Spring 2020

Vehicle Operator Attention Monitor

Matthew Krispinsky
mak191@zips.uakron.edu

Matt Marsek
mdm176@zips.uakron.edu

Matthew Mayfield
mam467@zips.uakron.edu

Brian Call
brc59@zips.uakron.edu

Vehicle Operator Attention Monitor

Senior Project Final Design Report

DT09

Brian Call

Matthew Krispinsky

Matt Marsek

Matthew Mayfield

Faculty Advisor: Osama Al Khateeb

April 24, 2020

Table of Contents

List of Figures

List of Tables

Abstract

Motor vehicle operators' attention levels can be monitored to improve driver safety. By recording and analyzing the drivers eye gaze, hand position, vehicle speed and engine rpm the driver's attention can be determined. A Raspberry Pi will be the main processing unit. Data will be pulled and analyzed from the OBD-II port on vehicle speed and engine rpm. The system will be powered from a 12V, 4A pin on the OBD-II port connected to the car battery. A webcam will be used to track the pupil location and determine when the driver is looking at the road. A battery powered microprocessor with sensors on the steering wheel will record hand position and wirelessly transmit to the main processing unit using bluetooth. The system will alert the driver upon determined infractions; not looking at the road for 2 seconds, hands not at the optimal positions for more than 1 second, or dangerous speeds and acceleration. The system will also track and store the number of infractions over a period of time for additional analysis. Vehicle operators attention levels can be monitored and improved using these methods. (BC, MK, MM, MDM)

Key Features:

- Encourages safe driving through the use of audio warnings when a driver uses bad driving practices (i.e. hands off the wheel, eyes not focused on the road, etc.)
- Utilizes eye tracking technology to detect drivers general gaze direction, how attentive to the road they are, and to detect if they are drowsy or asleep.
- Communicates with vehicle onboard computer to retrieve vehicle speed and RPM.
- Uses wireless communication to have independent subsystems mounted on the steering wheel without interfering with vehicle operation.

1. Problem Statement

1.1. Need

There is a need for more attentive while driving on the roads. Driving while distracted is a major cause of car accidents. Table 1, provided by the AAA Foundation for Traffic Safety, shows rates of involvement of some distractions in car accidents. Car accidents have many negative repercussions. Such as damage to cars and other property, financial losses, traffic jams, injuries, and potential loss of life. Many distractions and inattentive driving tendencies are not currently monitored. For instance taking eyes off the road, taking hands off the wheel, and driving with potentially distracting noise levels. (BC, MK, MM, MDM)

Table 1. Crash causes from certain distractions while driving

Stutts, J.C., Reinfurt, D.W., Staplin, L., & Rodgmen, E. (2001). The role of driver distraction in traffic crashes.

Driver Distraction	Percent of Crashes Involving:			
	2+ Vehicles	Vehicle Going Straight	Front Impact	Serious or Fatal Driver Injury
Outside person, object, event	66.1 [1] (4.0) [2]	42.7 (4.7)	69.7 (2.7)	5.7 (1.8)
Adjusting radio/cassette/CD	37.8 (21.3)	46.5 (24.7)	87.9 (7.5)	1.9 (0.5)
Other occupant	55.9 (7.5)	68.2 (4.6)	59.1 (2.1)	8.3 (2.9)
Moving object in vehicle	17.0 (8.8)	43.9 (9.1)	91.1 (5.0)	11.3 (11.2)
Other device/object	48.8 (11.3)	61.0 (12.1)	81.6 (10.5)	13.7 (7.7)
Vehicle/climate controls	59.6 (11.0)	70.1 (10.9)	85.5 (6.7)	3.4 (1.1)
Eating, drinking	53.4 (8.4)	61.3 (8.8)	79.7 (9.3)	10.3 (3.4)
Using/dialing cell phone	82.9 (11.6)	68.5 (11.4)	68.8 (16.4)	8.4 (4.6)
Smoking related	15.6 (4.9)	59.3 (7.6)	42.7 (10.0)	7.8 (2.0)
Other distraction	53.8 (5.8)	61.8 (3.5)	73.7 (4.6)	12.7 (5.5)
Unknown distraction	85.3 (2.7)	69.0 (9.9)	83.3 (10.4)	6.5 (2.8)
OVERALL	57.0	55.3	74.6	7.9

[1] Percent of crashes
[2] Standard error

1.2. Objective

A system that will monitor driver attentiveness while driving. The system will issue feedback when it detects the driver is not being attentive. It will also record data over time on how attentive the driver is. The system will be constructed of a series of sensors to monitor attentiveness and potential distractions along with a computer and a speaker. The sensors will provide input to a computer which will then store the data and trigger auditory feedback for the driver when necessary. (BC, MK, MM, MDM)

1.3. Background

The project will be a system to be installed in cars for the purpose of alerting the driver when he or she is being attentive and to record safe driving habits. This will be accomplished by using a number of sensors to monitor when the driver is not being attentive or is distracted. The sensors will feed the information to a processor. Using algorithms, the processor will determine if the driver is or is not being attentive. If the driver is not being attentive, the processor will activate a speaker which will beep to alert and remind the driver that he or she is distracted and needs to refocus on driving. The processor will also record and store instances when the driver was distracted or not focusing on the road. [MDM]

The sensors the system will use will be an eye tracker, a steering wheel hand sensor, and a deciblemeter. The system will also have a device that reads the OBD-II port. These systems will input into a raspberry pi and there will be a speaker at the output. [MDM]

