Driver's Safety Analyzer: Sobriety, Drowsiness, Tiredness, and Focus

by

Claudio Fernandes Dias

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science in Engineering

in the

Electrical Engineering

Program

YOUNGSTOWN STATE UNIVERSITY

May, 2020

Driver's Safety Analyzer: Sobriety, Drowsiness, Tiredness, and Focus

Claudio Fernandes Dias

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

4/13/2020

Claudio Fernandes Dias, Student Date

Approvals:

Dr. Coskun Bayrak, Thesis Advisor*

4/10/2020

Dr. Frank Li, Co-Advisor Date

4/10/2020

Dr. Faramarz Mossayebi, Committee Member Date

Dr. Salvatore A. Sanders, Dean of Graduate Studies Date

* Deceased

ABSTRACT

The Driver's Safety Analyzer was designed after deeply researching over the subject and realizing the need of it in the world. The car safety system, in a whole, is developed to protect the driver from having a car accident. Breaking the safety system in different parts, first, there will be the controlling system that prevents the car from leaving its lane if there is another car in its blind spot. Second, the system which automatically stops the car when approaching a vehicle. Third the system is developed to protect the driver from the car accident, such as the airbags, etc.

However, it is known that many car accidents are caused by the driver's irresponsibility. According to psychologists, the human beings are, by average, overconfident of themselves. They will always assume they are above average (more capable than an ordinary human). This behavior is what pushes the person to commit wrong decisions such as, driving after drinking alcoholic beverages, driving without sleeping, texting while driving, and driving for long periods of hours without resting.

The Driver's Safety Analyzer is developed to possibly solve the overconfidence behavior problem. Using microprocessor, microcomputer, and coding the system can monitor the driver's behavior and control the environment preventing the driver to commit wrong decisions.

ACKNOWLEDGMENT

First, I am greatly indebted to my advisor Dr. Coskun Bayrak, for his patient and support throughout all my journey as a graduate research assistant (GRA). I cannot put in words how honored I felt when I received the proposal for the GRA position from him. Thank you for giving me the chance to continue learning and improving myself as a student and as a researcher. The support provided by Dr. Coskun Bayrak made my road to complete my major, projects, and the research, easy. Without his trust in me and the commitment to provide anything I needed such as, parts, paperwork, 3D printings, doubts, and advices. I would not have been able to accomplish everything we proposed for this thesis. I hope he will be satisfied with my thesis and feel proud about all the time spent with it. In addition, I hope, one day, I will be able to reach his level of knowledge and dedication with his job, department, university, and all the students he advises.

I would like to express my gratitude to Dr. Frank Li, for his support at the beginning of my journey as an undergraduate student until my first semester as a graduate student. Without him warning me about the possibility of working as a graduate assistant and introducing me to Dr. Coskun Bayrak I would have never become a GRA. I am grateful for him giving me, first, the chance of study electrical engineering, and last, for advising me to pursue my master's degree. Thank you, Dr. Li, for all the trust you had in me and for all the time you supported me when I was struggling with problems.

I would like to thank my close friends Dibya Gautam, Karim Mohamed, Dillon Kennedy and Lei Chen for always being by my side, for helping me through every class taken at Youngstown State University (YSU). I will never forget all the days we used to

study and work on projects for hours and hours in order to be the best we could possibly be. In addition, I would like to thank you for the time we spend together, inside and outside the university, having fun and relaxing.

I cannot avoid mentioning my best friends Bryan Martinez and Donald K. Adams. When I first moved out from Brazil to United States, 5 years ago, I was alone. I moved to US by myself only knowing the International Department of YSU. Since the day I met Bryan and Donald, I felt, and feel, embraced and accompanied.

Additionally, I need to thank the community of microprocessors and microcomputers for helping newbies like me. All the guides and source code displayed online, the forums with many people willing to help each other are amazingly helpful. All my thanks to you, open source community.

At last, I have a special thanks to my mother Edileia Candelaria Fernandes, she is the reason why I was able to take the first step, all the way back in Brazil, until my last step here typing and submitting this thesis. The thrust she placed in me is unmeasurable, I am indebted with her forever for making all this possible. Thank you, mom.

Table of Content

List of Figures

I. INTRODUCTION

i. Background

With the analysis of the driver's state of mind and sobriety the thesis proposal and the project related to this study are to solve and prevent car accident from happening. The thesis is divided in two sections which will analyze the driver's capability to drive without causing warm to himself/herself or others. The alcohol detection system (ADS) will analyze the amount of alcohol in the driver's blood, deciding if the driver is sober and still capable of driving his car. The face features detection system (FDS) will be analyzing the driver from start of the car's engine until the end of the trip. FDS records the driver's face features, more specifically, the eyes. From the camera recording, FDS will be watching two main possibilities. The first possibility is related to how attentive the drive is while driving, if the driver's attention is in the main road. FDS will know if the driver is looking directly at the road while driving or if the driver is looking in sufficient duration to the road ahead of the vehicle. The second possibility analyzes the tiredness and fatigue of the driver. The system will count the duration the driver's eyes has been closed. Using that data, FDS will decide if the driver has kept his eyes closed for long enough and triggers an alarm.

ADS and FDS systems were built using open sources microprocessor and microcomputer. ADS was built using Arduino Uno and FDS was mounted under the microcomputer Raspberry Pi. Both systems could have been made only using Raspberry Pi, however, Arduino was chosen in order to maximize the learning and improvement throughout the student's Master of Science degree program.

ii. Motivation and Problem Definition

The main of transportation used in United States is the automobile, more specifically, the car. The Society uses cars to move from point 'a' to point 'b', in an efficient, fast and safe deallocation. A car would not be a choice if its safety is not validated.

With the invention of the automobile in the late 19th century, came the inevitable side effect of automobile collisions. As automotive collisions increased in frequency, it became clear that, unlike other ways, which relied on personal responsibility, there was a possibility that automobiles would need to be governed by law [1]. Insurance become obligatory (in most of the states) and laws have been transcribed to supervise and inspect the automobiles generating a controlled system for automobiles.

Crash tests from the National Highway Traffic Safety Administration (NHTSA) and the Insurance Institute for Highway Safety (IIHS) are very telling of how a vehicle will fare in an accident situation. Car brands like Volvo, Mercedes-Benz Opens and Honda pride themselves on the structural integrity of their vehicles. But beyond strength in construction, buyers should also be looking at active and passive safety features, ranging from airbags and car seat anchors to driver-assistance systems like lane-departure warning, blind spot monitoring, forward-collision warning and automatic braking, to name a few [2].

People, automobile manufacturers and insurance companies are concerned about the safety of the drivers and passengers. A total of 37,133 people died in motor vehicle crashes in 2017. The U.S. Department of Transportation's most recent estimate of the

annual economic cost of crashes is $242 billion dollars [3]. The contribution to the death toll are the alcohol level, speeding, lack of safety belt use and other problematic driver behaviors. Death rates vary by vehicle type, driver age and gender, and other factors [4]. In 2016, 10,497 people died in alcohol-impaired driving crashes, accounting for 28% of all traffic-related deaths in the United States. Of the 1,233 traffic deaths among children between the ages 0 and 14 years in 2016, 214 (17%) are involved an alcohol-impaired driver. In 2016, more than 1 million drivers were arrested for driving under the influence of alcohol or narcotics. That's one percent of the 111 million self-reported episodes of alcohol-impaired driving among U.S. adults each year. Drugs other than alcohol (legal and illegal) are involved in about 16% of motor vehicle crashes [4,5].

