supply electricity, an ignition coil to increase the voltage, a device to interrupt current from the coil, a distributor to direct current to the correct cylinder, and a spark plug projecting into each cylinder.

In this project we will measure the starter motor voltage on the ignition phase and by observing the output and the voltage drops we can see if all the cylinders compress at the same rate. In a healthy engine all the cylinders should be compressing at the same rate. This will be achieved by connecting our device on the battery and trying to start the engine.

1.4 Raspberry Pi and OBD

OBD is a very helpful technology which provides very insightful information. It can provide information for every part of the engine and any problems that may exist. It can provide information that even the most skilled mechanic can not know, like for example issues with the electronic controls that every car nowadays include in great amounts.

On the downside though, OBD was not always available and even when it became available some manufacturers did not straight away equip their cheaper models with it. Also since up to 20 years ago cheap manufacturers did not have many smart controls used in their vehicles there was not a big need for the technology. In addition, a lot of older cars are still in use by many and they can not have access to the leisure of this technology. Moreover the fact that they are expensive equipment to acquire and the annually subscriptions that need to be paid to manufacturers so you can have access to updates regarding the engine or electronics, it only makes them viable as an investment to mechanics that are going to use it daily.

The fact that they are expensive to buy makes them also expensive to use. When you think that your car may not perform as supposed the first reaction is to visit your mechanic for an inspection. Though, doing that means that your mechanic will charge you for just making the inspection and the use of the equipment even if it turns out that nothing is wrong.

That creates the need for a cheaper alternative that can provide basic assumptions for the engine health and this is what this project aims to create. To do that it has been decided that a Raspberry Pi and a Custard Pi will be utilised.

Raspberry Pi (RPi) is a credit card sized computer that runs on a Linux OS distribution called Raspbian. The reason that RPi was chosen for this project is that it also has the functionality of using General Purpose Input/Output (GPIO) pins that allows connecting sensors and getting input or providing output from them, and it can be used to read voltage from a car battery. The RPi though does not have analogue input and that is why the Custard Pi (CPi) is being used. CPi is a RPi add on board connected to the GPIO ports of RPi. The CPi takes the analogue input from the car battery and turns it into digital input that can be processed by the RPi. The RPi can be connected to any HDMI output and be used as a proper computer. The size of the RPi makes it extremely portable and the cost of it, total cost of about 50 pounds for both RPi and CPi, makes it a much cheaper alternative for the purpose of measuring voltage from a car battery.

1.5 Report Overview

This report is presented the following way. The first section describes the general situation and techniques used in determining the health of a car engine and an outline of the aims of the project and the structure of the report. The second section reports on research into how car engines work and what role the battery has on that and especially on the ignition part and the parameters that are affected and they are to relation to the aim of the project. Also some research is done on the hardware involved. The third section is about what is used to sample the data needed and the circuit and code involved in facilitating the sampling. The fourth section is about how the data collected are processed and displayed and the fifth section is about testing the system on cars. The last section concludes the whole report, by evaluating if the project aims were met.

2 Research

2.1 Car batteries

When talking about car batteries there is the common belief that a car battery is 12 volts(V). Actually a fully charged battery rests at 12.6 V. (Controls, 2018)

Below there is a table with the levels that a battery is at, at certain voltages.

Battery Voltage	State of charge
12,6	100%
12,4	80%
12,2	60%
11,9	40%
11,5	20%
10,5	0%

When the engine is running the alternator charges the battery to keep it fully charged. Also the regulator keeps the output between 13.6 to 14.6 V to protect the electrical components throughout the vehicle. The role of the voltage regulator is to control the field current applied to the spinning rotor inside the alternator. When there is no current applied to the field, there is no voltage produced from the alternator. When the voltage drops below 13.6 volts, the regulator will apply current to the field and the alternator will start charging. When the voltage exceeds 14.6 volts, the regulator will stop supplying voltage to the field and the alternator will stop charging.(Ofria, 2018)

2.2 Internal Combustion Engine

The idea of an Internal Combustion Engine was first introduced by French inventor Philipe Lebon around 1800 but he did not have the chance to build it as he died in an attack. After him many inventors and engineers tried to build a working engine but the first efficient engine to be built was by Belgian engineer Etienne Lenoir in 1860.

The engine built by Lenoir was a two-stroke engine with a battery for electrical ignition and was mainly used for powering small machines and it was powered by coal. The next step in internal combustion engines was made by German engineer Nikolaus

Otto when he patented the four-stroke engine in 1876. The engine was using half the fuel compared to the Lenoir engine and it was more powerful. Then his patent was invalidated in 1886 as his competitors found an older patent describing the four-stroke cycle. This resulted in many other inventors being able to use the basic idea and in 1886 German Karl Benz introduced a refined Lenoir engine using liquid fuel.

Then in 1892 German engineer Karl Diesel introduced the diesel engine which had a much higher compression than petrol engines. Because of the high compression there was not any need for electrical ignition as the fuel was ignited the moment it was injected in the cylinder. (internal-combustion engine, 2016) (internal-combustion engine, 2017)

A four-stroke engine means that each cylinder requires four strokes of its piston in order to complete a power stroke. (Heywood, 1988) (Windsor, 2011)

The four strokes are the following:

1. Intake stroke which starts with the piston at the top of the cylinder and it starts drawing a fresh mixture air-fuel by opening the inlet valve while moving to the bottom of the cylinder and closes the valve when reaching the bottom.

2. Compression stroke which moves the piston to the top thus compressing the mixture. When the piston almost reaches the top the combustion is initiated and the pressure of the cylinder rises.

3. Power(expansion) stroke where the piston is pushed to the bottom due to the pressure created in the compression stroke. Towards reaching the bottom the exhaust valve opens.

4. Exhaust stroke where the piston is pushing the gases out and when it reaches the top the exhaust valve is closing an the inlet valve opens and the cycle starts again.

In figure 1 we see a graphical representation of the four strokes and the position the piston is in, in each if the cycles.

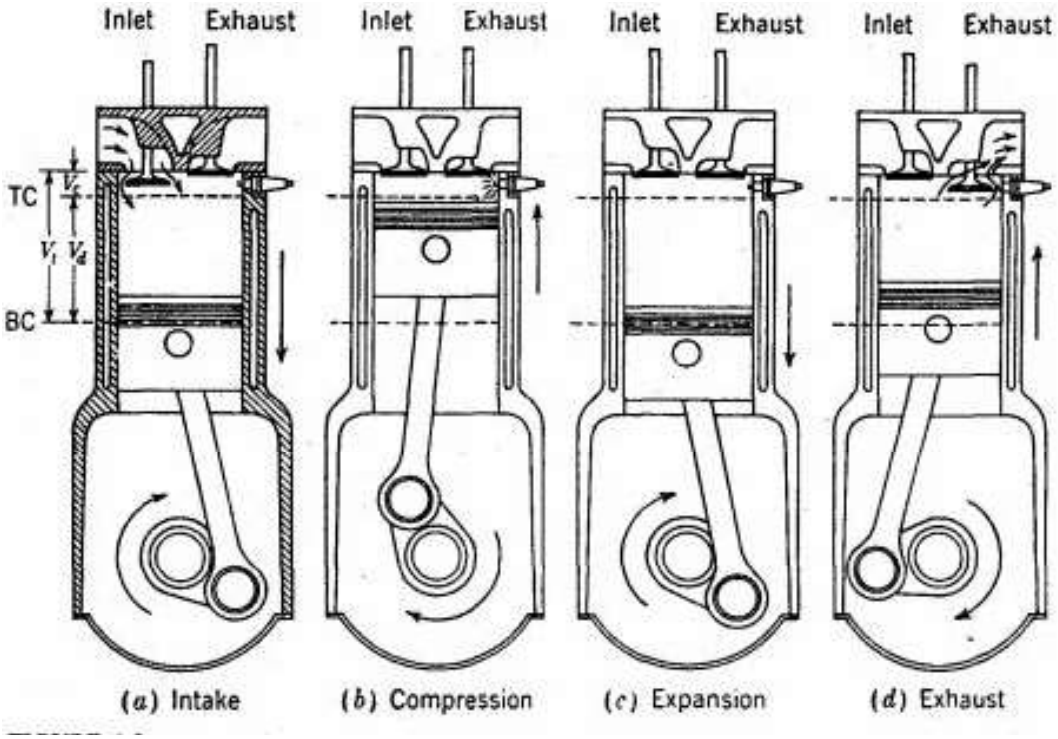

Figure 1: A representation of the four cycles and the position of the piston and status of valves at each point. (Heywood, 1988)

2.3 Ignition

A battery ignition system has a 12V battery that is being charged by an engine-driven generator, a coil to increase the voltage, a device to interrupt the current from the coil, a distributor to direct current to the cylinders and a spark going to each cylinder.

The device used to interrupt the current is a reluctor, a magnetic pulse distributor that timely produces electric signals.

The ignition coil is a winding of wire coiled around an iron core with a secondary winding over it attached to the distributor. When the breaker cam opens a moving finger inside the distributor touches contacts each of them being projected to a different cylinder. The projection to each each cylinder and spark plug, happens when the piston has almost reached the top of the compression stroke.

The spark plug consists of an electrode embedded in insulated ceramic and a ground

electrode which between them they have a small gap. Because of the high voltage projected the spark jumps the gap and ignites the air-fuel mixture. (ignition, 2017)

2.4 Starting system and Cranking Speed

For an engine to be able to start the combustion process an electric motor cranks and it rotates the flywheel which then starts moving the crankshaft which initiates piston movement. Thus the cylinders fire up and the engine keeps running. Sufficient power is needed in order to allow the flywheel keep rotating for the first couple of firing strokes until the engine develops sufficient power to run unassisted.

A petrol engine requires a minimum of 50-100 revolution per minute(rpm) where a diesel engine requires between 150-250 rpm.The starter is turning at about 10 to 15 times that. (Pearson, 2011) (Tutorials, 2014) (starter, 2005)

2.5 Existing similar applications

While I was searching for similar applications I came across VERUS from Snap-on. (Snap-on, 2015b) VERUS is a 10 inch tablet with a specialised software. The main functionality is the same as we are trying to meet in this project. You connect on the battery and crank the engine to inspect the voltage projected to the starting motor. It also some other functionality like plugging into the spark plugs to identify the cylinder sequence. This tool costs around 2500 pounds which makes it very non-affordable.

The difference with our project is that our project really simplifies the project as it only needs a couple of wires clipped to the battery where the Snap-on need specialised equipment and accessories to perform the same tests. Also our project only costs about 50 pounds which is a very big difference compared to the price of Snap-on.

In figure 2 we can see a picture of the VERUS Snap-on diagnostic system.