In terms of the OBD-II port, there are a wide variety of signals and parameter IDs that are available. The current plan is to power the system via the port and use several data points that are of interest for data collection and possible decision making. For the purposes of this

system, a few that stand out are the speed and relative accelerator pedal position. In terms of speed (which comes in as unit of km/h or mph), even if the system will not have the knowledge of the current road's speed limit, it still may be used as a theoretical cap for "unreasonable" speeds, such as speeds 100mph or more, thus notifying the driver that they are not being attentive to their own speed. Similarly for relative accelerator position (which comes in as a percentage), the system may once again "cap" at a certain level that would be reasonable and again notify the driver if they are accelerating recklessly. [MAK]

An eye tracker will be included for the purpose of determining if the driver is looking at the road or not. Eye tracking technology works by using sensors to capture and track the light rays reflected off the corneas of a person's eyes. These sensors then feed data to a processing unit where the data is processed according to algorithms. Other devices can then use that data accordingly. There are two ways to incorporate sensors in eye tracking technologies. One way is to attach a lens to the cornea with sensors and transducers in it that can track the light rays as they enter the eye. The other way is by using a monitor of some kind that shines infrared light into the eyes and tracks the reflections. [2] The second method is the method that will be used for this project as it is much less invasive than the first way. The system is meant to be not so invasive as that could potentially distract the driver which is counterproductive to what the goal of the project. Currently, the company EyeTechDS is developing an eye tracking technology to alert drivers who are not looking at the road are currently under development. One such design tracks when the eyes look away from the road with a dashboard mounted, infrared camera. The device produces an audible beep when the driver has looked away after a short duration to alert the driver that they are not looking

at the road. [1] An eye tracking alert system similar to this one is intended to be used as an input into the alert system being developed. [MDM]

A hand pressure sensor will be included for the purpose of determining whether the diver's hands are on the steering wheel or not. Driving with one or both hands off the steering wheel is not safe and considered non-attentive driving. Further, an analysis of optimal hand positioning was done by the University of Galati, Faculty of Engineering, Romania. Their study focused on the hand coordinates on the steering wheel for optimal and safe driving. Using thermal imaging to analyze palm temperature and contact area, it was determined that the optimal comfortable and safe hand positioning on the steering wheel is in the 3 o'clock and 9 o'clock positions. [3] The system will also record how often the driver has his or her hands in these positions. This data will be recorded for insurance purposes. [MDM]

A combination of human face recognition and eye tracking systems have been implementing using a Raspberry Pi board. The software and device made was designed for eye gaze tracking and drowsiness detection. The system only checks the length of time that eyes are not open. The software was implemented first on a personal computer with Intel Core i5 processor and then on a Raspberry pi for speed comparisons. Also, Class 4 and Class 10 SD cards were tested for effects on detection time. The algorithm used was a multistep process designed in order to minimize the detection times. Using Haar Cascade, a machine learning object detection algorithm, the software first identifies the face of the subject. To save processing time the software then eliminates the portions of the face eyes will not be present. It then finds the center of the eye to calculate eye gaze. The algorithm also checks

drowsiness by seeing if a driver's eyes are closed for too long. Real time image processing requires powerful hardware to implement quickly. The detection time that was measured between the personal computer and Raspberry Pi were vastly different. The PC had a detection time of 28-32 milliseconds, the Raspberry Pi with Class 4 SD had 1200-1500 milliseconds and the Raspberry Pi with Class 10 SD had 850-990 millisecond response time. The hardware for this real time detection use must process quickly enough to offer an intervention before an accident would occur. [BC]

A Raspberry Pi 4 Model B has been used a microphone to classify water sources in a bathroom. Water flow is classified based on sound analysis and can accurately identify three different water sources from a single microphone. By analyzing the frequency domain of the measured sound from the microphone, different sounds can be identified. A simple USB microphone was used. The sound was sampled at 44. 1kHz. Looking at the spectrum concentration and amount of energy within certain frequency ranges a decision-making algorithm is able to identify which water sources are on with between 97% and 100% accuracy. This process could also be used to identify sounds in cars that are considered distracting versus normal road or construction noise that is unavoidable. The Raspberry Pi has enough processing power to quickly perform these calculations and provide real time feedback. The programming for this was done in Python which means that by utilizing other languages it could be even faster which would be important with multiple sensors and eye tracking being implemented using the Raspberry Pi. [BC]

Hand positioning whilst driving can be the decisive factor when a decision needs to be made quickly. In the proposed setup, the system will constantly check whether the

drivers' hands are both located on the wheel at the appropriate position. As previously described, the 3 o'clock and 9 o'clock positions were deemed to be "optimal". Thus, the goal is to not only record how long the driver's hands are in the optimal position throughout the drive, but to also keep the driver aware of their hands in real time with an indicator. This indicator will be tied to the encompassing alert system to distinguish whether the driver is starting to get distracted or not. A standalone vibration system will also be implemented into the wheel so that the driver can tell that the area in which they are not being a responsible driver resides in their hand positioning. [MAK]

Drowsiness is a leading factor in serious traffic accidents. According to the National Highway Traffic Safety Administration, there are about 56,000 crashes caused by drowsy drivers every year in the US, which results in about 1,550 fatalities and 40,000 nonfatal injuries annually. [6] This statistic furthers the idea that if a drivers' hands were to start slipping or if one hand was being used to try to support their head as they start to fall asleep, the alert system accompanied with the vibration system would constantly prevent said driver from falling asleep behind the wheel. In a similar project scenario, a group had designed a seat which would be excited via a motor mounted inside the seat to vibrate and alert the driver. This systems' wheel will be behaving similarly, such that it will be incorporating a module that transmits a signal to activate a motor or similar device to vibrator inside the steering wheel using wireless communication. [7] [MAK]

A patent designed by Jason Lisseman and Tom Mogg "provides a method and mechanism to evaluate and measure the driver's well being. In addition, it would be desirable to provide such a mechanism as a cost effective device which can be integrated into

vehicle designs. ” (US 8725230B2, May 13, 2014). That design for a steering wheel sensor is somewhat relatable to this system, albeit for the purpose of measuring biological parameters of the driver. In this case, here the metrics would be for pressure, would be enough for the system to recognize the user has their hands on the wheel so that their level of responsibility may be assessed. [8] [MAK]