In 2017, there were 34,247 fatal crashes in the United States involving 52,274 drivers. As a result of those fatal crashes, 37,133 people were killed. There were 2,935 fatal crashes that occurred on U.S. roadways in 2017 that involved distraction (9% of all fatal crashes). These crashes involved 2,994 distracted drivers, and some crashes involved more than one distracted driver. Distraction was reported for 6 percent (2,994 of 52,274) of the drivers involved in fatal crashes. In these distraction-affected crashes, 3,166 fatalities (9% of overall fatalities) occurred. Much attention across the country has been focused on the dangers of using cell phones and other electronic devices while driving. In 2017 there were 401 fatal crashes reported to have involved cell phone use as a distraction (14% of all fatal distraction-affected crashes). For these distraction-affected crashes, the police crash report stated that the driver was talking on, listening to, or engaged in some other cell phone activity at the time of the crash. A total of 434 people died in fatal crashes that involved cell-phone-related activities as distractions [6].

Drowsy driving was reportedly involved in 2.3 to 2.5 percent of all fatal crashes nationwide from 2011 through 2015. In 2015, 2.3 percent (824) of the fatalities that occurred on U.S. roadways are reported to have involved drowsy driving. In 2015, the total number of fatalities increased by 7 percent compared to 2014, but the proportion reported to involve drowsy driving decreased, from 2.6 to 2.3 percent. The number of fatalities in crashes involving a drowsy driver between 2011 and 2015 has remained constant, fluctuating between 2.3 percent of fatalities and 2.6 percent [7].

The following table (figure 1) display the data from 2008 to 2017 of Death caused by car accidents, vehicle miles traveled by billions (VMT), fatalities per 100 million VMT, US population, fatalities per 100,000 population, and Change in per capita fatalities from previous years (CPCF). According to figure 1, 2015, 2016, and 2017 reached more than 11 fatalities per 100,000 population, a rate only reached before in 2010. Additionally, since 2015 the number of deaths by car accident is bigger than 35,000, what did not happen since 2008.

Year	Deaths caused by car accident	VMT	Fatalities per 100 million VMT	US Population	Fatalities per 100,000 population	CPCF
2018	36,750					
2017	37,133	3,213	1.16	326,213,213	11.40	▼-1.8%
2016	37,806	3,174	1.19	323,121,000	11.59	▲5.6%
2015	35,485	3,095	1.15	321,370,000	11.06	▲10.5%
2014	32,744	3,026	1.08	318,860,000	10.28	▼-0.9%
2013	32,932	2,988	1.10	316,129,000	10.40	▼-3.3%
2012	33,782	2,969	1.14	313,914,000	10.75	▲2.6%
2011	32,479	2,950	1.10	311,588,000	10.42	▼-2.3%
2010	32,999	2,967	1.11	309,326,000	10.67	▼-3.5%
2009	33,883	2,957	1.15	306,700,000	11.05	▼-9.7%
2008	37,423	2,977	1.26	303,824,640	12.32	▼-11.0%

Figure 1 Statistics Over the Years [8]

With nowadays technology towards automobile safety it is assumable the number of death and car accidents would decrease due to the new safety technologies. However, the number of deaths caused by car accident has increased if compared to previous years.

That said, the number of car accidents caused by distracted, drowsy and inebriate drivers, is about fifty percent of all car accidents in US. The thesis proposes to solve this problem using two systems. One, the alcohol detection system (ADS), the ADS analyzes the driver's state of mind and consciousness. ADS detects the amount of alcohol inside driver's blood deciding if the driver is sober or not. Two, the face features detection system (FDS). FDS monitors the driver's eyes, analyzing the eyes' direction, head position, and

the edges of the eyes. The data collected checks if the driver is attentive while driving, its rate of distraction, the tiredness and drowsiness of the driver.

iii. Significance

With the development of ADS and FDS, new doors are opened showing how the car companies or third-party companies could create or develop projects such as the one reproduced by this thesis and improve their cars' safety system. These efforts could result in a safer driving environment. The contribution and the significance of the thesis to the society are the following:

- **Source Material:** The idea of creating a project which evaluates the driver's capability of driving is scarce, so the project will contribute to the community and could serve as source material for further development and implementations.
- **Example and Guidance:** The thesis will be published and be part of the open data over the internet, adding more material, about this particular subject, to be used as examples for improvements. People, students, and researchers will be able to study and inspire themselves with the work presented in this thesis and use the material as guidance or part of their own projects, thesis, or study.
- **Driver's Safety:** With the capability of evaluating the state of mind of the driver, if he is drunk, tired, drowsed, distract, or asleep. The system will give feedbacks to the driver keeping him alert (if necessary) or it will lock

the car and block the driver from driving if he is not sober, concluding in helping the driver's safety.

- **Others Safety:** The project will be constantly monitoring and giving feedbacks to the driver, evaluating his capability of driving, and the amount of attention the driver is paying to the traffic ahead. In addition, the system will be taking actions according to the driver's state of mind. Furthermore, all the prevention created for the driver will ensure that the driver won't be driving carelessly, and as a result of it, it will indirectly increase the safety of other drivers and people around the driver.
- **Portability and Cost:** The prototype has a small size, which makes it easy to install and can be use in any car. With its small size, the same prototype can be moved and installed in different cars. In addition, the average cost to build the prototype is reasonable small.

iv. Contribution

The main reasons to address the proposed problems was the lack of material related over libraries and internet. Moreover, while reaching for source material and references, it was only founded one published project related to FDS and a few related ADS, however, with different applications.

Evaluating the driver's state of mind and level of alcohol are crucial in preventing car accidents concluding in saving the driver's life and others, who could be involved in car accidents caused by a driver without a good state of mind and sobriety.

With further implementation the thesis could be used in real car's systems and put into the real-world practice preventing the fifty percent of deaths by car accidents related to drunk, asleep, distracted, using cellphone, and drowsed drivers.

II. LITERATURE REVIEW

i. Arduino

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package [9].

In addition, Arduino is an open-source microprocessor, which makes it easy to find source code online and in forums. Additionally, for being famous and popular, Arduino has a big community of supporters and users that are willing to help each other with their projects and programming codes.

Figure 2 Arduino Uno

ii. Raspberry 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. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

The Raspberry Pi is one of the cheapest microcomputers available in the market. Furthermore, with its large compatibility and community, opensource based, it is the best microcomputer to develop prototypes that could suppress Arduino in some features for being a microcomputer and not just a microprocessor.

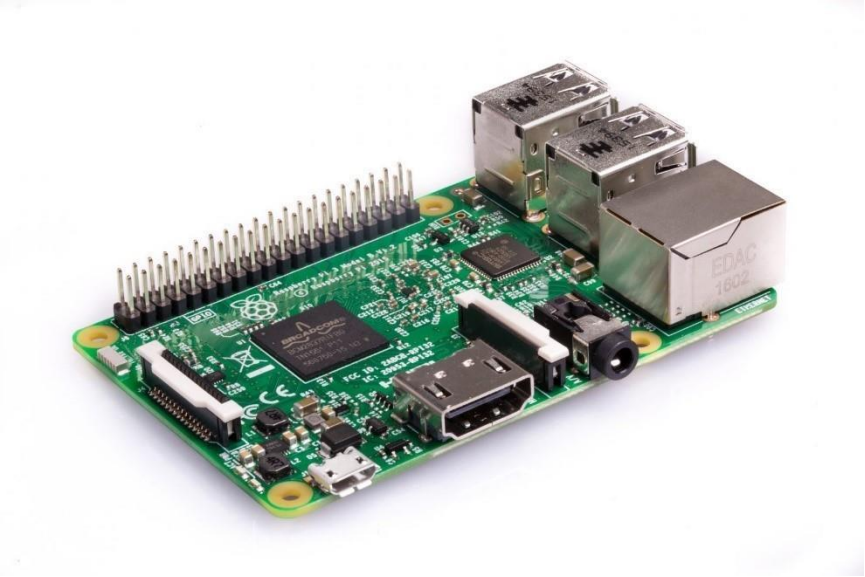

Figure 3 Raspberry Pi

iii. Alcohol Gas Sensor MQ-3

The MQ3 gas sensor is an alcohol sensor which is used to detect the alcohol concentration inside an environment. This sensor provides an analog resistive output based on alcohol concentration in the air. When the alcohol gas exists, the sensor's conductivity gets higher along with the gas concentration rising. It is suitable for various applications of detecting alcohol at different concentrations. It is widely used in domestic alcohol gas alarm, industrial alcohol gas alarms and portable alcohol detectors.