Figure 2: This is the VERUS diagnostic system. (Snap-on, 2015a)

2.6 Raspberry Pi and Custard Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. The Raspberry Pi GPIO allows external electronics to be connected and be controlled from the RPi. The RPi has two rows of 20 pins. (RPi, 2016)

Raspbian is the recommended operating system for normal use on a Raspberry Pi. Raspbian is a free operating system based on Debian, optimised for the Raspberry Pi hardware and it comes with over 35,000 packages. (Raspbian, 2016)

In figure 3 we can see a picture of the Raspberry Pi computer.

Figure 3: This is the Raspberry Pi computer. (RPi, 2016)

The Custard Pi is made by SF Innovations and has a 26 pin connector which is attached to the RPi and provides analogue and digital inputs and outputs. It uses the 3.3V supply from the RPi to power the board. The CPi has 2 digital inputs, 4 digital outputs, 0, 3.3 and 5V power connectors, 2 analogue inputs and 2 analogue outputs. The CPi takes a maximum of 3.3V as analogue input. (Innovations, 2017b)

In figure 4 we can see a picture of the Custard Pi board.

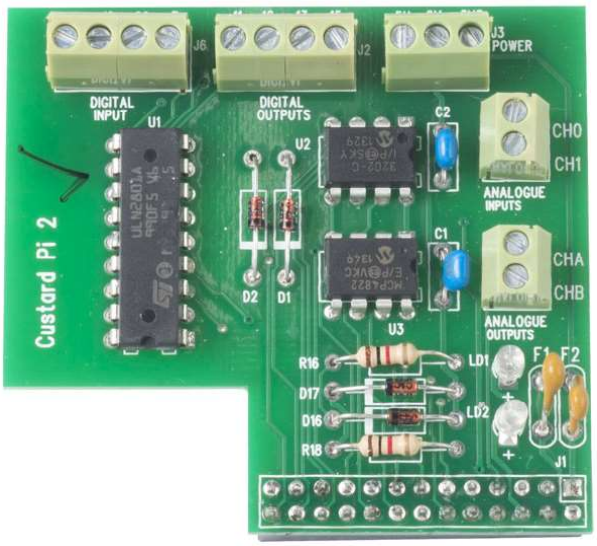

Figure 4: This is the Custard Pi board. (Innovations, 2017b)

2.7 Research Summary

The research on the parts involved to the project gave me a better understanding of how car engines work and how the ignition system works. This made me understand how the car battery which is going to be the main measurement for our project is affected at the ignition stage. Also the cranking speed is an important parameter that affects the project outcome.

Also the understanding of the Raspberry and Custard Pi makes it easier to be used and help us achieve our project aims.

3 Sampling

3.1 Voltage divider

Since a car battery outputs between 12 and 14.6 volts throughout the whole operation of the car we need to bring the voltage down to a maximum of 3.3V which is the maximum that a Custard Pi takes as an input.

In order to do that we are going to use a Voltage divider or else known as Potential divider.

A Voltage divider is a simple circuit of resistors that decreases the voltage from an input. The equation for this circuit is Vout = $\frac{R2}{R2+R1}$ * Vin where Vout is the output voltage we desire, Vin is the input voltage from the battery and R2 and R1 are the resistors used.

In figure 5 we can see the schematic of the voltage divider circuit.

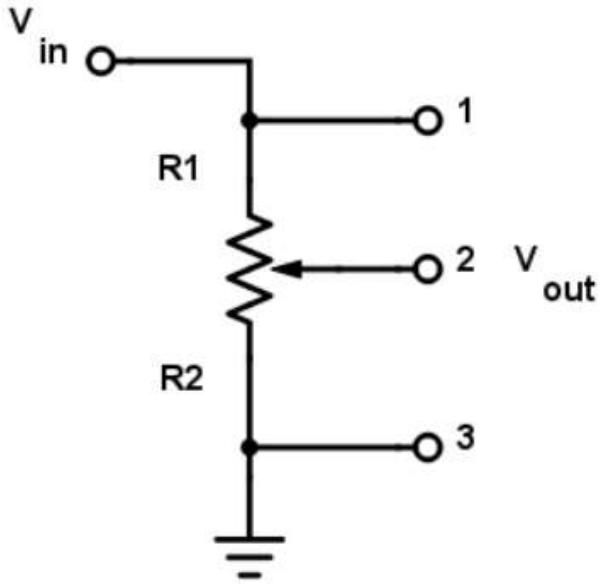

Figure 5: A schematic of the voltage divider circuit. (about circuits, 2016)

It is known that the desired output voltage is 3,3V and the input should be between the range of 13,7 to 14,7. (Controls, 2018) Although sometimes batteries may face issues and generate more than the desired and supposed voltage. For this reason, it was decided that the circuit would be built on the assumption that the maximum input is 20V. This way we are protecting our hardware from possible damage in case there is malfunction with the battery. In order to figure out the values of the resistors we are going to use, we solve the equation with respect to the resistors and plug in the the known values. This gives us that $\frac{R2}{R2+R1} = \frac{Vout}{Vin}$ so the quotient of the resistor values should be $\frac{3,3}{20}$. The closest resistor values that are available in the lab are 3,3 kω and 18kω which is going to give us a quotient of $\frac{3,3}{21,3}$ which is very close to the original desired value.

In figure 6 we can see the voltage divider circuit built on a breadboard and a stripboard.

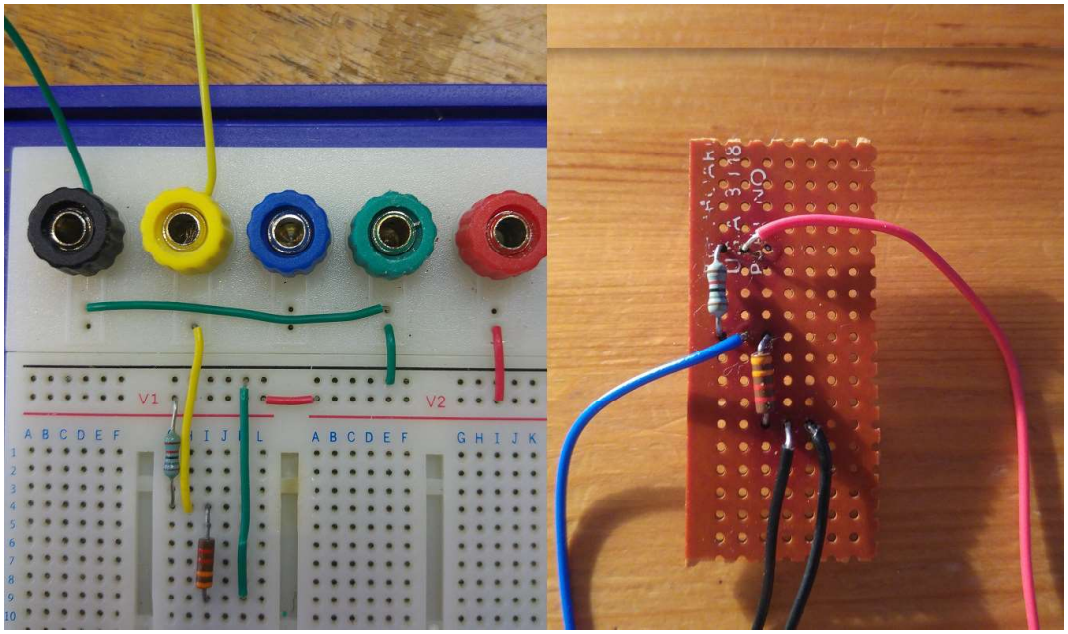

Figure 6: The voltage divider circuit on breadboard(left) and stripboard(right).

On the breadboard circuit the red connector is connected to the positive pole of the battery and the green connector to the negative pole. The yellow wire is connected to the input of the Custard Pi and the green wire to the 0V on the Custard Pi.

On the stripboard the red wire is connected to the positive pole of the battery and the blue wire to the input of the Custard Pi. Out of the two black wires, the one is connected to the negative pole of the battery and the other one to the 0V of the Custard Pi.

3.2 Analogue to Digital Conversion

Since the car battery outputs an analogue signal that is read by the Custard Pi we need to translate that to a digital value to be processed by the Raspberry Pi. For the purpose of doing that we are going to use the Analogue to Digital Converter (ADC) provided by the manufacturers of the Custard Pi. (Innovations, 2017a)

```
#1/usr/bin/env python
#program to read analogue voltage on Custard Pi 2
import RPi.GPIO as GPIO
import time
```

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(26, GPIO.OUT) #pin26 is chip select 1
GPIO.setup(23, GPIO.OUT) #pin23 is clock
GPIO.setup(19, GPIO.OUT) #pin19 is data out
GPIO.setup(24, GPIO.OUT) #pin24 is chip select 0
GPIO.setup(21, GPIO.IN) #pin21 is data in
#set pins to default state
GPIO.output(24, True)
GPIO.output(26, True)
GPIO.output(23, False)
GPIO.output(19, True)
#set up data for ADC chip
word5= [1, 1, 0, 1, 1]
word6= [1, 1, 1, 1, 1]

def readchannel0():
#reads analogue voltge from channel 0
GPIO.output(24, False) #select channel 0
anip=0 #initialise variable
#set up channel 0

for x in range (0,5):
    GPIO.output(19, word5[x])
    #time.sleep(0.01)
    GPIO.output(23, True)
    #time.sleep(0.01)
    GPIO.output(23, False)

#read analogue voltage
for x in range (0,12):
    GPIO.output(23,True)
    #time.sleep(0.01)
    bit=GPIO.input(21)
    #time.sleep(0.01)
    GPIO.output(23,False)
    value=bit*2**(12-x-1)
    anip=anip+value
```