Capacitive sensing would be a cheap and effective way to determine when the drivers hands are on the steering wheel. Capacitive sensors can be sturdy and flexible to fit to the shape of the steering wheel while not disrupting the operation of the vehicle. Capacitive sensors are low powered and will provide the quick accurate response needed. Capacitive touch sensors are widely available for purchase or can be built for custom applications. [9] [BC]

1.4. Marketing Requirements

- System needs to increase drivers’ self-awareness of performance
- System needs to not be intrusive nor interfere with vehicle operation
- System needs to be easy to install while also having a simple interface
- System needs to aid insurance companies in assessing driver responsibility
- System needs to have a reasonable price point

(BC, MK, MM, MDM)

2. Engineering Analysis

2.1. Circuits

The main controller and peripherals will be powered via the OBD-II port. All OBD-II standards include a 12V pin capable of supplying a minimum of 4A. The main microcontroller of the project will be a Raspberry Pi which requires 5V and 3A to power. In order to power the Raspberry Pi from the OBD-II port, a voltage regulator is needed to reduce the voltage from the OBD-II port from 12 volts to 5 volts. The overall equation for a voltage regulator is as follows.

$$V\,out = A * V\,in$$

Where *Vout* is the output voltage, *A* is the voltage gain of the system, and *Vin* is the input voltage. The voltage gain is calculated below.

$$A = V\,out/V\,in$$

$$A = 5/12$$

$$A = 0.417$$

The voltage regulator will need a gain of approximately 0.417 volts/volt to achieve the proper voltage level to power the Raspberry Pi. It will also have to be able to supply a current of 3A. Additionally, a regulator of high efficiency is desirable since the power is drawn from the car's battery. A switched regulator will be used in the design due to their high efficiency. [MDM]

2.2. Electronics

The hand sensor subsystem, shown in Figure 3, will be powered by a rechargeable battery. This subsystem includes a microcontroller, a bluetooth module, and the force sensitive resistor sensors. The operating range of the microcontroller will be 2.0V to 3.6V. A low energy bluetooth module operates at a voltage between 1.8V and 3.6V. The force sensitive resistors will be directly to the microcontroller. A rechargeable 3.6V battery will be used to power these devices. The force sensitive sensors operate by changing the resistance when outside pressure is applied to any part of the resistor strip. The resistance of the device with no pressure applied is upwards of 100kΩ and down to less than 1Ω with higher pressure. The force sensitive resistor will be put in a voltage divider circuit to get an output described by the equation:

$$Vout = \frac{V+}{1+RFSR/RM}$$

Where V+ is the rail voltage, RFSR is the variable resistor, and RM is the second resistor in the divider. The circuit will implement a op-amp in a voltage buffer configuration after the voltage divider with the output voltage going to the controller. [BC]

2.3. Signal Processing

The eye tracking subsystem will be taking a constant infrared image of the drivers' eyes throughout the entire length of the drive as soon as the ignition is started. Each pass, the subsystem will be detecting for three different states; if the drivers' eyes are on the road, maintaining adequate responsiveness, if the drivers' eyes are off the road, whether it be down inside the cabin or off to the sides for too long, and if the drivers' eyes are completely closed, due to drowsiness. Every scan, the IR camera will be polling

this data to the Raspberry Pi and make decisions based on the state of the driver. Referring to Figure 11, the system will be sensing if the driver has their eyes closed for too long (>3 seconds) by doing additional checks after delaying, thus mitigating the possibility of the camera sensing the driver blinking. This system will behave in a similar manner for the distracted state, firstly sensing if the drivers' eyes are off the road, but after an additional scan on the chance that the driver is checking their rear view mirror, looking left/right at a turn, etc. [MAK]

Similarly, the hand sensor subsystem works in a similar fashion, once again polling data as soon as the ignition is started. The signal from this subsystem feeds the microcontroller is split into three states; if the drivers' hands are both hand sensors, telling the system the driver maintaining adequate responsiveness, if the driver only has a single hand on the hand sensors, meaning they are being less responsible or possibly turning, and if the drivers' hands are completely off the steering wheel, possibly due to driving with their knee/not on the actual sensors or due to the driver being asleep at the wheel. Whilst continuously scanning the dual hand sensors, the subsystem will be sending one of the three state signals to the main module. Referring to Figure 12, this system will implement a similar scheme: firstly checking if both sensors are actuated, then checking if only one, and then finally if none. There are of course delays and additional checking if one of these states is in question (except for both actuated) to attempt to ignore the instances where a driver is turning and shifting their hands on the wheel. Finally, the subsystem will either continuously poll if there are no wrong-doings or issue a signal to the main system based on the severity of the infringement. [MAK]

Lastly, the OBD-II communications standards vary depending on what protocol that the vehicle is using. These various protocols have different voltage levels while also having different means of communication. Then there is also the conversion from their respective protocol to some other means of communication, such as UART, RS232 etc. that our main processing logic can easily handle and interpret. [MAM]