It is a low-cost semiconductor sensor which can detect the presence of alcohol gases at concentrations from 0.05 mg/L to 10 mg/L. The sensitive material used for this sensor is SnO2, whose conductivity is lower in clean air. It is conductivity increases as the concentration of alcohol gases increases. It has high sensitivity to alcohol and has a good resistance to disturbances due to smoke, vapor and gasoline. This module provides both digital and analog outputs.

In this project the MQ-3 sensor was used connected to the Arduino with the objective to analyze the level of alcohol in the driver's breath. The level of alcohol in the driver's breath will excite the MQ-3 sensor and change its resistance output, changing the current flowing through the analogy port of the sensor. The analogy current is analyzed by the Arduino. In addition, with Arduino coding, the value coming through the analogy port will be transformed in mg/L of alcohol particles. This value will trigger an Arduino output and this output will be connected to the car. So far, the output gives the written value of mg/L in an 16x2 LCD screen.

Figure 4 MQ-3 Alcohol Sensor

iv. Facial Recognition

The facial recognition used in the project was created using ubuntu 18.04 and programmed in Python (programming language). In the system it was necessary to installs pip, a package management system that simplifies installation and management of software packages written in Python. Pip was used to install NumPy, scikit-image, OpenCV3, OpenCV4, and Dlib. NumPy is a package that contains N-dimensional array object, sophisticated (broadcasting) function, tools for integration C/C++ and Fortran code, linear algebra, Fourier transform, and random number capabilities. Scikit-image (formerly scikits.image) is an open-source image processing library for the Python programming language. It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more. It is designed to interoperate with the Python numerical and scientific libraries NumPy [10]. OpenCV is a Python library which is designed to solve computer vision problems, OpenCV is a wrapped class for the original C++ library to be used in Python. In addition, all OpenCV array structures are converted to/from NumPy array. Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high-performance computing environments. Dlib's open source licensing allows it to be used in any application, free of charge [11].

Using OpenCV and Dlib facial landmark was detected in an image. The landmarks (key points) that are interesting for the device are the ones that describe the shape of the face attributes like: eyes, eyebrows, nose, mouth, and chin. These points gave a great

insight about the analyzed face structure, that can be very useful for a wide range of applications, including face recognition, face animation, emotion recognition, blink detection, and photography.

The Face Landmark Detection algorithm offered by Dlib is an implementation of the Ensemble of Regression Trees (ERT) presented in 2014 by Kazemi and Sullivan. This technique utilizes simple and fast feature (pixel intensities differences) to directly estimate the landmark positions. These estimated positions are subsequently refined with an iterative process done by a cascade of regressors. The regressors produces a new estimate from the previous one, trying to reduce the alignment error of the estimated points at each iteration. The algorithm is blazingly fast, in fact it takes about 1–3ms (on desktop platform) to detect (align) a set of 68 landmarks on a given face [12].

The indexes of the 68 coordinates can be visualized on the image below:

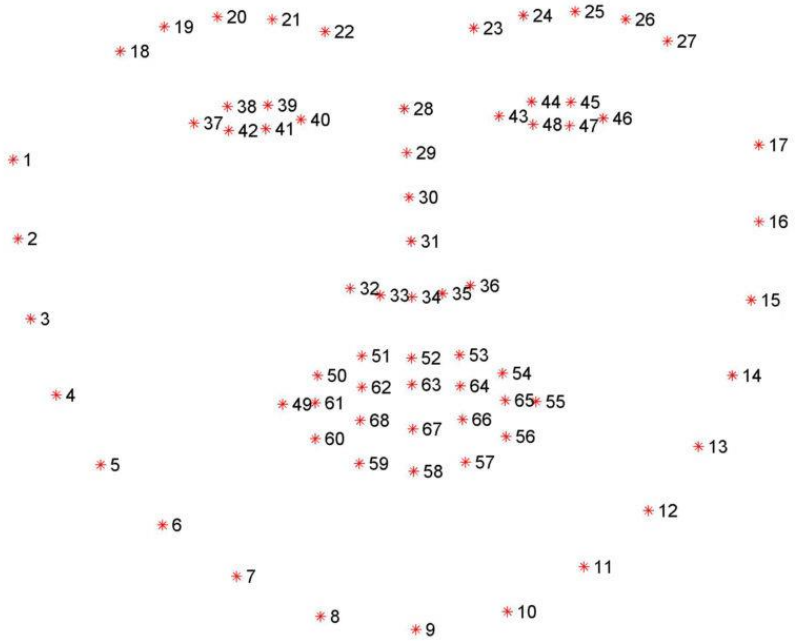

Figure 5 Facial Diagram Built with Dlib(Landmarks)

Using a coding program running in python there is a feature inside Dlib called training option which uses some parameters such as:

- "Tree Depth — Specifies the depth of the trees used in each cascade. This parameter represents the "capacity" of the model. An optimal value (in terms of accuracy) is 4, instead, a value of 3 is a good tradeoff between accuracy and model-size.
- Nu — Is the regularization parameter. It determines the ability of the model to generalize and learn patterns instead of fixed data. Value close to 1 will emphasize the learning of fixed data instead of patterns, thus raising the chances for over-fitting to occur. Instead, an optimal nu value of 0.1 will make the model to recognize patterns instead of fixed situations, totally eliminating the over-fitting problem. The amount of training samples can be a problem here, in fact with lower nu values the model needs a lot (thousands) of training samples in order to perform well.
- Cascade Depth — Is the number of cascades used to train the model. This parameter affects either the size or accuracy of a model. A good value is about 10-12, instead a value of 15 is a perfect balance of maximum accuracy and a reasonable model-size.
- Feature Pool Size — Denotes the number of pixels used to generate the features for the random trees at each cascade. Larger number of pixels will lead the algorithm to be more robust and accurate but to execute slower. A value of 400 achieves a great accuracy with a good runtime speed. Instead, if speed is not a problem, setting the parameter value to 800 (or even 1000)

will lead to superior precision. Interestingly, with a value between 100 and 150 is still possible to obtain a quite good accuracy but with an impressing runtime speed. This last value is particularly suitable for mobile and embedded devices applications.

- Num Test Splits — Is the number of split features sampled at each node. This parameter is responsible for selecting the best features at each cascade during the training process. The parameter affects the training speed and the model accuracy. The default value of the parameter is 20. This parameter can be very useful, for example, when we want to train a model with a good accuracy and keep its size small. This can be done by increasing the amount of num split test to 100 or even 300, in order to increase the model accuracy and not its size.
- Oversampling Amount — Specifies the number of randomly selected deformations applied to the training samples. Applying random deformations to the training images is a simple technique that effectively increase the size of the training dataset. Increasing the value of the parameter to 20 or even 40 is only required in the case of small datasets, also it will increase the training time considerably (so be careful). In the latest releases of the Dlib library, there is a new training parameter: the oversampling jittering amount that apply some translation deformation to the given bounding boxes in order to make the model more robust against eventually misplaced face regions.

By properly fine-tuning the training options is possible to customize the training process in such a way that satisfies the constraints of the system being developed.

Such constrains can be about executive speed, memory and storage consumption, and overall accuracy and robustness. That characterizes the platform being developed with (desktops, mobile devices, and embedded systems (In general, different platforms have different sets of quality requirements).

Moreover, by selecting only the relevant landmarks, it is possible to create specific models that localize a particular subset of landmarks, thus eliminating unnecessary points." [12].