```
    GPIO.output(24, True)
    volt = anip*3.3/4096
    return volt
```

The program sets pins 19, 23 and 24 as outputs and pin 21 as input. Pin 26 is chip select. The program clocks in 12 bits at pin 21 and works out the decimal value based on position and value of the bit. Then it multiplies by 3.3 which is the maximum of voltage to be input in the Custard Pi and divides by 4096 which is the possible values for 12 bits.

3.3 Signal generator

In order to test the analogue to digital conversion and the voltage divider we will use a signal generator that outputs analogue signal in different frequencies so we can see how well can our system capture data in different speeds. Below there are a number of graphs showing how well the Custard Pi performs at each frequency. We plugged values of around a maximum of 12V as we were plugging a wave that was completely offset above 0. The values shown in the graphs are at a maximum of 1,8 to 1,9V as this is the value 12V are brought down by the voltage divider.

3.3.1 1Hz

In figure 7 we can see the the representation of the 1Hz wave.

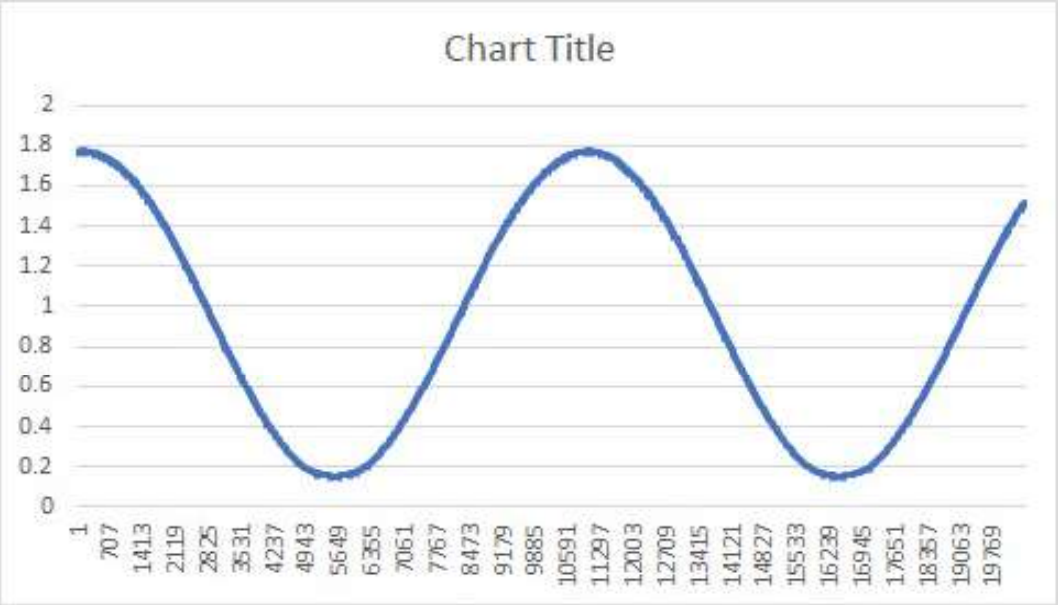

Figure 7: The graph of 1Hz wave

In 1Hz waves the amount of values between peak to trough and peak to peak are about 5000 and 10000 respectively. The wave is very nicely captured.

3.3.2 10Hz

In figure 8 we can see the the representation of the 10Hz wave.

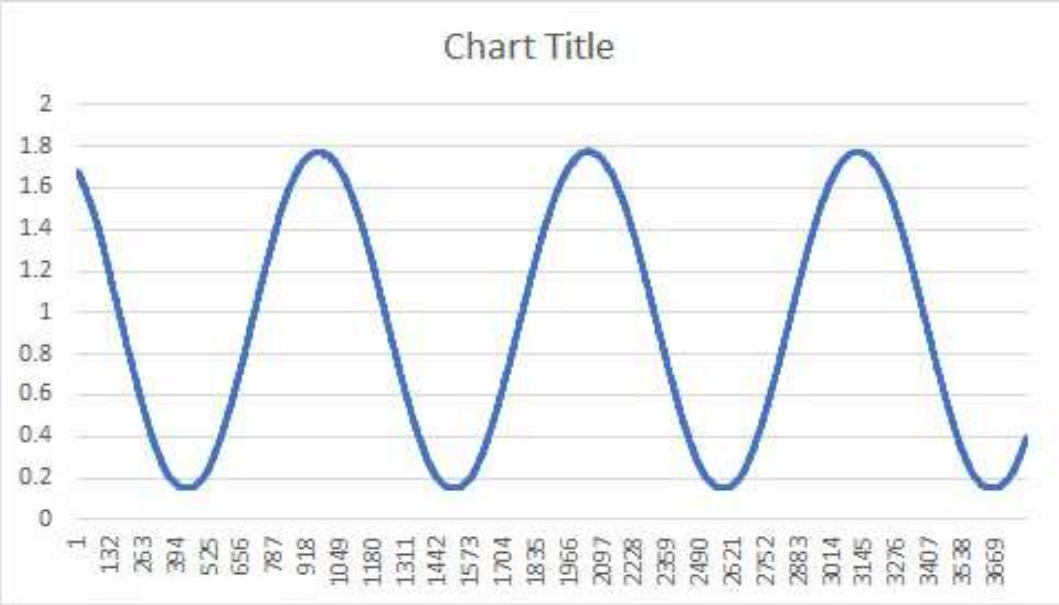

Figure 8: The graph of 10Hz wave

In 10Hz waves the amount of values between peak to trough and peak to peak are about 500 and 1000 respectively. The wave is very nicely captured.

3.3.3 100Hz

In figure 9 we can see the the representation of the 100Hz wave.

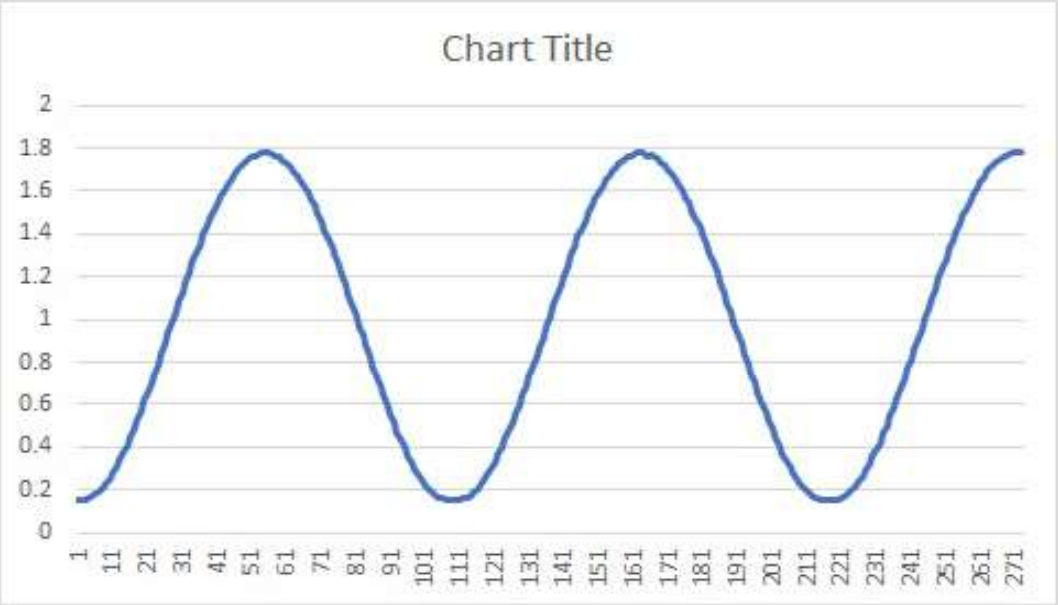

Figure 9: The graph of 100Hz wave

In 100Hz waves the amount of values between peak to trough and peak to peak are about 50 and 100 respectively. The wave is very nicely captured.

3.3.4 1kHz

In figure 10 we can see the the representation of the 1kHz wave.

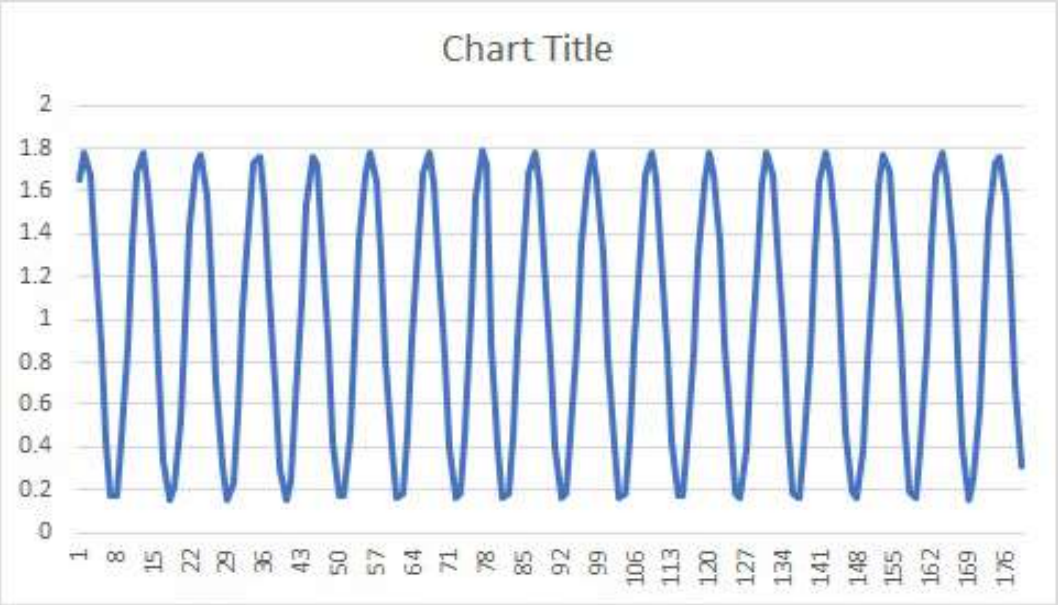

Figure 10: The graph of 1kHz wave

In 1kHz waves the amount of values between peak to trough and peak to peak are about 5 and 10 respectively. The wave is captured quite nicely but it is not very smooth at the edges as it is a quite fast wave.

All the measurements taken and the inspection of the data gave us a very good understanding of the correlation between the frequency of the wave and after how many values we can expect a peak or a trough in each situation.

This understanding will help later on when trying to plot the data to give a parameter to the function for how soon to capture peaks and troughs.

In the figure below we can see the signal generator we use for testing the system.

Figure 11: The signal generator used for testing the system

4 Signal processing

4.1 Peaks and Troughs

In order to decide if our system is working as it is supposed to, we need to capture the peaks and troughs of the signal. By capturing the peaks and troughs of the wave we can see the voltage drop at each cylinder at ignition stage which is correlated to the pressure in each cylinder.

4.1.1 PeakFind method

After researching for a while for libraries to use to detect peaks and troughs and trying to write my own I ended up using a library from Marcos Duarte. (Duarte, 2017)

```
        # %load ./../functions/detect_peaks.py
        """Detect peaks in data based on their amplitude and other
                                    features."""

        from __future__ import division, print_function
```

```
import numpy as np

__author__ = "Marcos Duarte, https://github.com/demotu/BMC"
__version__ = "1.0.4"
__license__ = "MIT"


def detect_peaks(x, mph=None, mpd=1, threshold=0, edge='
                                rising',
kpsh=False, valley=False, show=False, ax=None):

"""Detect peaks in data based on their amplitude and other
                                features."""


x = np.atleast_1d(x).astype('float64')
if x.size < 3:
    return np.array([], dtype=int)
if valley:
    x = -x
# find indices of all peaks
dx = x[1:] - x[:-1]
# handle NaN's
indnan = np.where(np.isnan(x))[0]
if indnan.size:
    x[indnan] = np.inf
    dx[np.where(np.isnan(dx))[0]] = np.inf
ine, ire, ife = np.array([[], [], []], dtype=int)
if not edge:
    ine = np.where((np.hstack((dx, 0)) < 0) &
                                (np.hstack((0, dx)) > 0))[0]
else:
    if edge.lower() in ['rising', 'both']:
        ire = np.where((np.hstack((dx, 0)) <= 0) &
                                (np.hstack((0, dx)) > 0))[0]
    if edge.lower() in ['falling', 'both']:
        ife = np.where((np.hstack((dx, 0)) < 0) &
                               (np.hstack((0, dx)) >= 0))[0]
```