2.4. Communications

A system will be developed to convert the OBD-II port signals to Universal Asynchronous Receiver-Transmitter (UART) signals. This will allow the raspberry pi to communicate with the car's computer. OBD-II ports may fall under one of multiple protocols. There are five main protocols available for OBD-II standards. They are SAE J1850 VPW used in GM vehicles, ISO 9141-2 used in Chrysler, European, and Asian vehicles, ISO 14230 KWP2000 also used in Chrysler, European, and Asian vehicles, SAE J1850 PWM used in Ford vehicles, and ISO 15765 CAN used in vehicles sold in the United States from 2008 and afterward [10]. OBD-II standards utilize differential signals because cars produce a lot of electric noise. The differential signals are very resistant to noise and protect the integrity of the signals from electromagnetic interference from the car. Transceivers will be needed for each OBD-II standard in order to convert the differential signals into logic signals. The logic signals can then be converted from the various OBD-II standards to UART standard with an interpreter chip. Once converted to UART, the signals can be sent to the Raspberry Pi for further processing. UART is a basic serial communication scheme that the raspberry pi has built in. The system will have to be able to take any of these protocols as an input and convert it to a

logic UART signal so the raspberry pi can process the data. Also, the system will have to convert logic UART signals coming from the raspberry pi to differential signals for any of the OBD-II standards mentioned above so commands can be sent to the OBD-II and two way communication can be established. [MDM]

The converted signal (UART) will also be our means for communication back to the OBD-II port. Therefore, speed of ingestion and reading of data coming to and from the OBD-II port is crucial as well. A fast programming language will be needed to quickly do this task while also being robust enough to be adaptable for our future needs and wants for the project. [MAM]

2.5. Embedded Systems

A microcontroller, shown in Figure 3, will be used in the hand sensor subsystem to receive input from the force sensitive resistors, perform logic operations, create an output signal, and control the bluetooth low power module. A 16-bit architecture device programmed in C will be used. The device will have a 10-bit A/D converter for the inputs. The device output will use UART to communicate with the bluetooth module for signal transmission to the main controller. [BC]

A microcontroller will also be used for the OBD-II protocols translation into UART for communication to the main controller and vice versa. They are typically a 16-bit architecture device with various means of programming capabilities and commands. They are generally relatively low power as well, running on 3-5V, while also providing baud rates of up to 10Mbps. The device will communicate the translated data

to and from the main controller, sending and receiving data and commands, which we ultimately hope to fully harness the potential of. [MAM]

3. Engineering Requirements Specification

Marketing Requirements	Engineering Requirements	Justification
1	The system must alert the driver when it detects when the driver looks away from the road for 2 seconds.	Using either auditory or visual warning to inform, but not distract the driver. Distance traveled at 60 mph is 176 feet.
1	The system must alert the driver when it detects when the driver releases from the optimal position on the wheel for 1 second when the vehicle is not turning or stationary.	Using either auditory or visual warning to inform, but not distract the driver. Distance traveled at 60 mph is 44 feet.
1,2,3	The hand sensor subsystem will operate a minimum of 8 hours without being recharged.	With an embedded system and low energy bluetooth module this time should provide average driver enough time.
2,3	The system must be powered from the OBD-II port with a maximum current draw of 4 A.	Using a voltage regulator the main controller and peripherals will use only the OBD-II for power.
1,3,4	The system shall utilize eye tracking to detect if a driver is looking forward when the car is driving forward.	Eye tracking should be able to get adequate samples per second to confirm user attention level.
1,2,3,4	The system will read all OBD-II communications standards and converted to UART standard for on board communication.	Main module will have adequate storage space, and a system to record and organize data.
1,2	The eye tracking subsystem will be able to determine when the driver is drowsy or unattentive.	Eye tracking should be able to differentiate the levels of attention level.

1,4	The system will log an average number of infringements per hour of driving time.	The main controller can analyze data over time.
2,3	The hand sensor subsystem will wireless communicate to the main module and must have adequate range (1-5 feet) and maintain steady communication.	This range provides ample spacing between the devices and reliable communication.
2,3	System will start up automatically when vehicle starts and will operate without input from the driver.	The system must be easy/simple to setup and run for the average user.

Marketing Requirements:

1. System needs to increase drivers' self-awareness of performance.
2. System needs to not be intrusive nor interfere with vehicle operation.
3. System needs to be easy to install while also having a simple interface.
4. System needs to aid insurance companies in assessing driver responsibility.
5. System needs to have a reasonable price point.

[BC, MAK, MAM, MDM]

4. Engineering Standards Specification

	Standard	Use
Communications	UART	Microcontroller to bluetooth module, and OBD-II
	USB	Communicating from the camera to the Pi
	Bluetooth	Communication from steering device to the Pi
	CAN	Communication from OBD-II to the Pi
	ISO	Communication from OBD-II to the Pi
	J1850	Communication from OBD-II to the Pi
Data Formats	CSV	Organization and storage of important flagging and data.
Programming Languages	C	Embedded system programming
	Python	GUI and process control for eye-tracker
	Python	Bluetooth communication and data formatting for the Pi
Connector Standards	OBD-II	Connecting the Raspberry Pi to the vehicle
	USB	Connecting the webcam to the Raspberry Pi
	DB9	Connecting the OBD-II port to the transceivers

5. Accepted Technical Design

5.1. Hardware Design:

Level 0

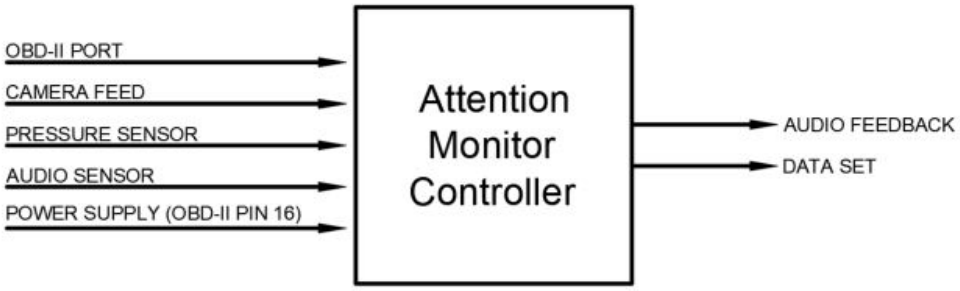

Figure 1. Level 0 Block Diagram Table. (BC, MK, MM, MDM)

Table 2. Level 0 Functional Requirements Table. (BC, MK, MM, MDM)

Functional Requirements Table	
Module	Vehicle Operator Attention Monitor
Designer	Matthew Mayfeild, Matt Krispinsky, Matt Marsek, Brian Call
Inputs	- OBD-II Port (To Measure Speed/Movement) - Camera Feed - Pressure Sensor - Audio Sensor - Power OBD-II Pin 16 12V power
Outputs	- Instantaneous Feedback Audio Response - Data Collection
Functionality	-Provide feedback to a driver when the driver is considered to be inattentive for too long of a period. -Store data to be transferred from the device for other analysis.