Figure 6 Divisions of the Facial Implementation [12]

Figure 7 Application of the Facial Implementation

III. METHODOLOGY AND PROPOSED SOLUTIONS

The proposed solution involves analyzing the driver's breath alcohol level, the driver's attention level while driving, the location of the driver's focus and the rate of tiredness and drowsiness of the driver. Creating a safer environment solving the following issues: lack of sobriety of drivers, tiredness, drowsiness and focus of the driver.

i. Registration

The alcohol detection system (ADS) and the face features detection system (FDS) work with a capacity confirmation in two ways: One, the ADS, using Arduino, connected to a MQ-3 sensor that receives information about the driver's breath. From the received information the sensor provides the driver's blood alcohol level. Two, the FDS, which is controlled by a Raspberry Pi (microcomputer), is connect to a HD camera in order to captures the driver’s eyes features (the width and length), with the captured data the microprocessor evaluates the driver’s rates of attention, focus, tiredness, and the direction that the driver's eyes are looking.

ii. Arduino System for ADS

The Arduino system is responsible to analyze the level of alcohol in the driver’s breath. This system is reported in two main parts: schematic and components and source code.

a. Schematic Diagram and Components

The components of ADS are the Arduino Uno, MQ-3 sensor, LCD display (7-segment display) with its background color brightness regulator, fans, switchers, LEDs,

resistors, and wires. The Arduino Uno is the microprocessor which controls the components, sensor and devices of ADS. The MQ-3 sensor is the main component which evaluates the level of alcohol inside the air and depending of the concertation of it, the output resistance of A0 will variate. It increases if the density of alcohol in relation with the environment increases and decreases if the density of alcohol in relation with the environment decreases. The LCD display is the feedback which shows if the system is working, initialized, and the level of alcohol of the system (driver's breath), additionally the LCD display displays different messages depending on the MQ-3 sensor reading. The LEDs work together with the LCD display as a different feedback mechanism. Each LED color is related to a range of alcohol density in relation to the environment. The other components complement the system in order to make ADS work perfectly.

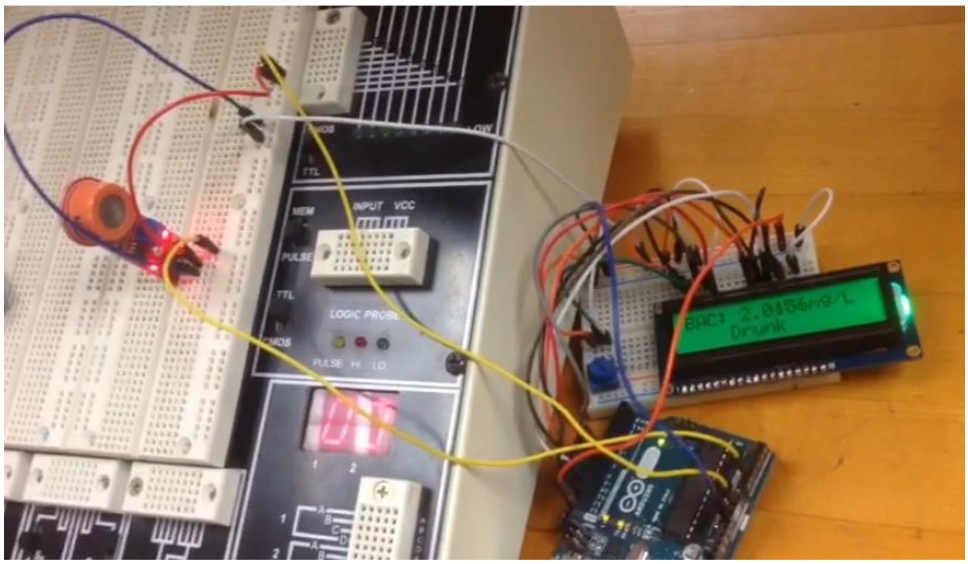

Figure 8 Arduino Project I (alcohol sensor)

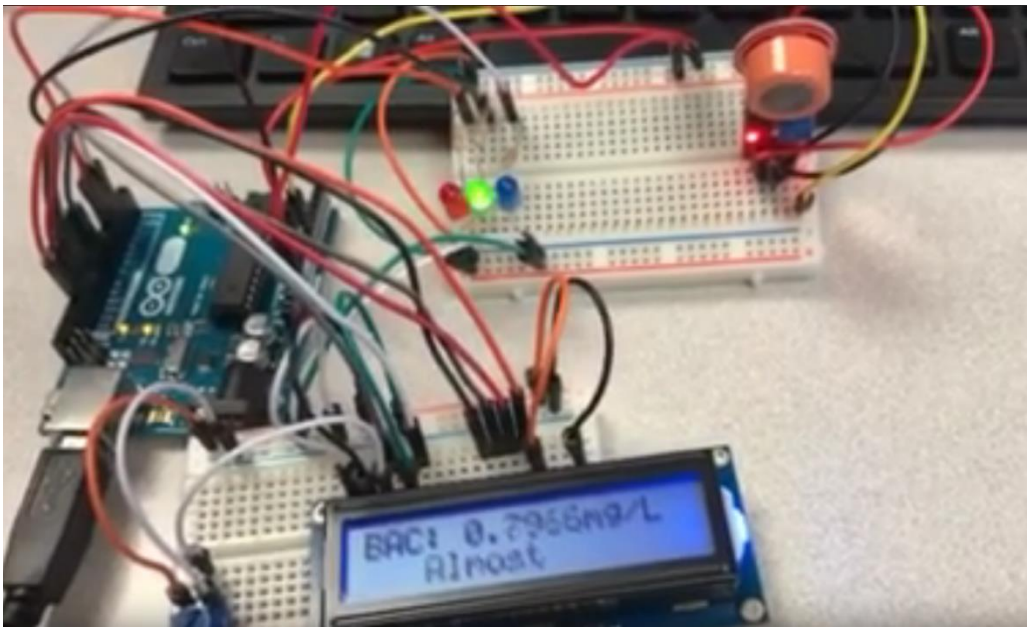

Figure 9 Arduino Project II (alcohol sensor)

b. Source Code

The microprocessor (Arduino Uno) is programmed in its own language (a simpler version of the computing language C++). The source code controls the MQ-3 sensor output resistance, buttons, fans, LEDs (green, blue, and red), and LCD display.

The first part consists of declaring common variables used inside the different loops of the source code, declaring libraries, registers and constants. The MQ-3 sensor was declared as A1 (analog register 1), the LCD display as the digital ports 7,8,9,10,11,12, the LEDs red, green, and blue to 4,3,2 respectively, the fan (a pack of fans connected in parallel) to the digital port 13, the button to the digital port 1. In addition, a constant called "sensor A1" is declared, and the button is set to be normally closed.

```
#include <LiquidCrystal.h>;          // LCD library
LiquidCrystal lcd(7,8,9,10,11,12); // LCD display registers
#define sensor A1                   // Defining the MQ3-sensor
int mq3_analogPin = A1;             // A0 pin From the MQ3-Sensor
byte red=4, green=3, blue=2;        // LEDs registers
byte fan = 13;                      // Fan register
int b0 = 1;                         // Button register
int b0 state = 0;                   // Setting the button as normaly closed
```

Figure 10 Common Variables, Registers and Libraries

The second part is the setup function of the source code, where the microprocessor virtually connects the inputs and outputs with the wires, in order to use those inputs and outputs inside functions and attribute values such as HIGH (current flowing throw the port with five or three point three volts across the port in relation to the virtual ground), or LOW, the virtual ground (the closest possible zero current flowing through the gate).

```
//Setting up Inputs and Outputs, and presetting functions
void setup()
{
  pinMode(red,OUTPUT);              // Red LED (pin 4)
  pinMode(blue,OUTPUT);             // Blue LED (pin 2)
  pinMode(green,OUTPUT);            // Green LED (pin 3)
  Serial.begin(9600);               // Open serial at 9600 bps
  pinMode(sensor,INPUT);            // MQ3- sensor (pin A1)
  pinMode(fan,OUTPUT);              // Fan (pin 13)
  pinMode(b0,INPUT);                // Button (pin 1)
  lcd.begin(16,2);                  // Setting LCD display 16 by 2 characters
  lcd.print("Alcohol Detector");    // Display message
  lcd.setCursor(0,1);               // Begin next Display at: column 0, row 1
  lcd.print(" Circuit Digest");     // Display message
  delay(2000);                      // Delay 2s
  lcd.clear();                      // Clear the display
}
```

Figure 11 Void Loop (setup)

The third and last part is the void loop where all the functions, calculations, ‘if and else’, and ‘for statement’ are implemented and declared to the system.