```
    ind = np.unique(np.hstack((ine, ire, ife)))
    # handle NaN's
    if ind.size and indnan.size:
        # NaN's and values close to NaN's cannot be peaks
        ind = ind[np.in1d(ind, np.unique(np.hstack((indnan,
                              indnan-1, indnan+1))), invert=True)]
    # first and last values of x cannot be peaks
    if ind.size and ind[0] == 0:
        ind = ind[1:]
    if ind.size and ind[-1] == x.size-1:
        ind = ind[:-1]
    # remove peaks < minimum peak height
    if ind.size and mph is not None:
        ind = ind[x[ind] >= mph]
    # remove peaks - neighbors < threshold
    if ind.size and threshold > 0:
        dx = np.min(np.vstack([x[ind]-x[ind-1], x[ind]-x[ind+1]])
                                , axis=0)
        ind = np.delete(ind, np.where(dx < threshold)[0])
    # detect small peaks closer than minimum peak distance
    if ind.size and mpd > 1:
        ind = ind[np.argsort(x[ind])][::-1]  # sort ind by peak
                                height
        idel = np.zeros(ind.size, dtype=bool)
        for i in range(ind.size):
            if not idel[i]:
            # keep peaks with the same height if kpsh is True
            idel = idel | (ind >= ind[i] - mpd) & (ind <= ind[i]
                                + mpd) \
                    & (x[ind[i]] > x[ind] if kpsh else True)
            idel[i] = 0  # Keep current peak
    # remove the small peaks and sort back the indices by their
                                occurrence
    ind = np.sort(ind[~idel])

    # if show:
    #     if indnan.size:
    #         x[indnan] = np.nan
```

```
        #     if valley:
        #         x = -x
        #     _plot(x, mph, mpd, threshold, edge, valley, ax, ind)
        return ind
```

The function accepts a number of parameters that define the behaviour of the function in detecting the signal's peaks. These are the following:

1. x: is a list containing the values of the signal
2. mph: this value limits the function to detecting peaks above this minimum peak height. The default value is None.
3. mpd: this value limits the function to detecting peaks that are least separated by minimum peak distance in number of data in the value list. The default value is 1.
4. threshold: this value limits the function to detecting peaks/valleys that are greater/smaller than this value in relation to their immediate neighbors. The default value is 0.
5. edge: This parameter is to detect rising/falling or both edges on a flat peak. The default value is None.
6. kpsh: This parameter is to keep peaks with the same height even if they are closer than mpd. The default value is False.
7. valley: If this value is set to true it detects troughs instead of peaks. The default value is False.
8. show: If this value is set to true it plots the wave with the peaks or troughs as soon as the detection sequence finishes. The default value is False.

The function returns ind, an array like which contains the indices of the peaks/troughs in the list of values.

When trying to detect the peaks and troughs the only parameters we take in consideration is the minimum peak distance (mpd) and the values of valley(true/false) as the rest of the parameters do not affect the ability of the function to detect the correct peaks.

In order to pass the correct value for mpd in each use case we observed the waves produced by the signal generator. After taking a lot of samples we saw that in 1Hz wave it is 10000 values from peak to peak, 1000 for 10Hz, 100 for 100Hz and 10 for 1kHz. So we can safely come to the conclusion that the amount of values between each peak decreases exponentially as the frequency of the wave increases.

So since a petrol engine cranks at about 60 rpm on average and a diesel at about 1500 rpm we assume that the starting motor turns at 600 rpm and 900 rpm respectively. Those values divided by 60 gives us that a petrol engine starter motor turns at 10 rotations per second (rps) and a diesel one at 15 rps and these are the frequencies at 10Hz and 15Hz. Given that we set the mpd at 1000 for petrol engines and at 1500 for diesel engines.

When detecting the peaks and troughs we first detect the peaks and return them in an array and then do the same for the troughs. Then we concatenate the two arrays and pass them to the plot function.

4.1.2 Plotting

The plotting function takes the list of values and array ind returned from the peak detection function and plots the wave and a red + at every peak or trough using the indices in ind.

```
        def _plot(x, ind):
          try:
              import matplotlib.pyplot as plt
              import numpy as np
          except ImportError:
              print('matplotlib is not available.')
          else:
              _, ax = plt.subplots(1, 1, figsize=(8, 4))

              ax.plot(x, 'b', lw=1)
              if ind.size:
                  ax.plot(ind, x[ind], '+', mec='r', mew=2, ms=8,
                     label='%d peaks and troughs' % (ind.size))
                  ax.legend(loc='best', framealpha=.5, numpoints=1)
              ax.set_xlim(-.02*x.size, x.size*1.02-1)
```

