The design and prototyping of a controller and sensoring system to increase the safety of a racing car and driver.

Table of Contents:

1. Abstract

I first chose this project idea because I have a passion for motorsport. While completing this project I have come to realise that there is a high level of health and safety already implemented in motorsport at a professional level. However, from the perspective of an amateur driver, I believe that certain tweaks can be made to make it better.

My initial idea for this project was to put a proximity sensor in the car to allow the driver greater awareness that is not impeded by his lack of vision. Therefore, I realized I needed a proximity sensor mounted in the blind spot of the car. For the design I was referring to my experience in racing in which the blind spots are located at the rear of the car.

I then began to research different safety devices and sensor types mounted on professional race cars, Formula One to be specific, as this sport is a keen interest of mine. I then decided that my system should have multiple sensors. To attempt this, proximity sensor would be mounted to the blind spot of the vehicle. Temperature sensors would be mounted to the tires and engine as well as pressure sensors mounted in the tires. An oil level sensor would be mounted in the oil sump and accelerometers mounted to the suspension.

Each of these sensors play a crucial role in making the car safer for the driver by alerting the driver of hazards while driving. It also gives the driver more information about the car allowing them to adjust the setup of the vehicle while they are still at the track. This allows them to produce faster lap times and gain more knowledge on how the setup affects the car on the track.

To start, I had to decide on a microcontroller to power and output the data from the sensors to multiple outputs. Therefore, I chose a raspberry pi 4 model B with 8GB of RAM. Although 2GB or 4GB would be more than enough for this number of sensors and outputs I wanted to ensure I had the most powerful microcontroller available. The coding languages that can be used with the Pi are Python or C++. Python is my chosen programming language for the Pi due to its simplicity in comparison to C++.

I will build my system using prototyping breadboards and will test each sensor independently and then in conjunction with each other. Some sensors may require external power and I will be using one Bluetooth sensor for this project.

1. Acknowledgements

I am deeply indebted to my tutor Nader Jarmooz. Thank you for helping me come up with the initial idea for my project and for supporting me throughout the project. At the beginning of this course, I was at a total loss and struggled to develop a starting concept. The support I have received has been second to none and I don't think I could have done it without you.

I would like to express my deepest appreciation for my Open University lecturers. I look back on my time at university fondly as it has been a long journey which started back in 2018. I couldn't imagine writing this project or achieving what I have done without the help of university staff and colleagues.

Lastly, I'd like to mention my partner and my family for supporting me through this long journey. Although my bachelor's degree is coming to an end, I can look back on my achievements and be proud of my accomplishments to date.

3. Declaration

I declare that this project is an original report of my research. It has been written by me and has not been submitted for any previous degree. The experimental work is entirely my own work unless referenced and acknowledged.

4. Introduction

4.1 Background

The advantages of safety systems in race cars of the modern era have astronomically increased driver safety by reducing the number of deaths caused by racing accidents (Andrews, C, 2017). The systems that are at the forefront of driver safety are normally only implemented in professional racing as they can be impractical on a road car. These systems are also used by mechanics, engineers, and drivers to set the car up for specific racing conditions (Jones, A, B, et al. 2007). Professional teams have a vast array of personnel and equipment to ensure the car is at peak performance for the entire race (Ravelli, U, 2021). Being a hobbyist racing driver is hard as you must play the role of driver as well as chief engineer/mechanic, trying to set the car up with minimal or any equipment.

The other issue is most hobbyist racing drivers don't have the facility to carry around a lot of equipment and it is mostly left in the pit or car park of the race track we attend. Some can trailer their car to the track whereas others, like myself, need to drive. The necessary requirements are extra equipment which needs to be stored in the car and taken to the track. Therefore, my system must either be able to fit in the car and stored away or be on the car going to the track.

The main danger when driving around the track is not being aware of your track position or surroundings (Mourão, P, 2017). Other drivers are always trying to overtake you whilst you do the same. Therefore, my system will alert the driver when there is a car within their blind spot as the driver's vision can often be inhibited by the size of the mirrors and the safety equipment in the car such as a helmet which restricts eyesight (Jennings, A, unknown).