Before using the MQ-3 sensor output (analog input for the system) inside the void loop it was necessary to convert the data from the analog port to a data in milligrams per

liters. Since 5 volts is fed to the sensor and it is connected in a voltage divider circuit with a 10k ohms resistor, the voltage going into the A1 port of the Arduino is between 0 and 5 volts. The Arduino then divides it into 1023 parts, so the value read from the sensor is a voltage where every step is a change of 4.8 millivolts (5 volts dived by 1023).

The formula used to calculate the resistance corresponding to the voltage read, was the following: $R_s = \frac{(V_c - V_{out})}{V_{out}} \times R_L$.

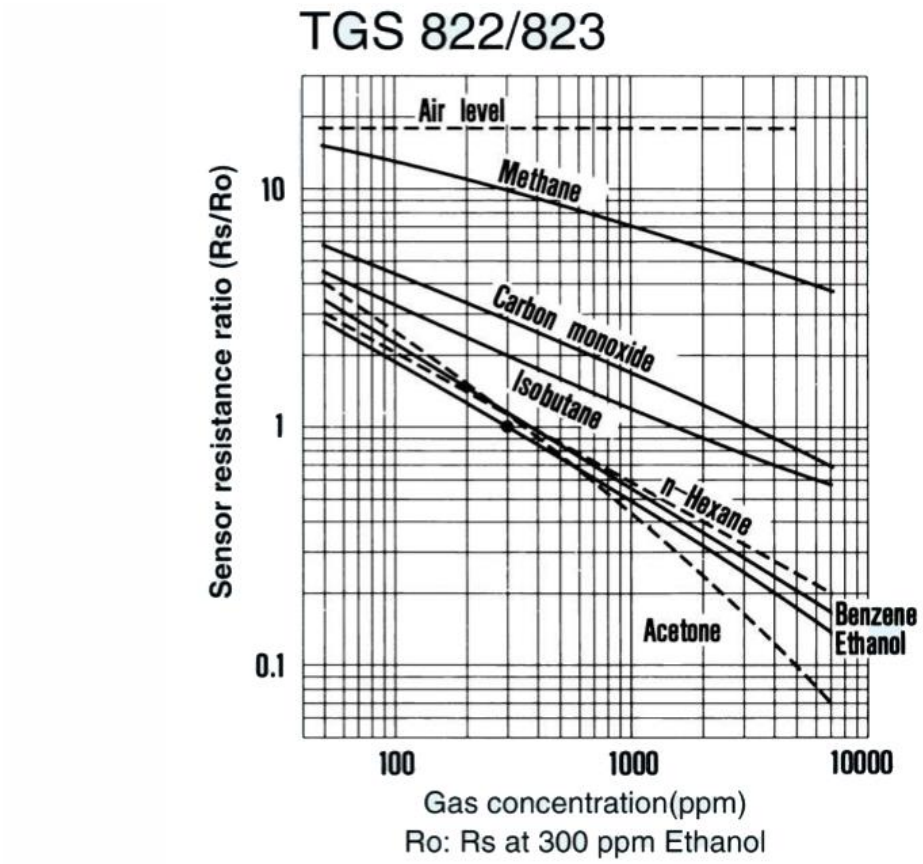

Figure 12 Sensor Resistance Ratio vs Gas Concentration

From the graph in the MQ-3 data sheet, it can also be seen that the resistance of the sensor in air is $R_s(air) = R_0 \times 19$.

If these two facts are combined, R0 can be expressed as a relation of $R_s(air)$ and the value of it can be deduced by reading the voltage of the sensor and using the voltage divider formula [14]. $\frac{R_s(air)}{19} = R_0$ in this case $R_0 = \frac{78k\Omega}{19} = 4105\Omega$.

From the calculated R_0 value, the resistor which will be placed in parallel with the MQ-3 sensor needs to be at least 4105Ω to reach the best result. In this study 5000Ω was used in parallel with the MQ-3 sensor in order to improve the data gathering from the analog output.

That said, the following step was to convert the data from the MQ-3 sensor to milligram per litter, in order to perform the conversion it was necessary to use $adcValue$ (variable related the MQ-3 sensor output, shown in figure 13) and R_0. The conversion was performed using the following formula: $^{mg}/_{l} = \frac{adcValue}{10} \times \frac{R_0}{1024} \times 0.67$

Additionally, the void loop begins with the declaration of button 0, and the variable calculation of $adcValue$, after that the $^{mg}/_{l}$ is calculated and declared to be displayed in the LCD display (figure 13).

```
void loop()
{
  b0_state = digitalRead(b0);          // Button connected to the constant
  float adcValue = 0;                  // New Variable starting at 0
  for(int i = 0; i < 10; i++){         // For statement which reads the A1 port
    adcValue += analogRead(sensor);    // MQ-3 Sensor Input
    delay(10);
  }

  float v = (adcValue/10)*(5.0/1024.0); // Combining the voltage step and the MQ-3 data
  float mgL = 0.67*v;                   // Function to convert the MQ-3 input to mg/L
  Serial.print("BAC");                  // Print "BAC" at the LCD
  Serial.print(mgL);                    // Print the variable mgL at the LCD
  Serial.print("mg/L");                 // Print "mg/L" at the LCD
  lcd.setCursor(0,0);                   // Display at column 0 and row 0
  lcd.print("BAC: ");                   // Print "BAC: " at the LCD
  lcd.print(mgL,4);                     // Print the variable mgL starting at column 4 at the LCD
  lcd.print("mg/L   ");                 // Print "mg/L   " at the LCD
  lcd.setCursor(0,1);                   // Display at column 0 and row 1
```

Figure 13 Void Loop

Following the code, there are four 'if and else' statements, they use the converted $^{mg}/_{l}$ as the main variable. The first one triggers when the variable value reaches the maximum that the systems accepts (1.5 $^{mg}/_{l}$), implying that the driver's rate of alcohol is above acceptable. After being trigged the following happen: "Incapable to Drive Stop the Car" is displayed in the LCD screen and red LED lights up (figure 14).

```
if(b0 == HIGH){                          // Swither on (system initialized)
 digitalWrite(fan,HIGH);                 // Fan initialized

if(mgL > 1.5){                           // If the Alochol level Higher than " "
  lcd.print("Incapable to drive");       // Display "Incapable to drive"
  Serial.println("Stop the Car");        // Next line Display "Stop the Car"
  digitalWrite(red,HIGH);                // Red LED on
  digitalWrite(blue,LOW);                // Blue LED off
  digitalWrite(green,LOW);               // Green LED off
}
```

Figure 14 Void Loop

The second statement triggers in between low level of alcohol (1.2 $^{mg}/_{l}$) and above the acceptable (1.5 $^{mg}/_{l}$). The statement is similar to the previous one, the changes are: the LCD display displays "Alcohol Level Above The Normal Level" and lights up the green LED (figure 15).

```
else if(mgL > 1.2 and mgL < 1.5){        // If the Alochol level if between " " and " "
 lcd.print("Alcohol Level Above");       // Display "Alcohol Level Above"
 Serial.println("The Normal Level");     // Next line Display "The Normal Level"
 digitalWrite(red,LOW);                  // Red LED off
 digitalWrite(blue,LOW);                 // Blue LED off
 digitalWrite(green,HIGH);               // Green LED on
}
```