```
ymin, ymax = x[np.isfinite(x)].min(), x[np.isfinite(x
                    )].max()
yrange = ymax - ymin if ymax > ymin else 1
ax.set_ylim(ymin - 0.1*yrange, ymax + 0.1*yrange)
ax.set_xlabel('Data over time', fontsize=14)
   ax.set_ylabel('Amplitude', fontsize=14)
ax.set_title("Peak and trough detection")
plt.savefig('Amplitude wave with peaks and troughs
                    marked')
plt.show()
```

After testing it with various signals from the signal generator we observed that the peak finding method and the plotting function work very well. But as the signals generated from the signal generator do not have any noise this was expected to be easy.

Since though when the system is to be tested on a car the wave is going to be noisy we used Matlab to add noise to our wave and when plugged again to the peak finding function we could see that the function still works well and detects the correct peak and troughs.

Below the 2 figures show a 10Hz wave without noise and with added noise and as it can be seen the function does not get any issues and detects the correct peaks and troughs.

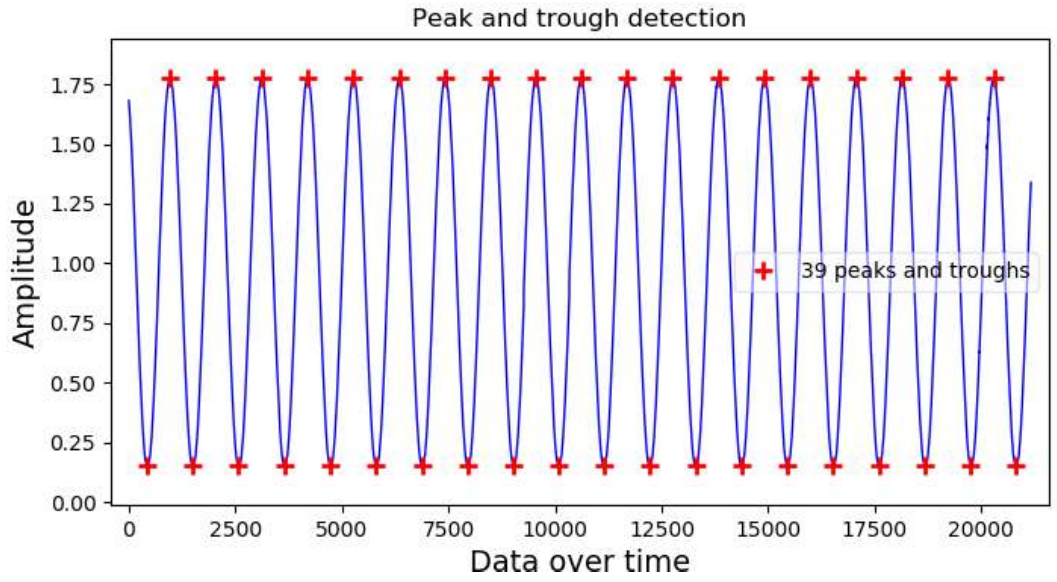

Figure 12: This is a 10Hz wave without any noise with peaks and troughs detected.

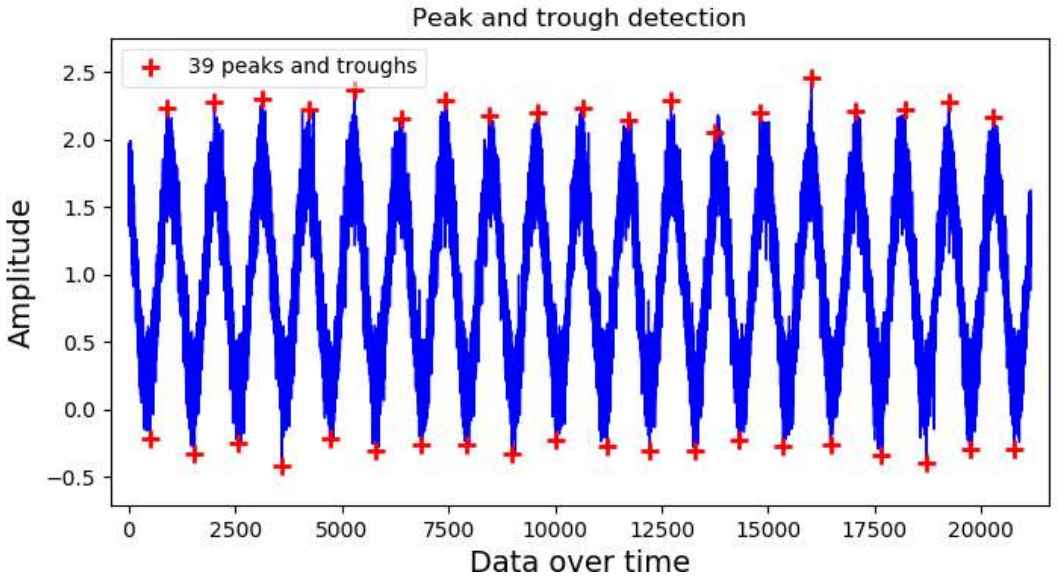

Figure 13: This is a 10Hz noisy wave with peaks and troughs detected.

4.2 Average of each cylinder

After detecting the peaks and troughs from the values list, they are then grouped into the amount of cylinders and the averages are calculated. What this means is that if for example a car has four cylinders, all the peaks are grouped in 4 data buckets and the same process happens for the troughs. Then the average of the peaks and average of the troughs for each cylinder are calculated and then plotted to show what is the average drop of each cylinder giving a graph like the following.

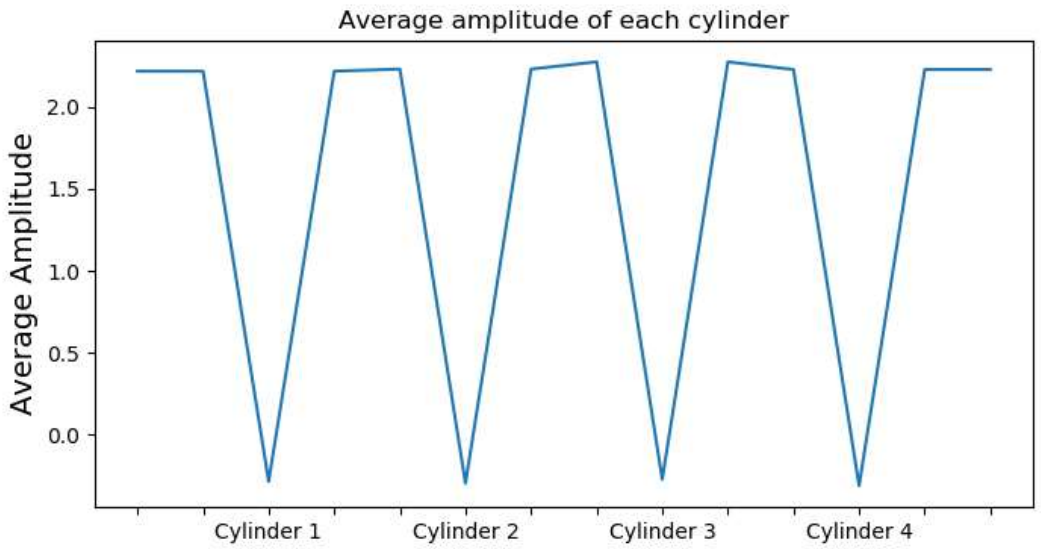

Figure 14: Graph representing the average drop in each cylinder.

The results plotted in figure 14 are the same data from figure 13. So we can see that the average peak for each cylinder is above 2V and the average troughs are below 0V which are the same results shown in the previous figure, subsequently meaning that our function captures the average values correctly.

4.3 Average of all stages

After calculating the average values for each cylinder another figure is plotted to show the average of the battery voltage before cranking the engine, the average of each cylin-

der at the cranking stage and the average of the battery after the engine has ignited.

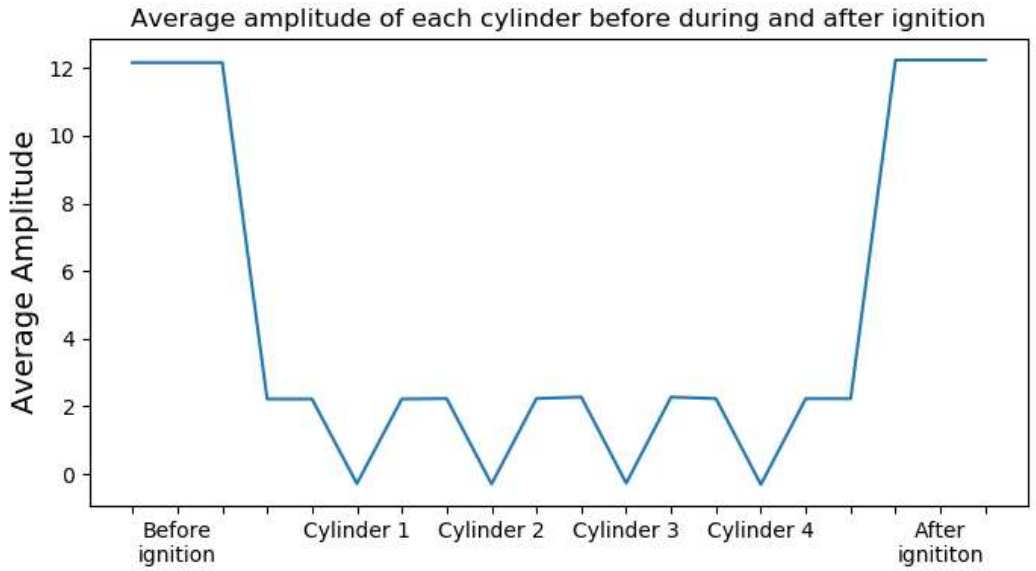

Figure 15: Graph representing the average drop in each cylinder and the average of the battery before and after the cranking stage.

The data used for this graph are two DC voltages of 12V and the data from the previous figures so to just demonstrate how the graph is going to look like on an actual measurement.

4.4 Percentage of performance in each cylinder