4.2 Requirements

The safety system I have designed will be built with 5 different sensors. A proximity sensor which is mounted on the car will be mounted in the rear and will be connected to an LED output which would be mounted on the dash (Kejik, P, et al. 2004). This allows the driver to judge the distance of a vehicle within its blind spot

also highlighting what side the vehicle is positioned on. A pressure sensor that would be mounted to the tires of the car will measure the change of pressure during a lap and could highlight the reason for loss of grip or give an indication that the driver could have a tire issue and needs to stop the car before a dangerous incident occurs. Also mounted to the tire will be a temperature sensor but this will also be attached to the engine. The temperature sensor will be used to allow the driver to see the temperature of the engine and tires allowing them to keep the car within its peak performance window. Finally, I will fit accelerometers around the suspension of the car which will allow me to monitor the movement in the suspension and possibly indicate where adjustments must be made to allow the car to drive better.

Only the temperature sensor will be Bluetooth, the other sensors will be hardwired to the microcontroller which will output the data to the displays.

To be able to control the sensors and retrieve the data, I will use a microcontroller. In this case, I decided to use a Raspberry Pi 4 due to its affordability and relative simplicity in usage. It was an obvious choice for me as it can also be coded with multiple languages.

The Pi will display the data to the driver on a monitor. However, this will only be displaying live data that is required to drive the car or alert if any issues appear. LEDs will be used for the proximity sensors that would be mounted to the dashboard within the driver's line of sight. The data will be stored in local storage so it can be interpreted after the race.

5. Project Methodology

5.1 Project aims and thesis

The overall aim of this project was to create a system that when implemented would increase the overall safety of hobbyist racing drivers and give them the tools to analyse their car track performance. This would give them a clearer picture of what could be going wrong as well as giving them a greater insight into what is happening with their car on the track.

At the end of this project, I plan on having a working prototype that could potentially be mounted to a car if I wished. However, this prototype will only be tested on a workbench in a controlled environment. All safety precautions will be taken during testing. Personal Protective Equipment (PPE) will be worn including, but not exclusive to, gloves, safety goggles, ESD wrist strap, and barrier cream if working with chemicals.

5.2 Approach

To approach this project, I first had a meeting with my tutor. During the meeting we discussed multiple options for a project. Some options were either too grand or too

small. After deciding on this project, I first had to decide what microcontroller to use. I settled on a Raspberry Pi 4. This microcontroller was selected due to it being affordable and not overly complex to use for this project (Ariyanto, M, 2019). It can be coded using C++ or Python and for this project I decided to code with python as I had previous experience with the coding language (Monk, D.S, 2016). The Pi can control multiple sensors due to its GPIO pins.

After this, I then had to look at what I wanted to achieve for my project. I wanted to be able to gain data from the car that would be beneficial not only during driving but also off track while the user is adjusting. Therefore, I decided I wanted to track the tyre pressure, brake and engine temperature, the G-forces experienced throughout a lap and the oil level (for leaks).

The main part of my project is the proximity sensors which in full scale would be mounted to the blind spots of the car. This will be displayed using multiple LED outputs. The number of LED lights will indicate how close a car would be from 1 LED indicating far away to multiple LED's indicating closeness. To measure this, I will use an ultrasonic sensor. The LEDs will be mounted to the dash in line of sight of the driver.

To measure the tyre pressure, I will use a barometric pressure sensor specifically MPL3115A2-12C from Adafruit as it will measure pressure, and temperature and it will give an altitude. This will be useful when looking at the data off track and will give the user a greater track awareness which allows them to pinpoint the track location of the data.

The engine will be fitted with a Bluetooth temperature gauge which will be attached to the engine block. The sensor I have chosen is independently powered via an external battery allowing it to gain data wirelessly. The issue with this method is that the battery level will need to be monitored by the user if used in full scale.

To measure the force around the track, accelerometers will be attached to the car suspension. Multiple sensors will be used but, during testing, I will only use one to test and prove that my system works. The sensor will allow the user to gain a greater appreciation of the forces experienced around the track and will demonstrate how stable the overall set-up of the car is. This could also highlight the components that need to be adjusted to have an optimum set-up.