Figure 15 Void Loop

The third statement it is the "default" statement, when the driver enters the car sober. The system will display "Capable to Drive" and it will light up the blue LED (figure 16).

```
else if(mgL < 1.2){                    // If the Alochol level Lower than " "
 lcd.print("Capable to Drive");        // Display "Capable to Drive"
 Serial.println("Capable to Drive");   // Next line Display
 digitalWrite(red,LOW);                // Red LED off
 digitalWrite(blue,HIGH);              // Blue LED on
 digitalWrite(green,LOW);              // Green LED off
}
```

Figure 16 Void Loop

The fourth and last statement, is the "error" checking statement, where, if the system is not working as planned, all LEDs will light up (figure 17).

```
else {                                 // If any error occurs
  digitalWrite(red,HIGH);              // Red LED on
  digitalWrite(blue,HIGH);             // Blue LED on
  digitalWrite(green,HIGH);            // Green LED on

}
```

Figure 17 Void Loop

iii. Raspberry Pi System (FDS)

The FDS is built in a Raspberry Pi microcomputer. This part is divided in two main parts: Schematic diagram and Components and Source Code.

a. Schematic Diagram and Components

Different from the Arduino project, the Raspberry Pi project uses three components, a high definition camera, a SD card, and a buzzer. In addition, during the testing process the microcomputer was connected to a laptop in order to have a real time feedback. Furthermore, the project was programed in python inside a virtual machine running Ubuntu 16.04 using OpenCV 4, pip, boost, boost.python, CMake, X11/Quartx, Imutils, paysound, and pyobjc.

FDS triggers when the driver begins to show sign of tiredness, drowsiness, or lack of attention while driving. After the system is trigged, the code runs calculations until it decides the next step. If it is decided that the driver stayed in one of the states cited for longer than accepted, the buzzer will play a loud noise warning the driver with the objective of bringing his attention back. However, if the driver keeps the same symptoms, the device will keep warning the driver until the driver reaches the fourth warning, after that the buzzer will not stop making noise until the car is fully stopped and the system is restarted.

The FDS has one input and one output. The input of the system is the camera. It was used the Logitech C920s pro HD webcam (figure 18). It was important to have a good quality camera because the system analyzes the driver's face in real time, to be more specific, it analyzes the shape and lines of the driver's eyes, and depending of the quality of the image input the microcomputer may lose the track of the eye lines.

Figure 18 Logitech Camera

The output of the system is a buzzer connected to the Raspberry Pi. The buzzer works together with the camera. From the data analyzed, provided by camera, the microcomputer evaluates if the buzzer will be turned on and create a loud noise alerting the driver or not. For example, if the driver sleeps while driving, the buzzer will be turned on and try to wake the driver up, or If the driver is lacking attention while driver, looking around too much, texting, reading, etc. the buzzer will turned on and warn the driver to restate a “safe” driving behavior.

Figure 19 TrafficHAT

b. Source Code

In order to integrate the FDS, and to have the necessary tool to convert the input from the camera in valuable and calculable data, it was necessary to import several packages (figure 20).

```
# import the necessary packages
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import playsound
import argparse
import imutils
import time
import dlib
import cv2
```

Figure 20 Packages

After importing the packages, the sound_alarm (buzzer output) and the eye_aspect_ratio were defined. The sound_alarm was defined as a playsound.playsound(path) rooting the sound that will be played by the buzzer. The eye_aspect_ratio calculates the vertical and horizontal distances between the edges of the eye: from the left side to the right side and from the top to the bottom, using the variable A, B, C and ear. A and B are used to calculate the vertical distance between the eyelids (top and bottom), and C is used to calculate the distance between the "sides" of the eye. Lastly, ear is the eye aspect ratio created by the addition between the A and B, divided by two times C.

```
def sound_alarm(path):
        playsound.playsound(path)

def eye_aspect_ratio(eye):
        A = dist.euclidean(eye[1], eye[5])
        B = dist.euclidean(eye[2], eye[4])
        C = dist.euclidean(eye[0], eye[3])
        ear = (A + B) / (2.0 * C)

        return ear
```

Figure 21 Definitions

After that, constants were created to be used in further calculations and functions. the constants were EYE_AR_THRESH, EYE_AR_CONSEC_FRAMES,

COUNTER_FRAME, ALARM_ON, COUNTER, and TOTAL. The EYE_AR_THRESH is used as thresholder for the "ear". Furthermore, when the "ear" reaches a value equal or smaller to this constant a counter is triggered (COUNTER_FRAME) that is related to the EYE_AR_CONSEC_FRAMES, which is the number of consecutive frames necessary in order to turned the ALARM_ON into "TRUE" (buzzer plays the noise).

```
EYE_AR_THRESH = 0.25
EYE_AR_CONSEC_FRAMES = 20

COUNTER_FRAME = 0
ALARM_ON = False

COUNTER = 0
TOTAL = 0
```

Figure 22 Constants Values

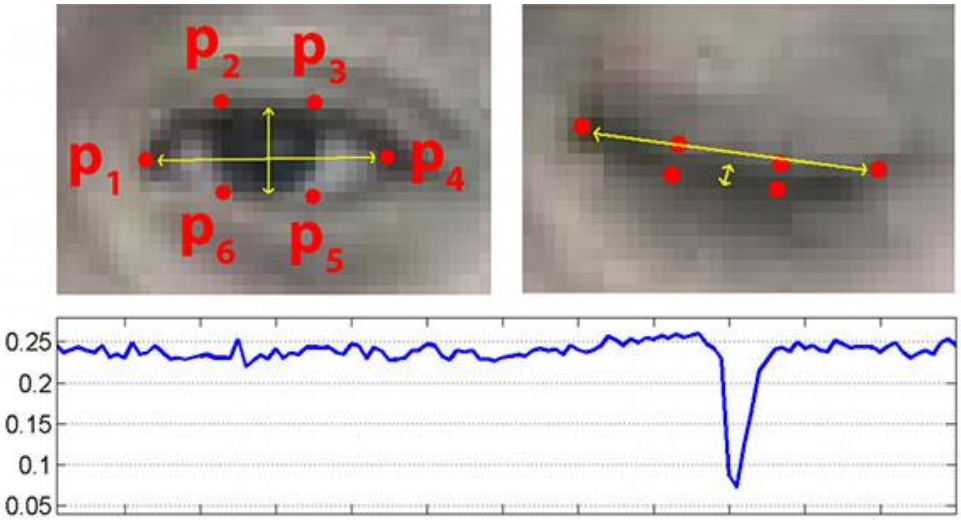

Figure 23 Calculation of Aspect Ration

Moving to the loops section, first it is necessary to convert the frames from the camera to frames readable inside the code. Using the following code lines (figure 24), the program grabs the frames from the threaded video file stream, resizes it, and converts it to grayscale channels (frame in gray). Further in the loop, the "rects" detects faces in the grayscale frame (from the imutils).

```
while True:
        frame = vs.read()
        frame = imutils.resize(frame, width=450)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        rects = detector(gray, 0)
```

Figure 24 While Loop

Continuing inside the loops section, the program is capable to detect more than one face at the time, however, in this project it is assumed that there is only one person sitting on the driving seat. The "shape" is attributed to apply the dlib's facial landmark detector and convert the result into NumPy array. Furthermore, with the NumPy conversion array slicing the program is able extract the x-y coordinates of the left and right eye. Lastly, with the coordinates extracted the program will calculate the aspect ratio (ear) of the coordinate's obtained from the extraction and use the result in the 'if else' statement.