Another result that we are presenting is the performance percentage of each cylinder. What this means is that we take the peak average of each cylinder and we compare it to the maximum peak value in all the data set and see how well each cylinder performs.

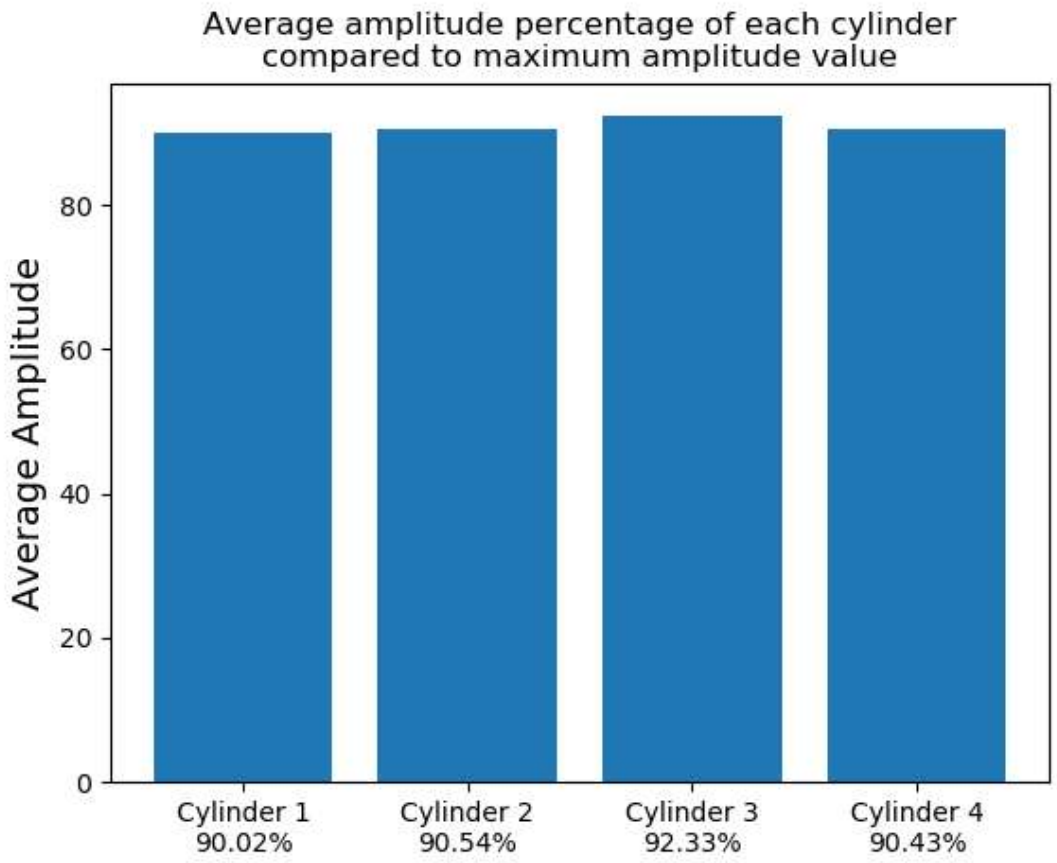

Figure 16: Graph representing the percentage of each cylinder compared to the maximum value in the data set.

The data used in this figure are the same as used before in finding the averages.

5 Testing

To test the system we used two different kind of cars. The first one was a 2008 Vauxhall Vivaro and the second one was a 1993 Mercedes 190E. The system was tested by connecting the device on the battery. Below there is a figure of the setup.

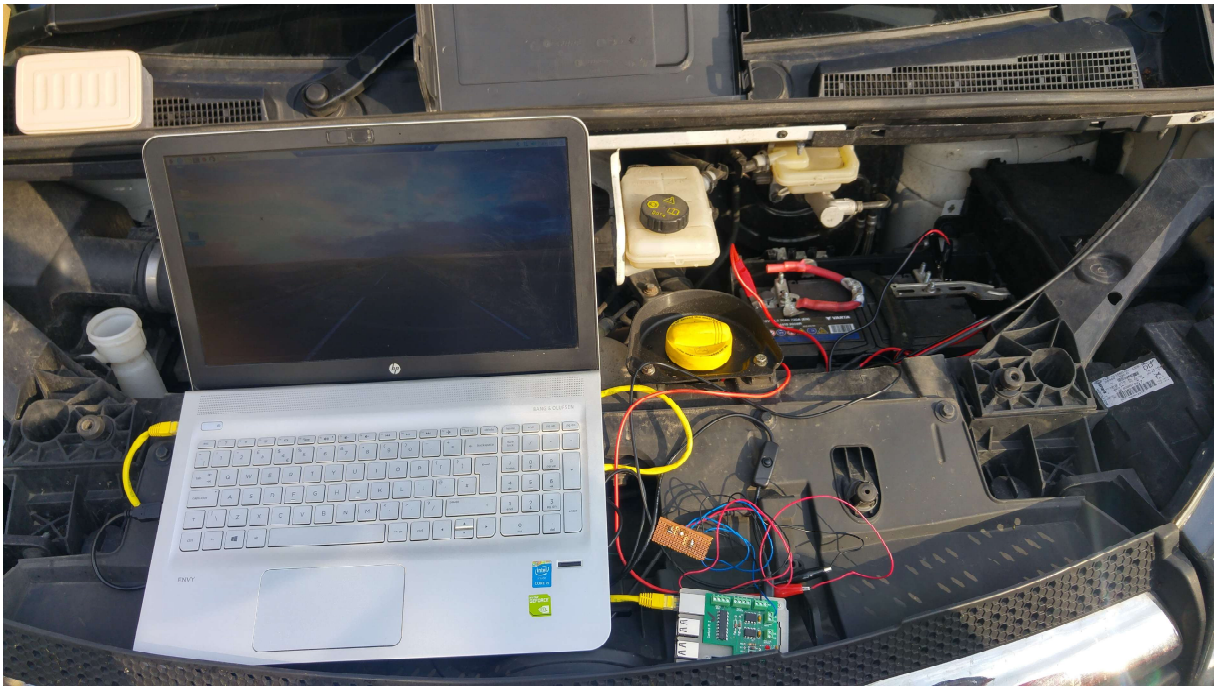

Figure 17: This is the setup to connect the device on a car for testing.

As it can be seen from the figure, the two poles of the battery are connected to the red and black wires on the stripboard which we implemented the voltage divider on. Then the blue and second black wire are connected to the input and the 0V of the Custard Pi respectively. The Custard Pi is mounted on the Raspberry Pi. The Raspberry pi is powered from the laptop with a USB cable and the Ethernet cable is used to tether the Raspberry Pi to the laptop in order to provide display and to be able to initiate the program.

5.1 Car 1

The first car is diesel powered so the mpd for detecting the peaks and troughs was set to 1500.

5.1.1 Sampling

After taking several samples we realised that the car starts very fast so we modified the system to sample for less time as if we let it sample for longer the results would not be representative about the engine's health.

5.1.2 Display

Below are the graphs generated after testing on the car.

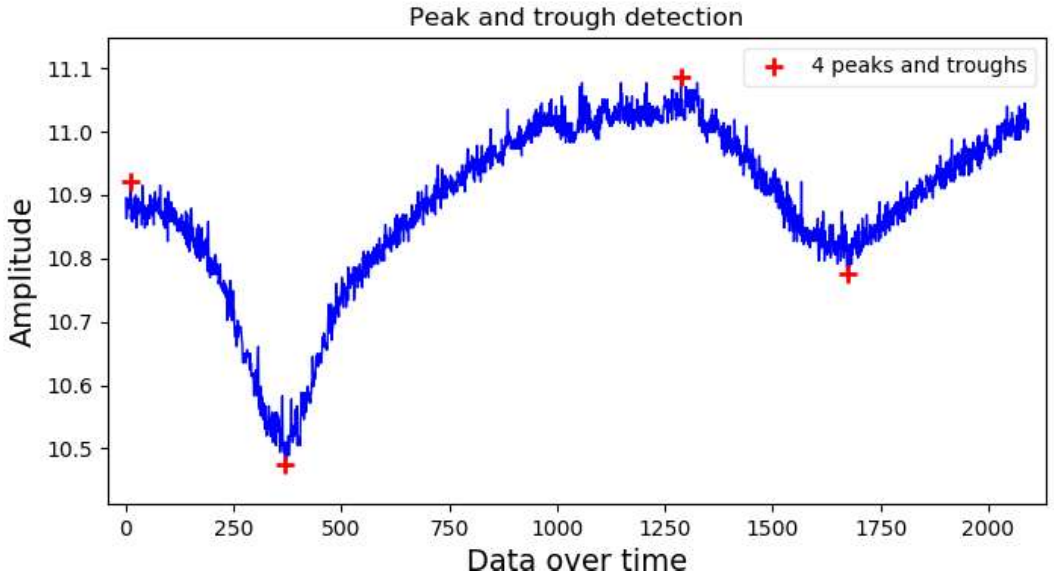

Figure 18: This is the signal during ignition for the first car with peaks and troughs detected.

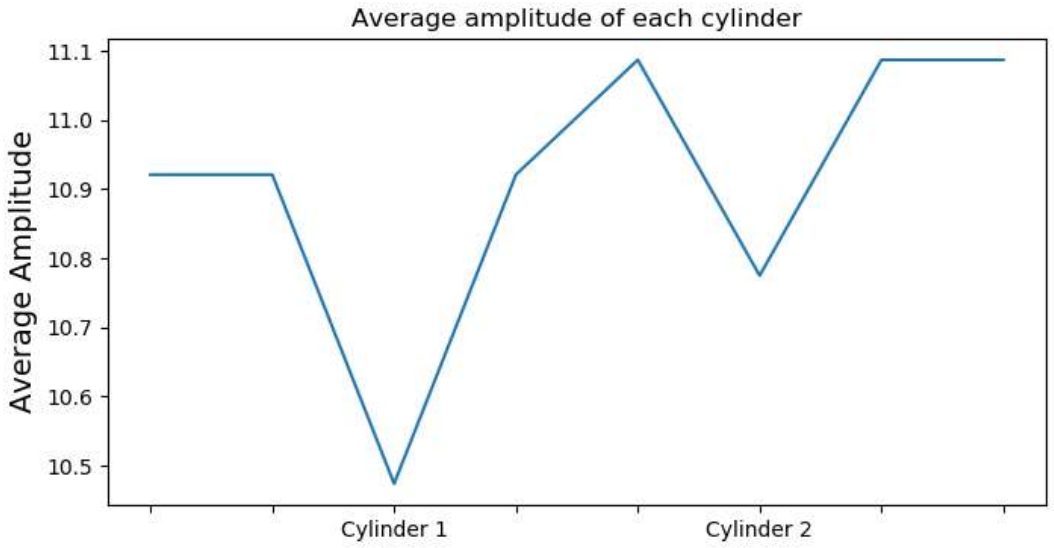

Figure 19: Graph representing the average drop in each cylinder.

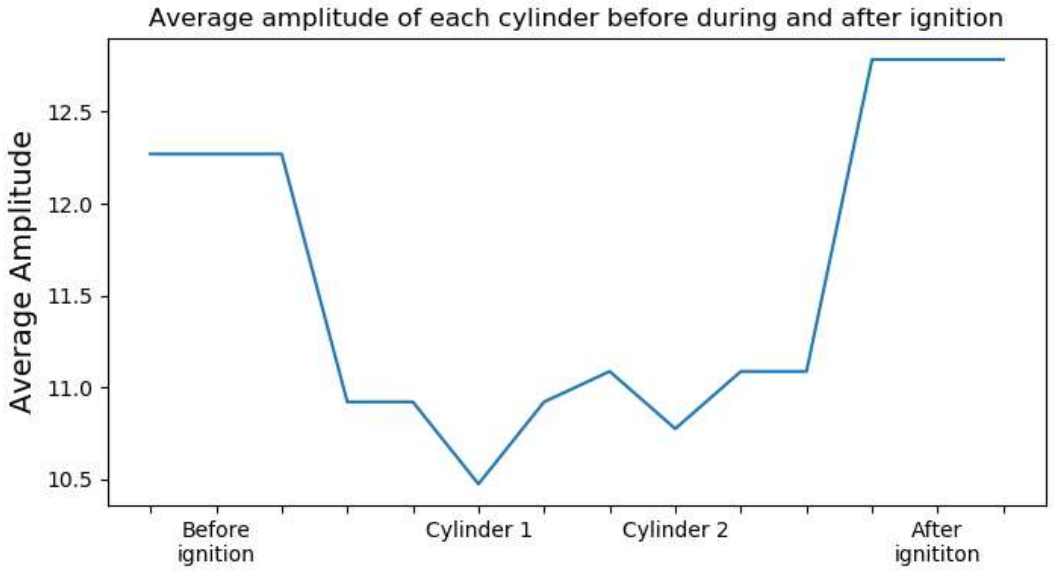

Figure 20: Graph representing the average drop in each cylinder and the average of the battery before and after the cranking stage.

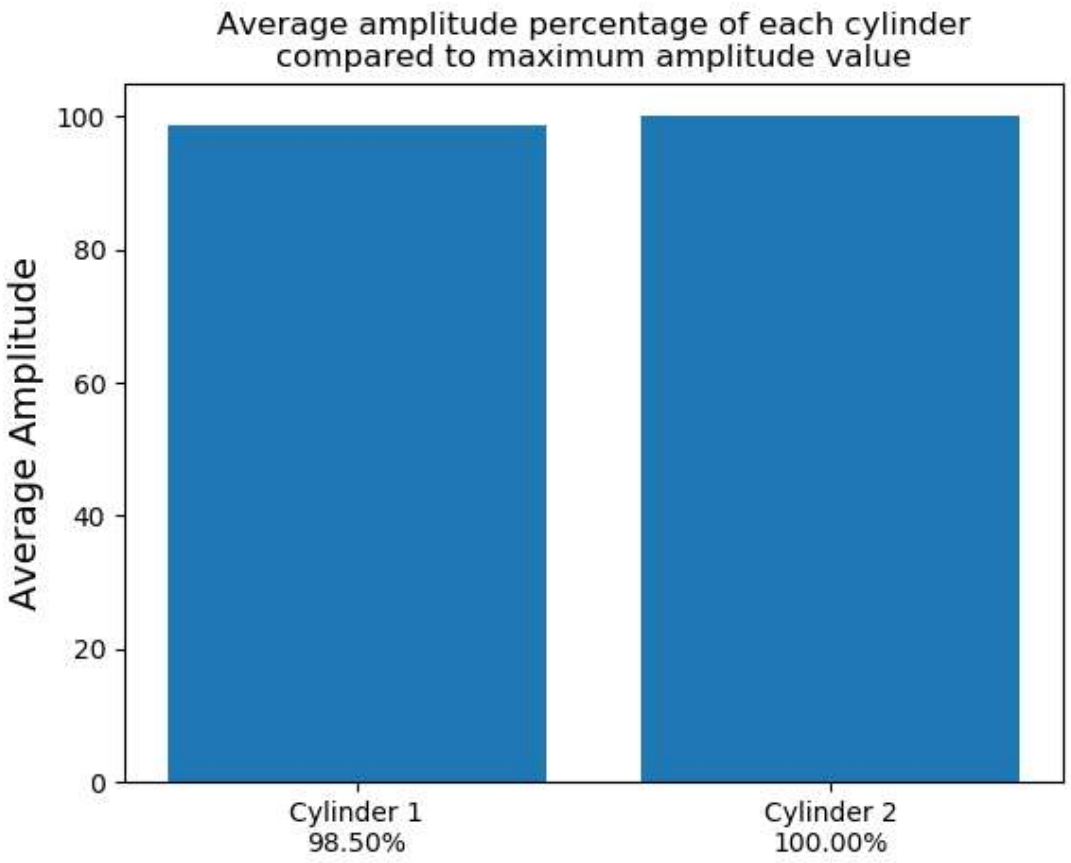

Figure 21: Graph representing the percentage of each cylinder compared to the maximum value in the data set.

5.1.3 Evaluation

The results from the first car were not very representative as the engine cranks very fast and we can not capture as least one compression for each cylinder in order to be able to see the difference between them. The voltage drop was not very high so we could assume that since it does not lose too much compression that the engine is in good state. The before and after ignition measurements though show that the battery is at good health and that the regulator keeps the voltage levels as desired in order to keep the car's electrical systems working without any problems.

5.2 Car 2