I selected sensors that specifically worked with my microcontroller, and I had planned on using a Bluetooth accelerometer. However, in testing, it was proving more difficult to use and the data was inconsistent. The temperature sensor will allow me to connect to my microcontroller without the need for modification to the sensor.

The sensors will be outputted to a screen, and I will use my pc monitor screen for testing along with a small screen that works directly with the pi. It is important to note that this screen may not be able to display the data in real time. All the data will be saved in local storage on the pi as this will allow the data to be examined off track.

The data displayed in the car will only be for a driver's aid so any data not required as such will be stored for later use.

My design will be influenced by my research. When researching how formula one cars are designed, I realised the magnitude of sensors and canbus systems that are fitted to the car. 'Design of Formula One Racing Car' (Triya, Nanalal, Vadgama, et al. 2015) conveyed that race cars are made from multiple sensor systems however, Formula One cars are not fitted with any blind spot sensors which is 'suspected' to keep the racing spirit alive. The recent safety regulations that implemented the halo device as shown in the paper 'Effect of Halo Protection Device on the Aerodynamic Performance of Formula Race car' (Lin, M, et al. 2020), highlights the aerodynamic effect of the halo but also shows the reason behind its implementation. This also aligns with (Monroe, C, unknown), highlighting the importance of implementing this safety feature. Although this safety feature makes the cars dramatically safer which has been proven in races, there are serious implications of this which include blocking the vision of the driver. I believe implementing a proximity sensor alongside the recent features, will make the car even safer than they are currently in 2022. I would argue that this would enhance racing while keeping the enjoyment of the sport.

The use of proximity sensors are used today in modern cars which is an added option and is only available in higher-end models. Therefore, it may be some time before this is a normal safety feature like ABS (anti-lock braking system). Proximity sensors have been proven to effectively reduce the number of accidents increasing driver safety without distracting the driver (Raj, S, et al. 2016).

Where my system differs from a normal sensoring system in cars (D'Orazio, L, et al. 2011) is that there is instant feedback to my microcontroller and therefore the driver can be instantly notified. Any data spikes or unusual data coming from the sensors will alert the driver that a fault may have occurred before it causes a serious accident on the track.

As my system is for racing, it will need to comply with multiple safety regulations of the type of racing it will be used for which will ultimately be down to the end user to ensure this (Vadgama, T, N, et al, 2015). In my experience, all electronic systems must be connected to a general kill switch so that in the event of a crash it can be used to shut down the vehicle without endangering others. As my vehicle has this hardware fitted to it, if I tested full scale then I would ensure the microcontroller power is derived from the kill switch power line.

As my vehicle already has electronic hardware in place such as a speedometer, I am not required to make those safety changes to the hardware that is already in place. However, the battery voltage may be another factor that I could monitor by using a voltage divider circuit to drop the voltage to the pi (SinghPannu, G, et al, 2015). Then in the code, I would multiply the output by the reduction to give me an accurate measurement of the battery percentage minus voltage loss which would be minimal as I will not be testing at full scale. The rest of the paper shows the coding and

design of the system used to display racing speed which could be something that I could replicate with a hall effect sensor. However, as previously stated, I will not do this as my current speedometer is more than accurate enough for my needs and I will not be testing this inside the vehicle. (Wan, S, et al, 2014).

5.3 Design of the system

As previously stated, my system will be composed of a multitude of sensors. All have been bought specifically for compatibility with my microcontroller. With little requirement for voltage reduction or rectification. However, if the microcontroller was powered by the car's battery, the voltage would need to be reduced. The raspberry pi's operating voltage is 5 volts therefore the reduction is calculated as follow

$$Voltage\ reduction = V_{out} = V_{in} * \frac{R_2}{R_1 + R_2}$$

$$Voltage\ reduction = V_{out} = V_{12} * \frac{1400}{1950 + 1400}$$

$$Voltage\ reducion = V_5 = V_{12} * \frac{1400}{1950 + 1400}$$

As shown the calculated resistances are: 1400 ohms and 1950 ohms. Due to 1950-ohm not being a standard value it would be rounded up to 2000-ohm resistor.

Another option would be to use a bridge rectifier chip like the LM2596S. This would convert the 12 volts into 5 volts reducing the need for a resistor circuit.