As said previously, it was used a computer as a feedback of the system. So, following the code, the next step is to add a mask on the top of the image sent to the computer and draw lines around the eyes of the driver, in order to have a feedback from the system while testing it. For this part, the cv2.drowCountours function is used.

```
for rect in rects:

        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)

        leftEye = shape[lStart:lEnd]
        rightEye = shape[rStart:rEnd]
        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)

        ear = (leftEAR + rightEAR) / 2.0

        leftEyeHull = cv2.convexHull(leftEye)
        rightEyeHull = cv2.convexHull(rightEye)
        cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
        cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

Figure 25 Calculations, variables Attributions, and Mask

Firstly, in the source code, it begins with the ‘if’ statements which controls the warning layer over the feedback screen (computer) and when to play the warning. Additionally, the ‘if’ statement checks if the eye aspect ratio is in the limit attributed before in the source code (EYE_AR_THRESH). If it is true, the constant COUNTER_FRAME will be incremented while the driver’s eyes opening is smaller than the accepted by the EYE_AR_THRESH.

Secondly, the next ‘if’ statement works with EYE_AR_CONSEC_FRAMES. If the driver has kept his eyes closed for enough time (EYE_AR_CONSEC_FRAMES), the system will play the warning sound in order to wake the driver, and catch his attention back to driving, and increment the total of warning already given (TOTAL).

Thirdly, in the system, the driver is given four chances (four warning) until the system asks the driver to park the car. In order to do that, a series of ‘if’ statements displaying different messages in the feedback screen (computer screen) is used. The default state is a screen displaying the EAR number and the number of warnings. After every warning and alarm, the warning number will be increased, and a unit warning will be displayed on the screen with the alarm sound together with it until the driver opens the eyes. This will happen until the driver stops sleeping or if he reaches the max amount of “lack of attention” while driving as proposed by the system (four times, in this test).

```
if ear < EYE_AR_THRESH:
        COUNTER_FRAME += 1

        if COUNTER_FRAME >= EYE_AR_CONSEC_FRAMES:

                if not ALARM_ON:
                        ALARM_ON = True
                        TOTAL += 1

                        if args["alarm"] != "":
                                t = Thread(target=sound_alarm,
                                        args=(args["alarm"],))
                                t.deamon = True
                                t.start()
                if TOTAL == 1:
                        cv2.putText(frame, "ALERT 1", (150,100),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)

                if TOTAL == 2:
                        cv2.putText(frame, "ALERT 2", (150,100),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)

                if TOTAL == 3:

                        cv2.putText(frame, "LAST WARNING!", (150,100),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
                if TOTAL == 4:

                        cv2.putText(frame, "4th Warning", (150,100),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
else:
        COUNTER_FRAME = 0
        COUNTER = 0
        ALARM_ON = False
```

Figure 26 Warning Loop

Lastly, when the driver reaches the last warning the feedback will display constantly “Last” in the warning counter, and “PARK THE CAR” in the middle of the screen. Additionally, the system will still trigger the alarm if the driver repeats any of the behavior cited previously.

```
if TOTAL >= 4:
        cv2.putText(frame, "Warning: LAST", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        cv2.putText(frame, "PARK THE CAR", (150,200),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
else:
        cv2.putText(frame, "Warning: {}".format(TOTAL), (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```

Figure 27 Warning Loop

The follow Images shows the FDS simulation:

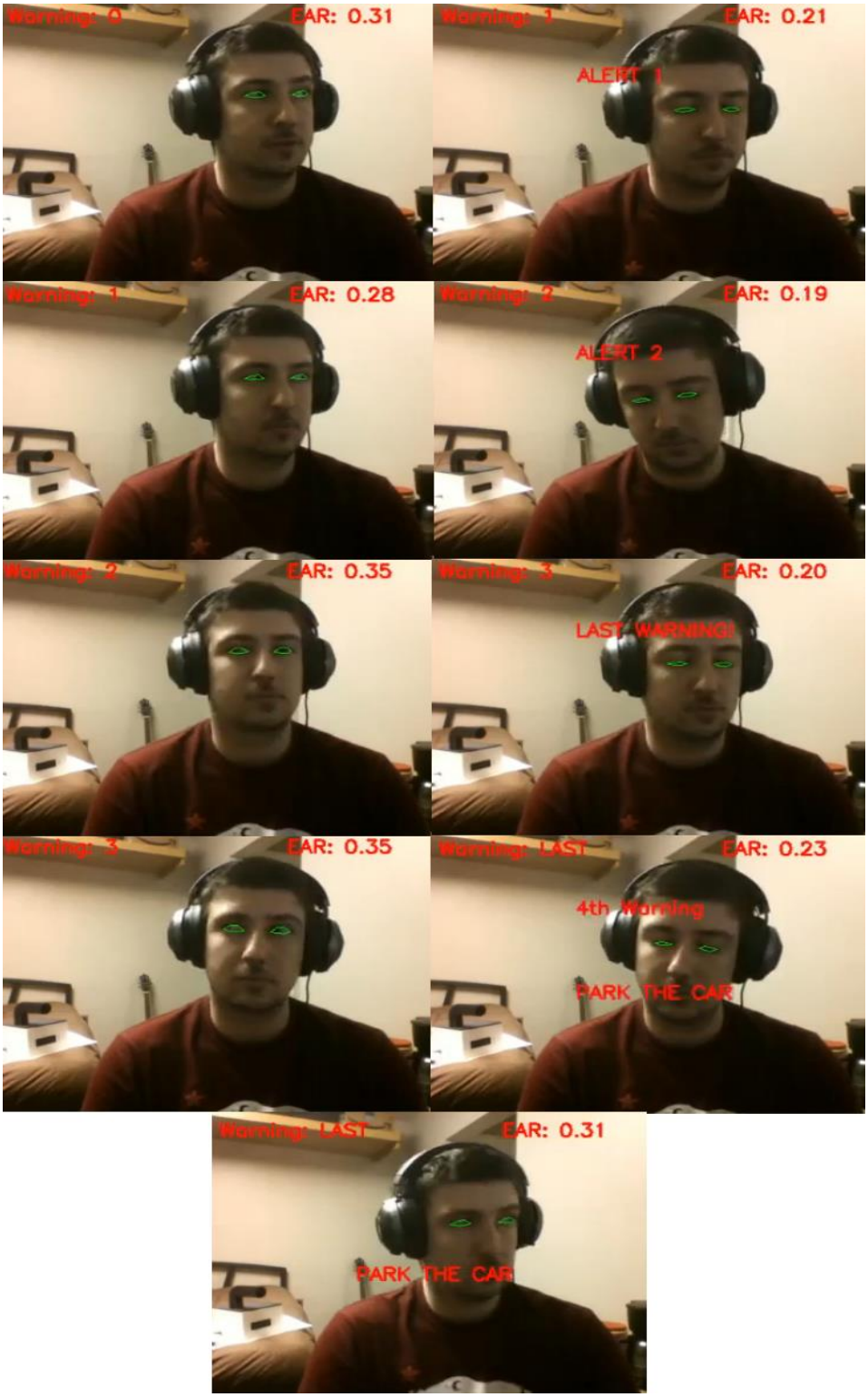

Figure 28 Project Test

iv. Prototype and 3-D Printing Design

After the creation of ADS and FDS it was necessary to develop a prototype to place both systems. The prototypes were designed to fit both Arduino and Raspberry Pi, with their components and power sources.

The prototypes were designed in AutoCAD and each prototype was 3-D printed inside the YSU laboratory. There were five version, the first and second version are shown in figure 29 and the last version is shown in figure 30. The last version has holes around the prototype in order to create a better air flow (the air flows constantly through the fans installed inside the pipe in the top of the prototype). In addition, on one of the prototype walls, the LCD monitor and the LED were connected, on the bottom surface, the Raspberry Pi, Arduino and the protoboard were attached to proper places, and lastly, the MQ-3 sensor was attached to a structure connected to the top of the prototype, underneath the fans.