The second car tested is petrol powered so the mpd used for detecting peaks ant troughs was 1000.

5.2.1 Sampling

After taking a few samples we realised that as the first car it cranks very fast so in order to take some longer and better results we took out the fuel so the car does not start and give us the time to capture more data.

5.2.2 Display

Below the first 4 figures were before taking out the fuel and the next were after taking it out.

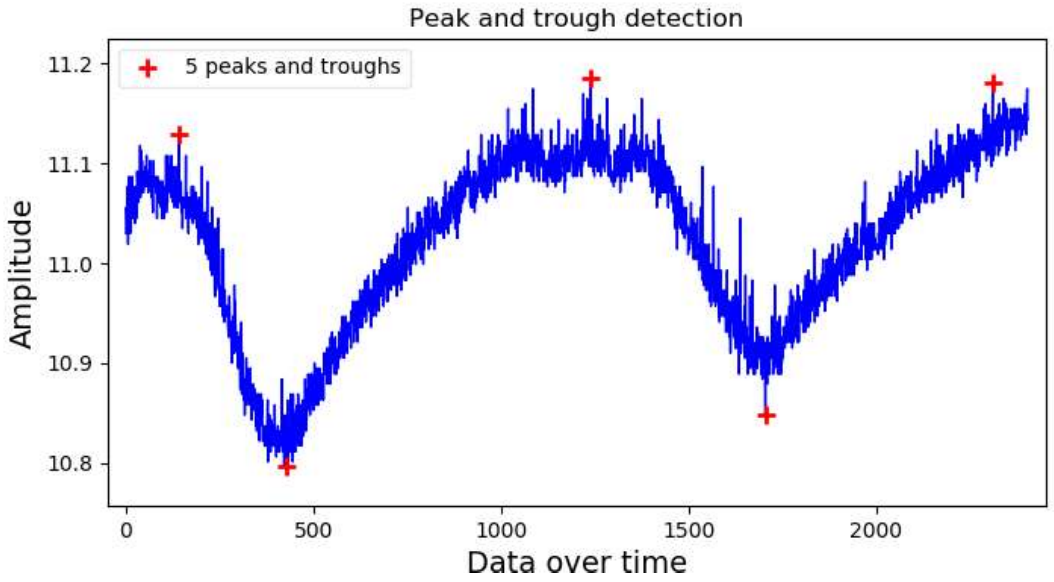

Figure 22: This is the signal during ignition for the first car with peaks and troughs detected before taking out the fuel.

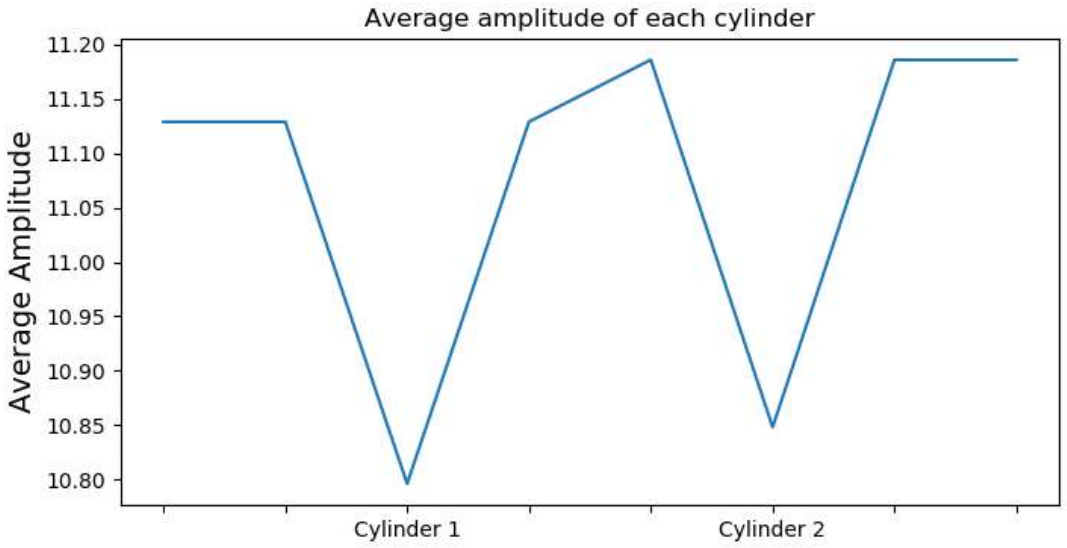

Figure 23: Graph representing the average drop in each cylinder before taking out the fuel.

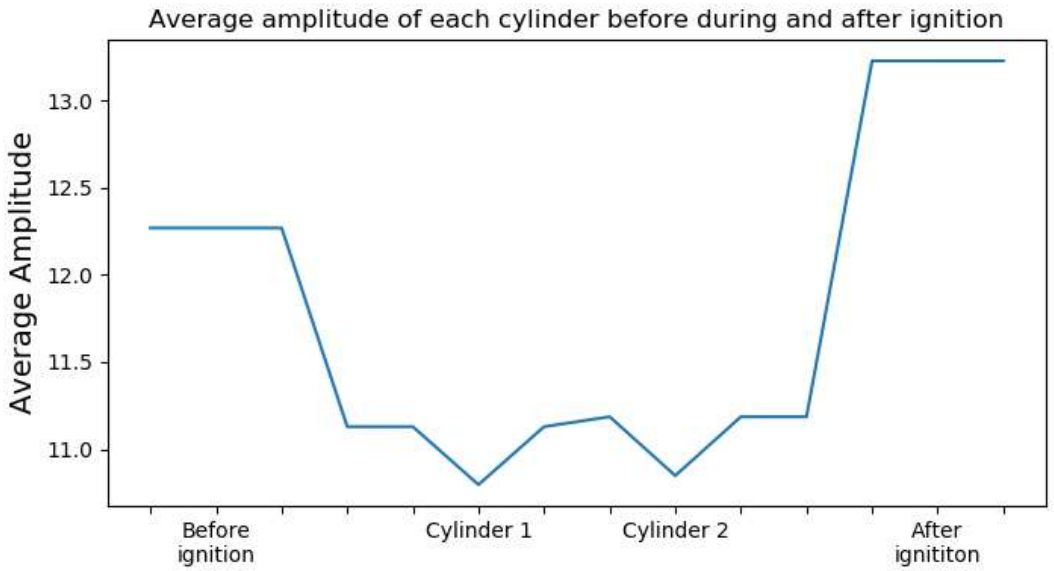

Figure 24: Graph representing the average drop in each cylinder and the average of the battery before and after the cranking stage before taking out the fuel.

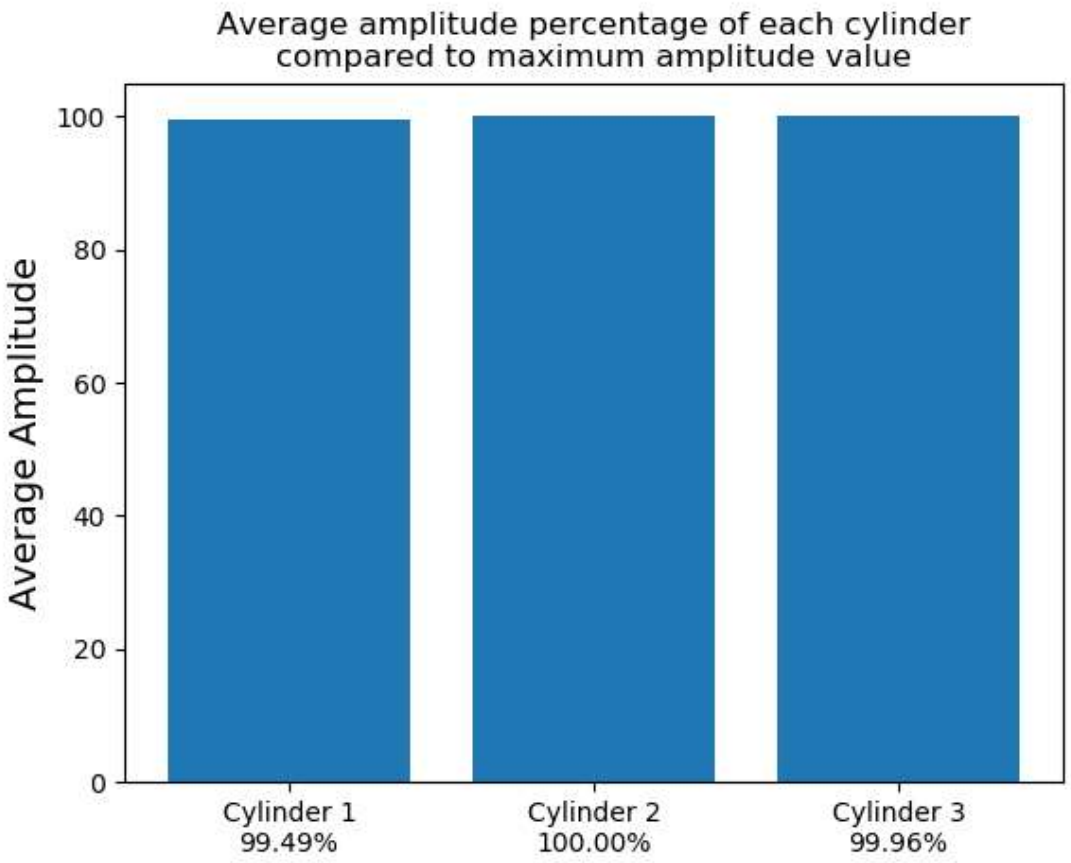

Figure 25: Graph representing the percentage of each cylinder compared to the maximum value in the data set before taking out the fuel.

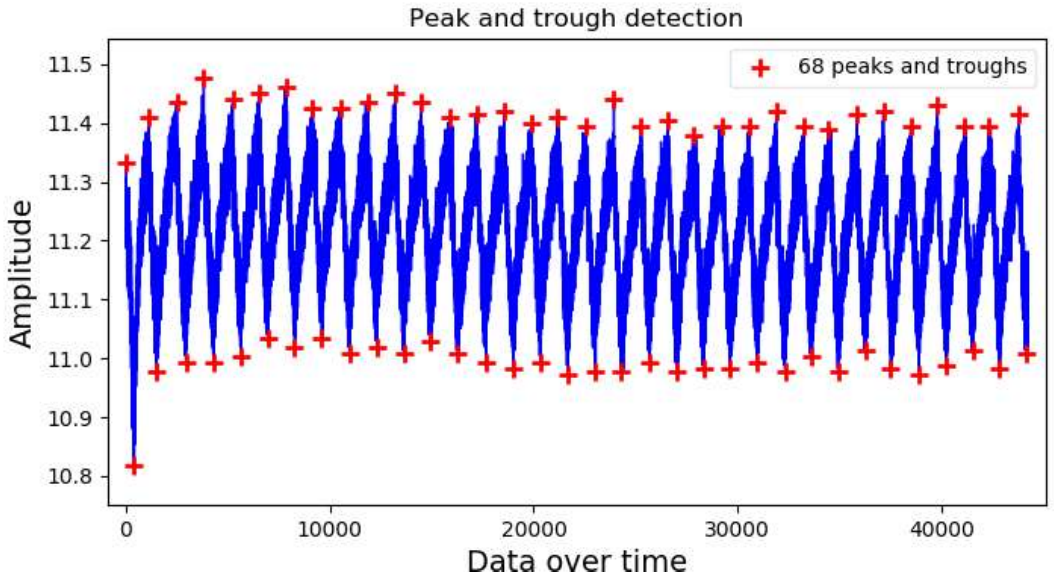

Figure 26: This is the signal during ignition for the second car with peaks and troughs detected after taking out the fuel.

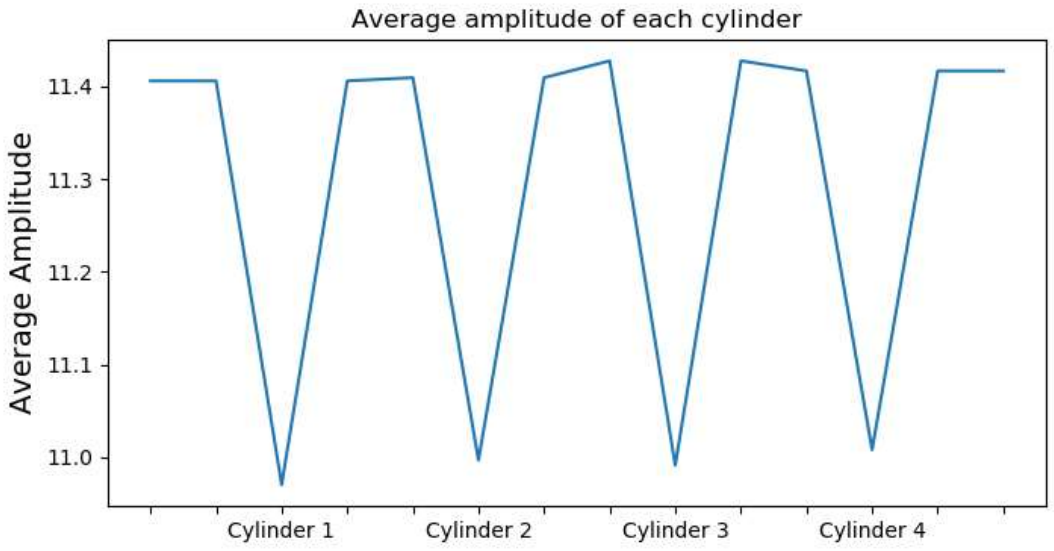

Figure 27: Graph representing the average drop in each cylinder after taking out the fuel.

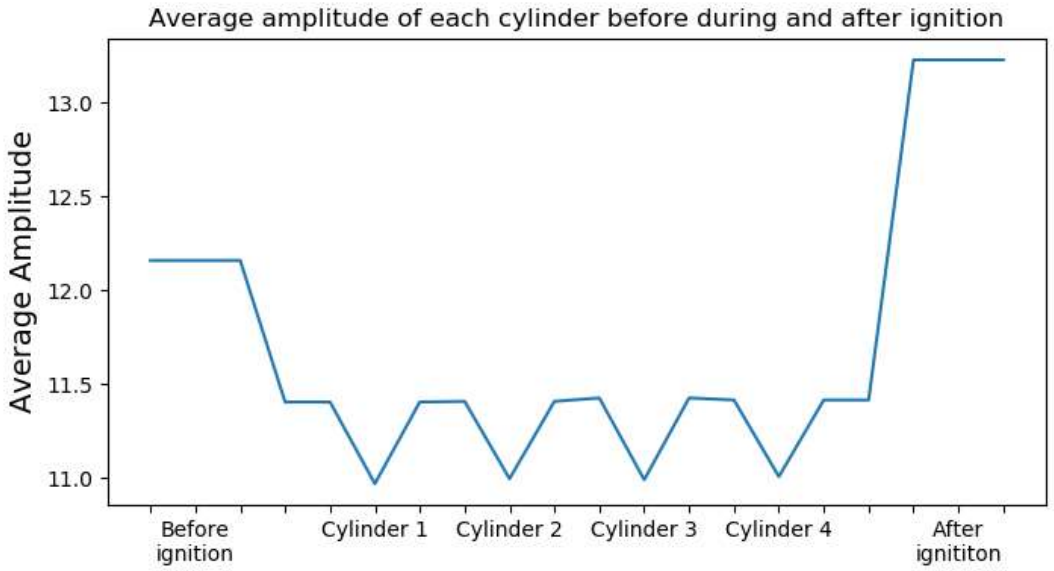

Figure 28: Graph representing the average drop in each cylinder and the average of the battery before and after the cranking stage after taking out the fuel.

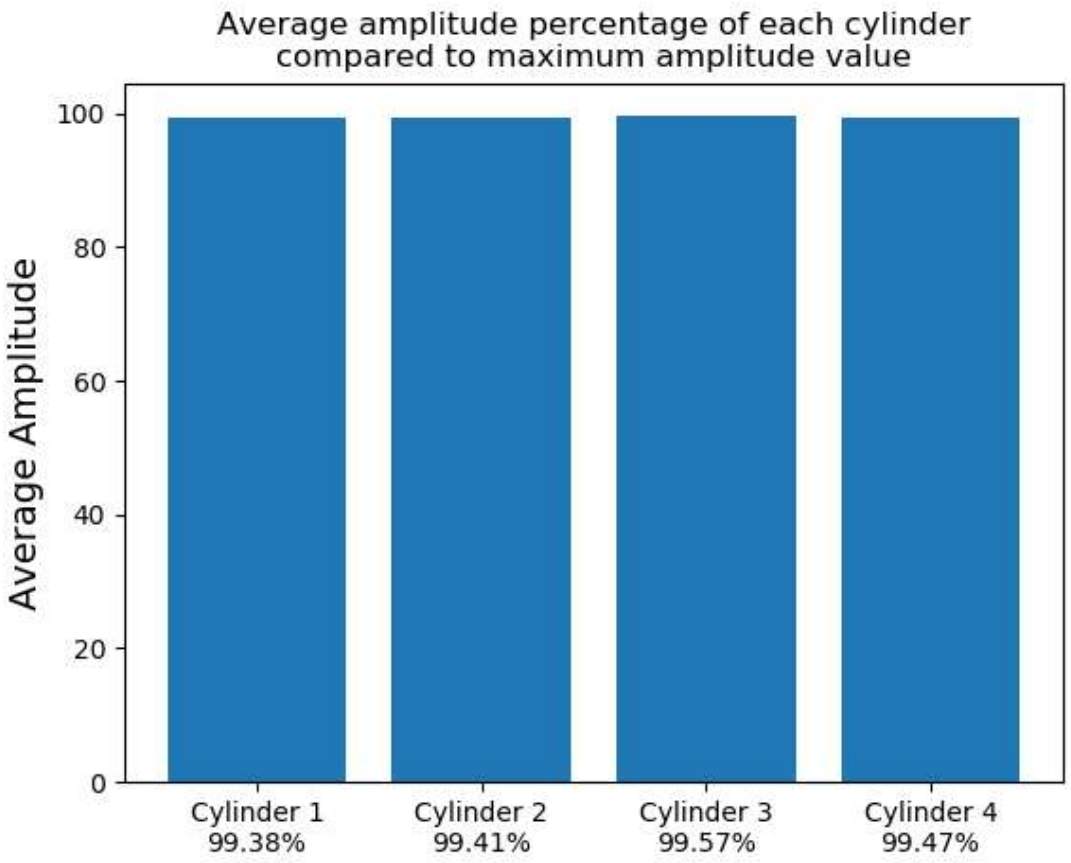

Figure 29: Graph representing the percentage of each cylinder compared to the maximum value in the data set after taking out the fuel.

5.2.3 Evaluation

The second car gave us some very good results. As it can be seen in the graphs the engine is evenly compressing at each cylinder with the drops being at even levels at each cylinder and the percentage of performance at each cylinder being above 99%. Also the levels of the battery before and after the ignition are as desired.

5.3 Evaluation of testing

Overall we are very happy from the results of testing as the system is working as desired and we got some very good results from the second car. The system is producing the desired result graphs and everything is working fine.

6 Conclusion

This project has given a brief insight how car engines work and how the ignition phase can provide data that are related to the engine's health before testing the system. Then a device was programmed to read voltage and present the data and various reports out of them.

The aim of this project was to create a cheap alternative to the OBD. Investigation into similar applications to the project idea led us to find that alternatives exist but are very expensive.