Level 1

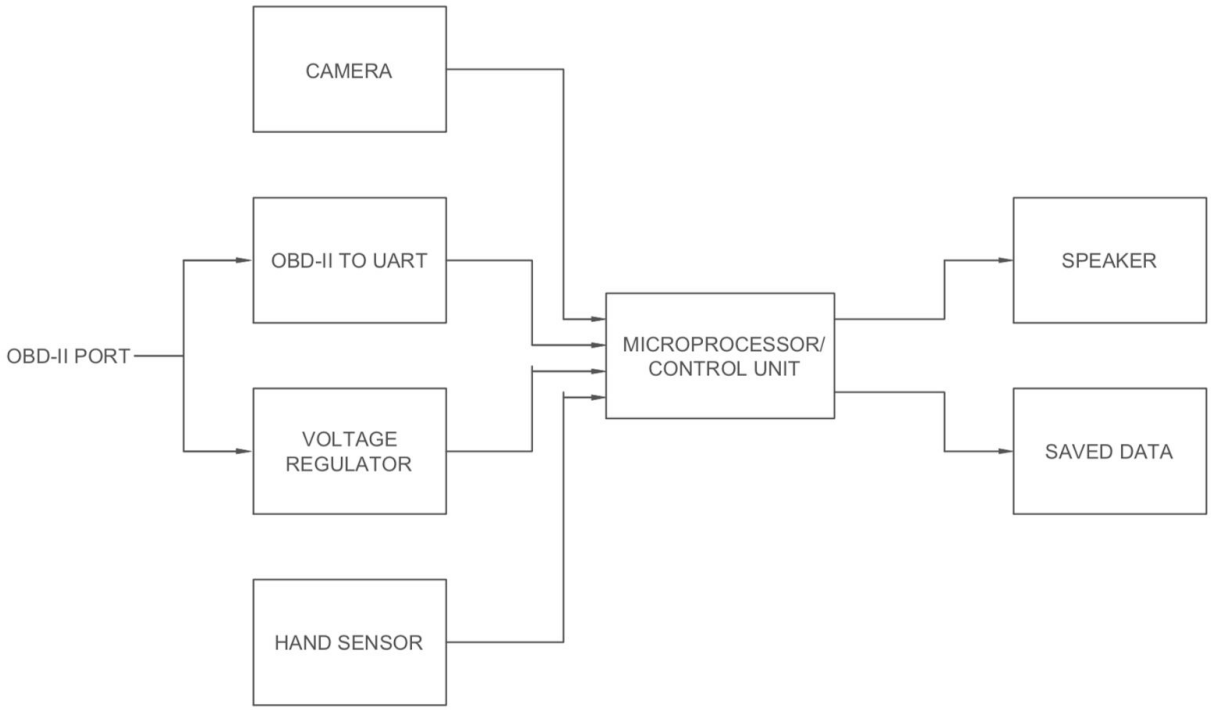

Figure 2. Block Diagram for Level 1 hardware. (BC, MDM)

Table 3. Functional Requirements Table for Level 1 Design of Hand Sensor. (BC)

Module	Hand Sensor
Designer	Brian Call
Inputs	-Battery Power Source -Two Sensors to detect hand location
Outputs	Wireless Feedback to main processing unit
Description	Provide feedback to the system indicating if the drivers hands are in a safe position.

Level 1

Table 4. Functional Requirements Table for Level 1 Design of Voltage Regulator. (MDM)

Module	Voltage Regulator
Designer	Matt Marsek
Inputs	-12 VDC from OBD-II port
Outputs	-5 VDC, 3 A to Raspberry Pi
Description	Converts OBD-II voltage to the proper voltage for powering the Raspberry Pi.

Table 5. Functional Requirements Table for Level 1 Design of OBD-II to UART Converter. (MDM)

Module	OBD-II to UART Converter
Designer	Matt Marsek
Inputs	-OBD II Port: Any of the protocols
Outputs	-UART to Raspberry Pi: 3.3 VDC
Description	Converts the OBD-II signals to UART signals so the OBD-II port and Raspberry Pi can communicate

Table 6. Functional Requirements Table for Level 1 Design of Camera. (MDM)

Module	Camera
Designer	Matt Marsek
Inputs	-Raspberry Pi connection: Power and control signals from Raspberry Pi
Outputs	-Data to Raspberry Pi: Video data to Raspberry Pi
Description	Camera to video the driver and detect when the driver is not looking at the road.

Level 1

Table 7. Functional Requirements Table for Level 1 Design of Speaker. (MDM)

Module	Speaker
Designer	Matt Marsek
Inputs	-Raspberry Pi connection: Audio signals from Raspberry Pi
Outputs	-Sound: Audible warnings
Description	Alerts the driver with the appropriate warning when it has been determined the driver is not paying attention to the road.

Table 8. Functional Requirements Table for Level 1 Design of Saved Data. (MDM)

Module	Saved Data
Designer	Matt Marsek
Inputs	-Raspberry Pi connection: Data from Raspberry Pi
Outputs	-None
Description	Stores data about how the driver drives for future use.