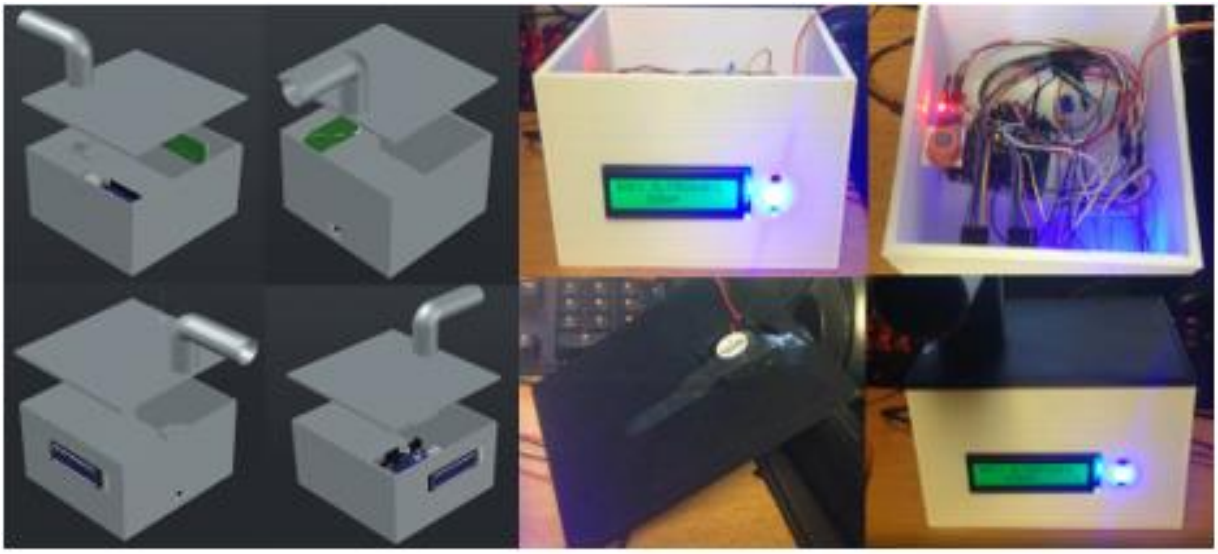

Figure 29 3-D design I

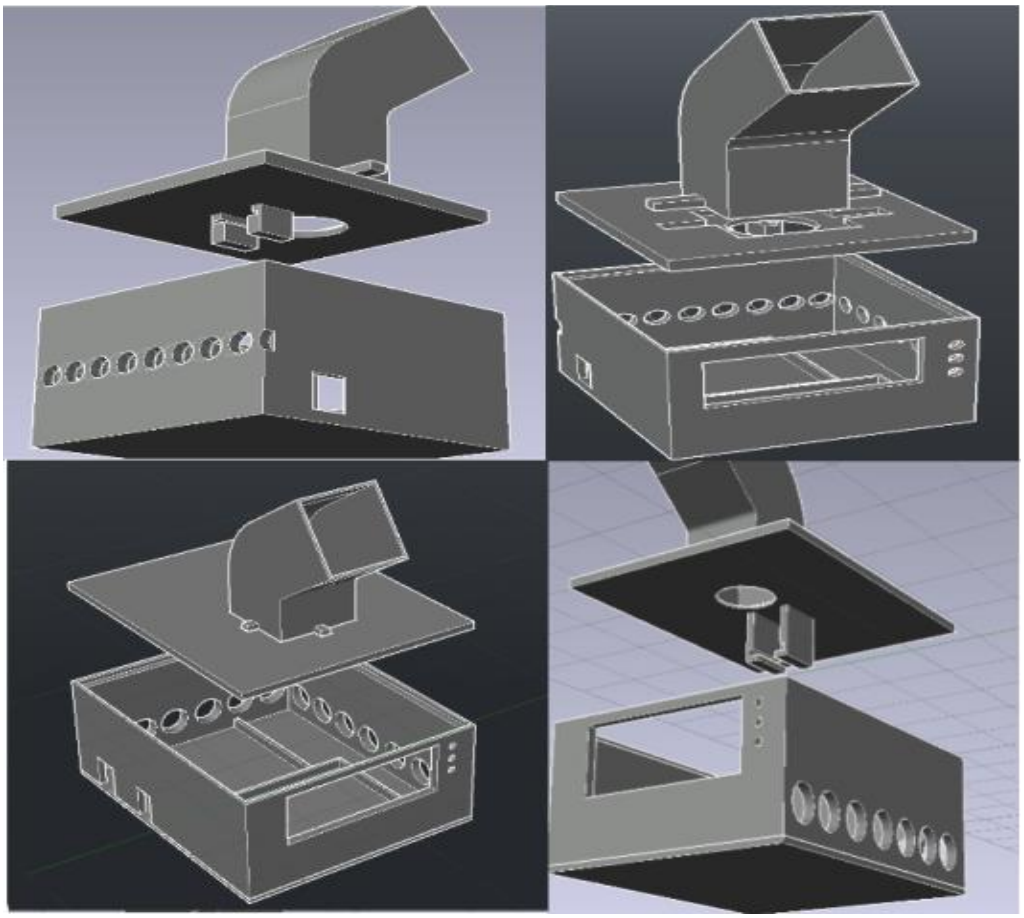

Figure 30 3-D design II

IV. CONCLUSION AND FUTURE IMPLEMENTATION

In conclusion, the FDS worked as well as expected, the facial recognition was on point and every test was performed without problems. However, the ADS had issues with the MQ-3 sensor, the sensor was not accurate as advertised, even using several sensors from different brands, there were not any consistence with the measurements between the sensors. During the testing period it was used twenty to twenty five different sensor and they all showed small difference in the measurement values, it was necessary to constantly adjust the resistor R_0 and the calculation conversion for each sensor, in order to generate symmetry between the MQ-3 sensors. In future implementation, it will be necessary to develop a better sensor for the ADS for a safer data gathering.

V. REFERENCES

[1] Wikipedia, January 2020, https://en.m.wikipedia.org/wiki/Vehicle_insurance_in_the_United_States

[2] Autotrader, January 2020, https://www.autotrader.com/car-shopping/7-criteria-car-buying-281474979927978

[3] Blincoe, L.J.; Miller, T.R.; Zaloshnja, E. and Lawrence, B.A. 2015. The economic and societal impact of motor vehicle crashes, 2010 (revised). Report no. DOT HS-812-013. Washington, DC: National Highway Traffic Safety Administration

[4] The Highway Loss Data Institute, January 2020, https://www.iihs.org/topics/fatality-statistics/detail/yearly-snapshot#fn1

[5] National Highway Traffic Safety Administration. Traffic Safety Facts 2016 data: alcohol-impaired driving. U.S. Department of Transportation, Washington, DC; 2017, https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812450External Accessed 16 April 2018.

[6] Federal Bureau of Investigation (FBI). Department of Justice (US). Crime in the United States 2016: Uniform Crime Reports. Washington (DC): FBI; 2017, https://ucr.fbi.gov/crime-in-the-u.s/2016/crime-in-the-u.s.-2016/tables/table-18

[7] National Highway Traffic Safety Administration (Research Note), January 2020, https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812700

[8] National Highway Traffic Safety Administration (Crash Stats), January 2020, https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812446

[9] Wikipedia, January 2020, https://en.wikipedia.org/wiki/Motor_vehicle_fatality_rate_in_U.S._by_year

[10] Wikipedia, January 2020, https://en.wikipedia.org/wiki/Scikit-image

[11] Sparkfun Start Something, January 2020, https://learn.sparkfun.com/tutorials/what-is-an-arduino/all

[12] Dlib C++ Library, January 2020, http://dlib.net/

[13] Anzalone, L., Training alternative Dlib Shape Predictor models using Python, January 2020, https://medium.com/datadriveninvestor/training-alternative-dlib-shape-predictor-models-using-python-d1d8f8bd9f5c

[14] Jenslabs Ketosense an Arduino Based Ketosis Detector, January 2020, https://jenslabs.com/2013/06/06/ketosense-an-arduino-based-ketosis-detector/

[15] Jenslabs Calibrating a Figaro TGS822 Sensor by Drawing, January 2020,https://jenslabs.com/2013/03/21/calibrating-a-figaro-tgs822-sensor-by-drawing/

[16] MQ-3 Sensor datasheet accessed 09 June 2019, https://www.sparkfun.com/datasheets/Sensors/MQ-3.pdf