The components will be mounted on a breadboard attached to the microcontroller's GPIO pins. The raspberry pi 4 pinout shown in Figure 1:

(Figure1) *IMAGE REDACTED FOR COPYRIGHT REASONS*

The general input pins will be used for data and the voltage pins will be used to power the sensors. Each sensor has a different operation voltage; therefore I will use the correct corresponding voltage.

5.4 Testing

I will independently test every sensor to ensure my code works as expected. I will then combine the sensors one at a time to minimise faults appearing and help with debugging.

Proximity sensor

The proximity sensor used during testing was an ultrasonic proximity sensor HC-SR04. This functions by sending ultrasonic waves out in pulses and then detects the distance by measuring the time taken for the ultrasonic wave to return.

An example calculation is shown below Speed is the speed of sound and time 5 seconds.

$$Distance = Speed\ x\ Time$$
$$D = 343 * 5$$
$$D = 1715\ meters$$

To wire the sensor to the Pi it was wired as follows:

- VCC wired into Pin 4 (5V)
- GND to pin 14(GND)
- TRIF to pin16 (GPIO23)
- ECHO to Pin18(GPIO24)

As the sensor is powered by 5 volts the voltage will have to be reduced as the GPIO ports can only tolerate 3.3 volt input. This was realised after reading the data sheet for the senor which is attached.

The resistors values were calculated using the voltage reduction equation below:

$$Voltage\ reduction = V_{out} = V_{in} * \frac{R_2}{R_1 + R_2}$$
$$Voltage\ reduction = V_{out} = V_5 * \frac{1400}{1950 + 1400}$$
$$Voltage\ reducion = V_{3.3} = V_5 * \frac{470}{330 + 470}$$

I used these values as I had these resistors to hand and my equation proves that it will reduce the voltage to make it safe for use.

The 330 ohm resistor is connected in between ECHO and Pin18 and the 470 ohm resistor is between the GNG and pin 14.

Insert circuit pic

The following script is the code I used.

To start I updated the GPIO library on the Pi this will not need to be done again.

```
#Libaries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 23
GPIO_ECHO = 24

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Distance = %.1f cm" % dist)
            time.sleep(1)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Measurement stopped")
        GPIO.cleanup()
```

Every second the distance will be measured until I stop the script with CTRL+C.

The following distances were measured using my hand-moving testing at different ranges.

proximatry	displayed distance
10cm	9.8cm
10cm	9.8cm
15cm	14.7cm
15cm	14.8cm
30cm	29.8cm
35cm	35cm
40cm	39.8cm
45cm	44.6cm
50cm	50.2cm
55cm	55.3cm
60cm	59.6cm
65cm	65.2cm
70cm	69.8cm
75cm	75.1cm
80cm	80.3cm
85cm	84.6cm
90cm	89.9cm
95cm	95cm
100cm	100.2cm

The data will be outputted to LED's later in testing.

Pressure sensor

The pressure sensor used during testing was the MPL3115A2. This functions as a temperature, pressure, and altitude sensor.

This sensor is powered by the 3-volt output from the Pi, therefore the voltage reduction will not be required.

To wire the sensor to the Pi it was wired as follows:

- Vin wired into Pin 1 (3V)
- GND to pin 9(GND)
- SCL to pin5(GPIO03)
- SDA to Pin3(GPIO02)

To start I had to manually install the Adafruit libraries for this sensor using the following script:

```
adafruit_mpl3115a2.mpy
adafruit_bus_device
```

This is the entire script written. This script is simple as the library has been preloaded. I only needed to create a loop to read the data from the sensor.

```
# Will read the pressure and temperature and print
import time
import board
import adafruit_mpl3115a2


# Create sensor object,communicating using the Pi's I2C bus
i2c = board.I2C()

# Initialize the MPL3115A2.
sensor = adafruit_mpl3115a2.MPL3115A2(i2c)


# configureing pressure at sea level for more accurate reading
sensor.sealevel_pressure = 102250

# Main loop to read the sensor values and print.
while True:
    pressure = sensor.pressure
    print("Pressure: {0:0.3f} pascals".format(pressure))
    altitude = sensor.altitude
    print("Altitude: {0:0.3f} meters".format(altitude))
    temperature = sensor.temperature
    print("Temperature: {0:0.3f} degrees Celsius".format(temperature))
    time.sleep(1.0)
```

The Pi will read the data from the sensor every second and print this data on the monitor.