Level 2

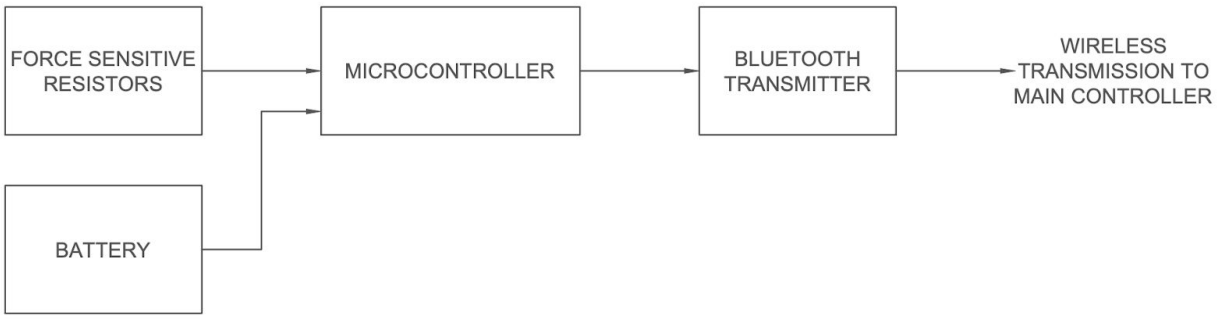

Figure 3. Block Diagram for Level 2 hand sensor hardware. (BC)

Table 9. Functional Requirements Table for Level 2 Design of Force Sensitive Resistors. (BC)

Module	Force Sensitive Resistors
Designer	Brian Call
Inputs	User input
Outputs	Analog signal to microprocessor
Description	Force sensitive resistor detects outside pressure, changes resistance to send an output signal.

Table 10. Functional Requirements Table for Level 2 Design of Battery. (BC)

Module	Battery
Designer	Brian Call
Inputs	DC power
Outputs	2-3.6V
Description	Battery and charging module to power the microprocessor

Table 11. Functional Requirements Table for Level 2 Design of Microcontroller. (BC)

Module	Microcontroller
Designer	Brian Call
Inputs	-DC Power -Signal from Force Sensitive Resistors
Outputs	Signal to Bluetooth transmitter
Description	Will process signals from the sensors, and send a new signal and control the bluetooth transmitter

Table 12. Functional Requirements Table for Level 2 Design of Bluetooth Transmitter. (BC)

Module	Bluetooth Transmitter
Designer	Brian Call
Inputs	-DC power (3.6V) -Signal from microcontroller
Outputs	Wireless transmission to main controller.
Description	Transmits the signal to the main controller with information of when the driver has their hands on the steering wheel.

Level 2

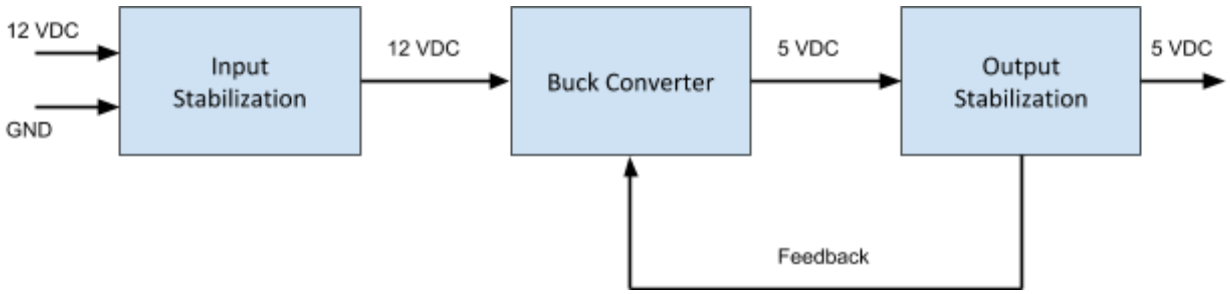

Figure 4. Block Diagram for Level 2 Voltage Regulator hardware. (MDM)

Table 13. Functional Requirements Table for Level 2 Design of Input Stabilization. (MDM)

Module	Input stabilization
Designer	Matt Marsek
Inputs	-12 VDC from OBD-II Port
Outputs	-12 VDC to Buck Converter
Description	Removes large voltage transients from appearing at the input

Level 2

Table 14. Functional Requirements Table for Level 2 Design of Buck Converter. (MDM)

Module	Buck Converter
Designer	Matt Marsek
Inputs	-12 VDC from input stabilization stage -Feedback
Outputs	-5 VDC to output stabilization stage
Description	Converts 12 VDC to 5 VDC and supplies 3 A

Table 15. Functional Requirements Table for Level 2 Design of Output Stabilization. (MDM)

Module	Output Stabilization
Designer	Matt Marsek
Inputs	-5 VDC from Buck Converter
Outputs	-5 VDC to Raspberry Pi -Feedback
Description	Provides feedback to the Buck Converter and keeps output ripple voltage low