When the system was tested in the case of the first car the results were not very representative as the car starts very fast not giving enough time to be able to get enough data to produce representative results. In the case of the second car the results were very good as the fuel was disabled giving more time to collect data about the cranking stage.

Overall we are very pleased by the results of the project as the aim was met and the results collected were very helpful and is a cheap alternative to the existing solutions. This therefore suggests that the project was successful.

References

about circuits, A. (2016). Voltage Divider . *https://www.allaboutcircuits.com/tools/voltage-divider-calculator/*. [Online].

Controls, J. (2018). *https://www.autobatteries.com/en-us*. [Online].

Duarte, M. (2017). Peak Detect Function. *https://github.com/demotu/BMC*. [Online].

Heywood, J. B. (1988). *Internal Combustion Engine Fundamentals*. McGraw-Hill, Inc.

ignition (2017). The columbia encyclopedia. (7th ed.). *http://search.credoreference.com.ueaezproxy.uea.ac.uk:2048/content/entry/columency/ignition/0?institutionId=1278*. [Online; Accessed 24 October 2017].

Innovations, S. (2017a). Custard Pi 2A - Analogue to Digital Converter . *http://www.sf-innovations.co.uk/downloads/cpi2_an_in.txt*. [Online].

Innovations, S. (2017b). Custard Pi 2A - General Purpose input/output board for the Raspberry Pi GPIO . *http://www.sf-innovations.co.uk/downloads/cpi2doc.pdf*. [Online].

internal-combustion engine (2016). The Hutchinson unabridged encyclopedia with atlas and weather guide. *http://search.credoreference.com.ueaezproxy.uea.ac.uk:2048/content/entry/heliconhe/internal_combustion_engine/1?institutionId=127*. [Online; Accessed 23 October 2017].

internal-combustion engine (2017). In P. Lagasse and Columbia University, The Columbia encyclopedia. (7th ed.). *http://search.credoreference.com.ueaezproxy.uea.ac.uk:2048/content/entry/columency/internal_combustion_engine/0?institutionId=1278*. [Online; Accessed 24 October 2017].

Ofria, C. (2018). A Short Course on Charging Systems. *https://www.carparts.com/classroom/charging.htm*. [Online].

Pearson (2011). Diesel Engine Starting Systems. *https://www.pearsonhighered.com/assets/samplechapter/0/1/3/2/0132373629.pdf*. [Online].

Raspbian (2016). Raspbian. *https://www.raspberrypi.org/documentation/raspbian/*. [Online].

RPi (2016). RaspberryPi. *https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/*. [Online].

Snap-on (2015a). Verus Vantage. *https://www.snapon.com/display/1060/Vantage-Ultra/VantageUltra_lg.jpg?* [Online].

Snap-on (2015b). Verus Vantage. *https://www.snapon.com/diagnostics/us/DiagnosticPlatforms.htm*. [Online].

starter, N. A. (2005). Starting motor. *http://northatlan.com/Starterdrives.htm*. [Online].

Tutorials, I. (2014). Requirements of the Starting System (Automobile). *http://what-when-how.com/automobile/requirements-of-the-starting-system-automobile/*. [Online].

Windsor (2011). *https://courses.washington.edu/engr100/Section_Wei/engine/UofWindsorManual/Four%20Stroke%20Cycle%20Engines.htm*. [Online].