The following data was taken during testing:

temperature	altitude	pressure
23.75	130	103040
23.71	130	103040
23.7	130	103040
24.5	130	103040
26.85	130	103040
28.86	130	103040
32.65	130	103040
35.67	130	103040
43.21	130	103040
45.58	130	103040
48.67	130	103040
53.54	130	103040
57.69	130	103040
64.49	130	103040
70.49	130	103040
72.54	130	103040
72.53	130	103040
72.51	130	103040
72.5	130	103040
72.48	130	103040
72.42	130	103040

Temperature sensor

The temperature sensor is a digital temperature senor on a voltage reduction board. This means that I do not need to attach any resistors to the circuit.

The sensor circuit is powered by 3 volts and the sensor will be wired as follows:

- Vc wired to pin 17 (3-volt supply)
- GND wired to pin 25 (Ground)
- Digital Output (Do) wired to pin 19 (GPIO)
- The LCD is connected as follows

One-wire interface needs to be enabled for this sensor to work. To do this, I entered the following into the command prompt:

```
sudo nano /boot/config.txt
```

This is then changed to include the added code below at the bottom of the file:

```
dtoverlay=w1-gpio
```

The Pi is then rebooted. This edit to the code configures the one wire interface to open when the Pi is booted.

The following code is then entered:

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
cd /sys/bus/w1/devices
ls
```

This code allows me to see the address of my sensor attached it then enter the following code to finish the one-wire interface.

```
28-000006659634 w1_bus_master1
cd 28-000006659634
cat w1_slave
```

This code will display raw temp data but the following code will display it on my SSH terminal:

```
import os
import glob
import time

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f

while True:
        print(read_temp())
        time.sleep(1)
```

The Pi was tested for 30 minutes to ensure I had enough data. The following is some data from the tests:

temperature
23.75
23.71
23.7
24.5
26.85
28.86
32.65
35.67
43.21
45.58
48.67
53.54
57.69
64.49
70.49
72.54
72.53
72.51
72.5
72.48
72.42

I had hoped to allow this sensor to send data via Bluetooth to the Pi, however, I could not get the Pi to see the sensor. I do believe it worked as I could see the sensor on my mobile phone, but I still could not connect.

My code is either incorrect or there is an issue with the device, perhaps my OS could be updated.

As I can prove the sensor works whilst wired, I am content that I can still get data.

Accelerometer

The accelerometer I selected was the ADXL345. This accelerometer works directly with the Pi which mitigates the requirement from an analogue to a digital converter.

The accelerometer was wired as follows:

- GND pin to Pin (GND)
- VCC pin to pin017 (3-volt supply)
- SDA pin to pin 3(SDA)
- SCL pin to pin 5(SCL)

For this sensor to work, I2C must be turned on. As this step was already completed before, it will not be required. However, if anything in the set-up is changed it will need to be activated again.

The following command is entered to ensure the Pi can see the sensor:

```
sudo i2cdetect -y 1
```

Once detected, I then installed the Adafruit ADXL34x Python library

```
sudo pip3 install adafruit-circuitpython-ADXL34x
```

This then allowed me to write the following script:

```
import time
import board
import busio
import adafruit_adxl34x

i2c = busio.I2C(board.SCL, board.SDA)
accelerometer = adafruit_adxl34x.ADXL345(i2c)

while True:
    print("%f %f %f"%accelerometer.acceleration)
    time.sleep(1)
```

The accelerometer data can now be read by typing the following script:

```
python3 ~/accelerometer.py
```

This then runs the previous script starting the accelerometer and receiving data. This data is then printed on the monitor used during testing. At full scale this data will not be used by the driver when the vehicle is in operation. The data will be saved and analysed when required.

The following data was received during testing. Testing was done 2 hours prior to ensure there were no flaws in the present data. There is a small amount which is shown below:

x	y	z
0.302536	0.23536	0.352039
0.302549	0.234487	0.32455
0.303549	0.235498	0.354889
0.304821	0.235498	0.365498
0.302578	0.235655	0.365419
0.302458	0.268942	0.364649
0.324856	0.214579	0.321654
0.347856	0.265498	0.345898
0.365874	0.246355	0.324985
0.358741	0.265498	0.347895
0.324785	0.265877	0.365478
0.332148	0.265875	0.324985
0.336585	0.248512	0.35644
0.334591	0.215789	0.365498
0.336575	0.236548	0.329655
0.324786	0.214579	0.365984
0.32479	0.265874	0.354198
0.324579	0.24579	0.365498
0.324569	0.235479	0.315645
0.329548	0.236548	0.349845
0.321569	0.21547	0.365198
0.326597	0.214579	0.365498

There were no anomalies during testing and the accelerometer seemed to function as required.

Proximity sensor with LED

After a discussion with other hobbyist drivers, I decided that more than one LED output could be a distraction. This could be implemented if I used the proximity sensors as a reversing sensor.

Therefore, I only used one LED per sensor, this would be repeated with the multiple sensors used if I was to test this in full scale.

In order to power the LED, it was wired in as follows:

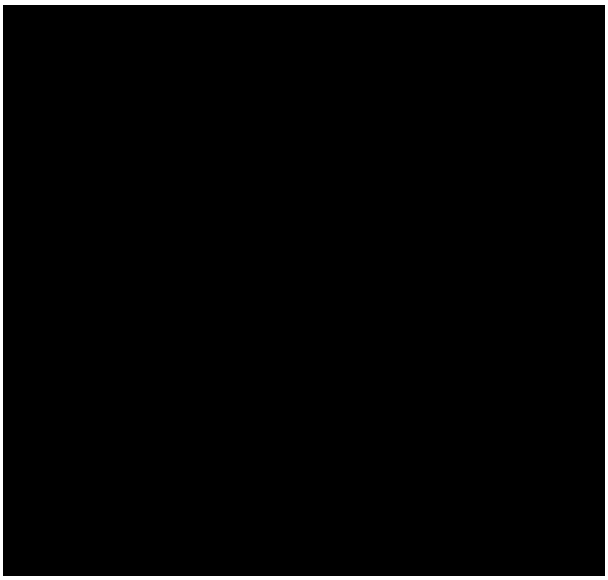
- Cathode (positive) to 220-ohm resistor
- 220-ohm resistor to pin32(GPIO12)
- Anode (negative) to pin34 (GND)

The script for the proximity sensor will remain the same. However, I had added an extra script for the LED:

```
#Libraries
import RPi.GPIO as GPIO
import time

LED_PIN = 36
# LED output pin
GPIO.setup(LED_PIN, GPIO.OUT)