Level 2

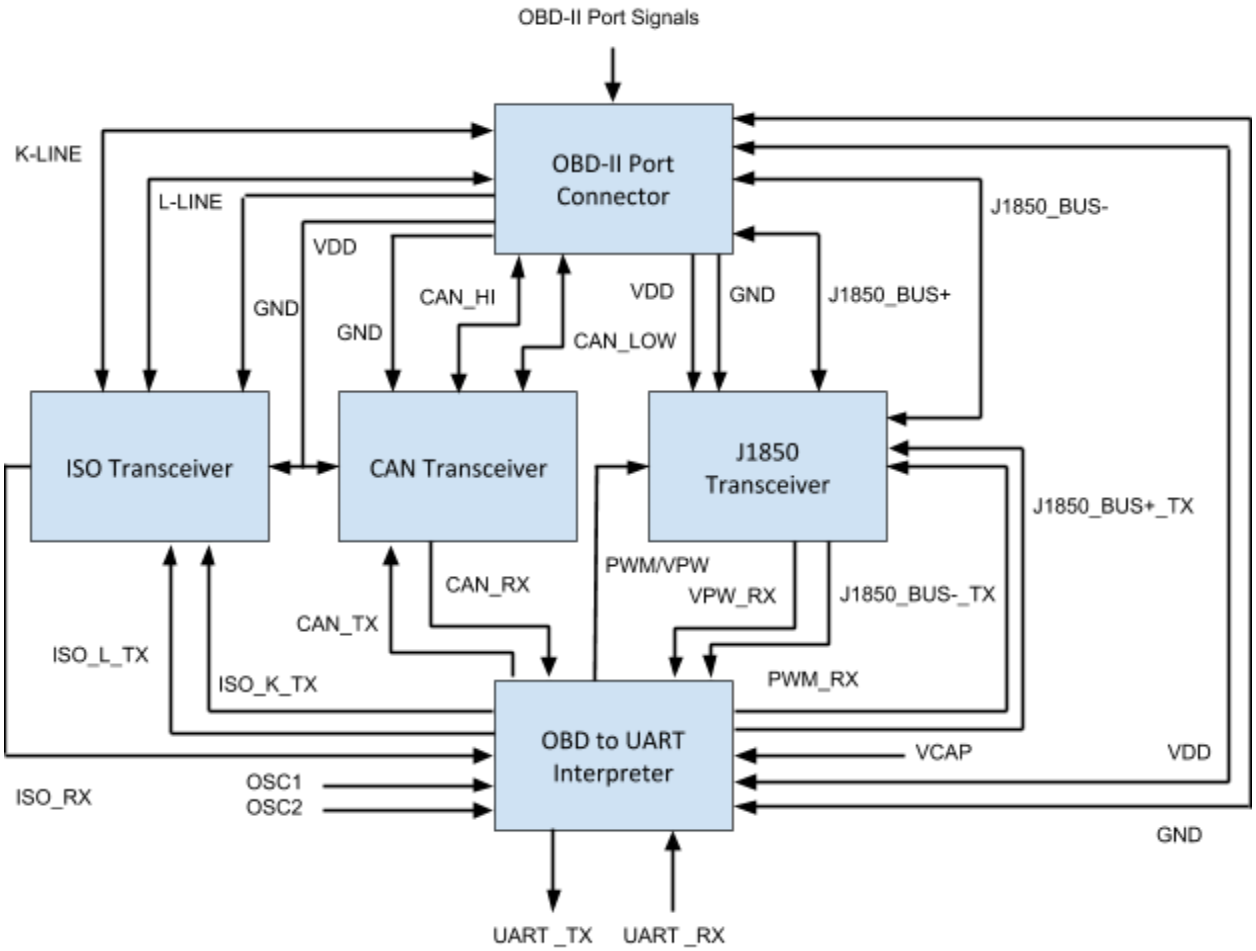

Figure 5. Block Diagram for Level 2 OBD-II to UART Converter hardware. (MDM)

Level 2

Table 16. Functional Requirements Table for Level 2 Design of OBD Port Connector. (MDM)

Module	OBD Port Connector
Designer	Matt Marsek
Inputs	-OBD-II port Signals: Any of the protocols -K-LINE: Digital input/output -L-LINE: Digital input/output -CAN_HI: Digital input/output -CAN_LOW: Digital input/output -J1580_BUS+: Digital input/output -J1580_BUS-: Digital input/output
Outputs	-VDD: 5 VDC -K-LINE: Digital input/output -L-LINE: Digital input/output -CAN_HI: Digital input/output -CAN_LOW: Digital input/output -J1580_BUS+: Digital input/output -J1580_BUS-: Digital input/output -GND: 0 VDC
Description	The physical connector that plugs into the OBD-II port and connects it to the rest of the circuitry.

Level 2

Table 17. Functional Requirements Table for Level 2 Design of ISO Transceiver. (MDM)

Module	ISO Transceiver
Designer	Matt Marsek
Inputs	-VDD: 5 VDC -K-LINE: Digital input/output -L-LINE: Digital input/output -ISO_K_TX: Digital input -ISO_L_TX: Digital input -GND: 0 VDC
Outputs	-ISO_RX: Digital output -K-LINE: Digital input/output -L-LINE: Digital input/output
Description	Converts the ISO protocol signals coming from the OBD Port from differential signals to logic signals.

Table 18. Functional Requirements Table for Level 2 Design of CAN Transceiver. (MDM)

Module	CAN Transceiver
Designer	Matt Marsek
Inputs	-VDD: 5 VDC -CAN_TX: Digital input -CAN_HI: Digital input/output -CAN_LOW: Digital input/output -GND: 0 VDC
Outputs	-CAN_RX: Digital output -CAN_HI: Digital input/output -CAN_LOW: Digital input/output
Description	Converts the CAN protocol signals coming from the OBD Port from differential signals to logic signals.

Level 2

Table 19. Functional Requirements Table for Level 2 Design of J1850 Transceiver. (MDM)

Module	J1850 Transceiver
Designer	Matt Marsek
Inputs	-VDD: 5 VDC -PWM/VPW: Digital input -J1850_BUS+_TX: Digital input -J1850_BUS-_TX: Digital input -J1580_BUS+: Digital input/output -J1580_BUS-: Digital input/output -GND: 0 VDC
Outputs	-PWM_RX: Digital output -VPM_RX: Digital output -J1580_BUS+: Digital input/output -J1580_BUS-: Digital input/output
Description	Converts the J1850 protocol signals coming from the OBD Port from differential signals to logic signals.