#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 18
GPIO_ECHO = 24

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Measured Distance = %.1f cm" % dist)
            time.sleep(1)
        # Turn ON LED
        GPIO.output(LED_PIN)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Measurement stopped by User")
        GPIO.cleanup()
```

Below is the data that I took from the function test of the proximity sensor:

proximatry	LED
10cm	yes
10cm	yes
15cm	yes
15cm	yes
30cm	yes
35cm	yes
40cm	yes
45cm	yes
50cm	yes
55cm	yes
60cm	yes
65cm	yes
70cm	yes
75cm	yes
80cm	yes
85cm	yes
90cm	yes
95cm	yes
100cm	yes



(proximity sensor)

IMAGE REDACTED FOR COPYRIGHT REASONS

Evaluation of the System

I believe my system is well designed and functional during testing. As shown, there were many faults, but I was able to rectify them through debugging. I was unable to get the data via Bluetooth to the raspberry Pi, but I was able to get temperature and acceleration data by hardwiring. My sensor choice could have been better in this regard.

As I was prototyping, I was never able to test my design in a vehicle and unfortunately, I cannot comment on the full build of the system Despite this, I believe that my design could potentially save the life of a driver if it was fully implemented. Therefore, I plan to push forward with my design and hopefully implement it in the future. Currently, I am unable to race my car but if the right opportunity occured, I would add the proximity sensor to the car as it will increase my safety.

I believe my system meets the requirements I had set with regard to the increase in driver awareness and safety as well as the ease of use. The extra sensors will also help with adjustments to the car but only if the information is used correctly which would ultimately be down to the individual user. I do believe that with enough information, the end user would be able to make an informed decision.

With regards to my choice of microcontroller, the Pi was the correct choice for me. Learning to code in python was simple enough as I have had previous experience. I was able to rectify errors within my code due to the vast range of information available about python which was available to me.

If I was building this to full-scale, I would require more microcontrollers as the number of sensors I require would be too much for one raspberry pi. I would have one microcontroller to power and receive data from the proximity sensors as well as another sensor which would be connected to the secondary Pi.

Overall, I am very happy with the overall results. I wish I could have got the Bluetooth devices to work with the Pi, but I will continue to try and get these to work. I will continue to practice with python as this language is becoming more widely adopted.

Bibliography

Andrews, C., 2017. Tech to keep drivers safe [Formula One motorsport]. Engineering & Technology 12, 76–77. https://doi.org/10.1049/et.2017.0307

Ariyanto, M., Haryanto, I., Setiawan, J.D., Munadi, M., Radityo, M.S., 2019. Real-Time Image Processing Method Using Raspberry Pi for a Car Model, in: 2019 6th International Conference on Electric Vehicular Technology (ICEVT). Presented at the 2019 6th International Conference on Electric Vehicular Technology (ICEVT), pp. 46–51. https://doi.org/10.1109/ICEVT48285.2019.8993866

D'Orazio, L., Visintainer, F., Darin, M., 2011. Sensor networks on the car: State of the art and future challenges, in: 2011 Design, Automation & Test in Europe. Presented at the 2011 Design, Automation & Test in Europe, pp. 1–6. https://doi.org/10.1109/DATE.2011.5763169

Jennings, A., n.d. Percent Area Visual Obscuration of F1 Racecar Canopies [WWW Document].

Jones, A.B., White, R., 2007. All Around the Track: Oral Histories of Drivers, Mechanics, Officials, Owners, Journalists and Others in Motorsports Past and Present. McFarland.

Kejík, P., Kluser, C., Bischofberger, R., Popovic, R.S., 2004. A low-cost inductive proximity sensor for industrial applications. Sensors and Actuators A: Physical, Selected Papers from Eurosensors XVI Prague, Czech Republic 110, 93–97. https://doi.org/10.1016/j.sna.2003.07.007

Lin, M., Papadopoulos, P., 2014. Effect of Halo Protection Device on the Aerodynamic Performance of Formula Racecar. International Journal of Mechanical and Mechatronics Engineering 14, 13–18.

Monk, D.S., 2016. Programming the Raspberry Pi: Getting Started with Python, Second Edition. McGraw-Hill Education.

Monroe, C., n.d. A Critical Review of the Halo Device in Formula One 8.

Mourão, P., 2017. The Economics of Motorsports: The Case of Formula One. Springer.

Navarro Gorgojo, P., 2021. Aerodynamic study of the wake effect on a Formula 1 car in curves.

Raj, S., Ezhilarasie, R., Umamakeswari, A., 2016. Zigbee-Based Collision Avoidance System in Blind Spot and Heavy Traffic using Ultrasonic Sensor. Indian Journal of Science and Technology 9. https://doi.org/10.17485/ijst/2016/v9i48/108015

RAVELLI, U., 2021. Aerodynamics of a 2017 Formula 1 Car: Design Improvements in Freestream and Wake Flows. https://doi.org/10.6092/978-88-97413-38-7

SinghPannu, G., Dawud Ansari, M., Gupta, P., 2015b. Design and Implementation of Autonomous Car using Raspberry Pi. International Journal of Computer Applications 113, 22–29. https://doi.org/10.5120/19854-1789

Triya Nanalal Vadgama, Mr. Arpit Patel, Dr. Dipali Thakkar, Sardar Vallabhbhai Patel Institute of Technology, 2015. Design of Formula One Racing Car. IJERT V4, IJERTV4IS040962. https://doi.org/10.17577/IJERTV4IS040962

Wan, S., He, C., Xie, M., Fan, Y., 2014. Design of Racing Electric Control System Based on AVR SCM 181, 8.