Level 2

Table 20. Functional Requirements Table for Level 2 Design of OBD-II to UART Interpreter. (MDM)

Module	OBD to UART Interpreter
Designer	Matt Marsek
Inputs	-VDD: 5 VDC -VCAP: Capacitor, 10 µF -OSC1: 16 MHz crystal oscillator -OSC2: 16 MHz crystal oscillator -ISO_RX: Digital input -CAN_RX: Digital input -PWM_RX: Digital input -VPW_RX: Digital input -UART_RX: 3.3 VDC -GND: 0 VDC
Outputs	-ISO_K_TX: Digital output -ISO_L_TX: Digital output -CAN_TX: Digital output -PWM/VPW: Digital output -J1850_BUS+_TX: Digital output -J1850_BUS-_TX: Digital output -UART_TX: 3.3 VDC
Description	Converts OBD-II signals to UART signals and vice versa.

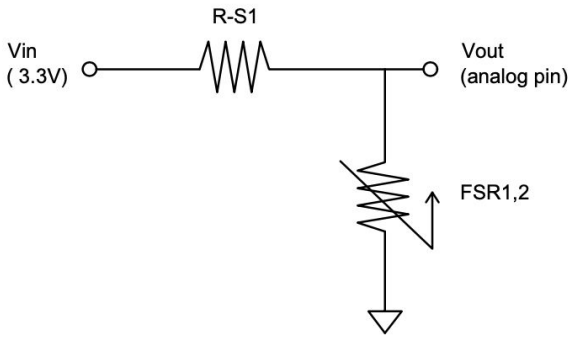

Figure 6. Steering Wheel Subsystem Force Sensitive Resistor Circuit. (BC)

The steering wheel subsystem is controlled by a PIC24FJ microcontroller programmed with C. In this design, two analog inputs and 1 digital output is utilized. The analog inputs are used for the hand sensors. The hand sensors, FSR1,2, use the SF15-150 force sensitive resistors. The range of this sensor is less than 50g to 10kg with a response time of less than 10ms. The figure above shows the circuit used with 3.3V input. R-S1 is a 10kΩ resistor. [B.C.]

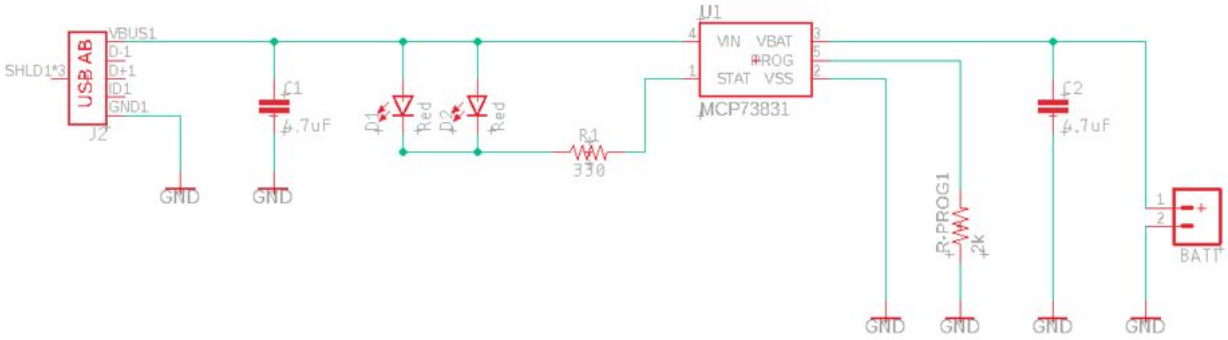

Figure 7. PCB Circuit Schematic: Power Supply. (BC)

The system is powered by a battery with MCP73831 to control the charging and voltage regulation. The circuit utilizes micro-usb to charge the battery. The battery and main power rail for the device is connected to the output 'VBAT'.

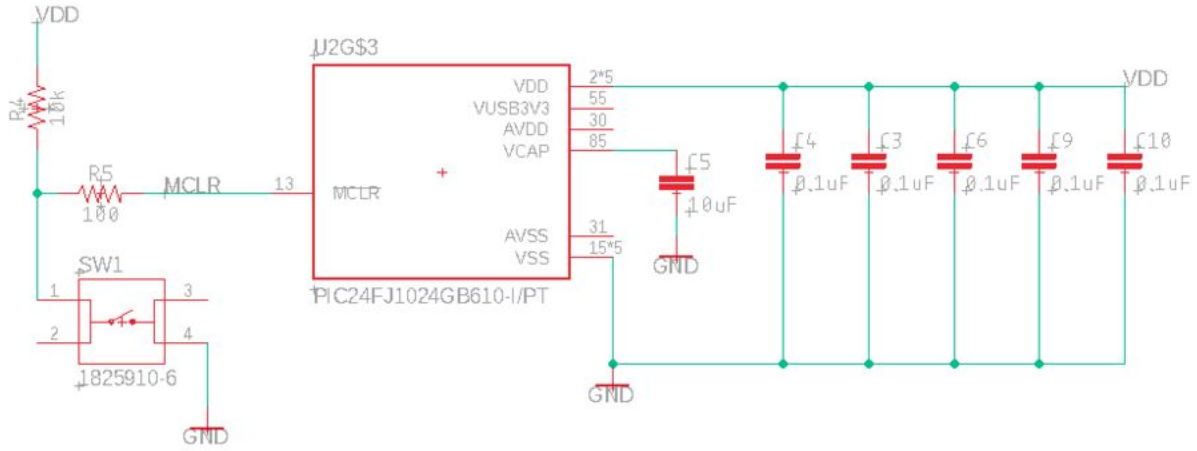

Figure 8. PCB Circuit Schematic: Microprocessor Power and Reset. (BC)

The PIC24FJ is set up to have a master clear switch at the MCLR input pin 13. The device is powered from the battery output. The minimum required connections recommended by the data sheet are used as guidelines for this design. The device requires six capacitors between VDD and VSS.

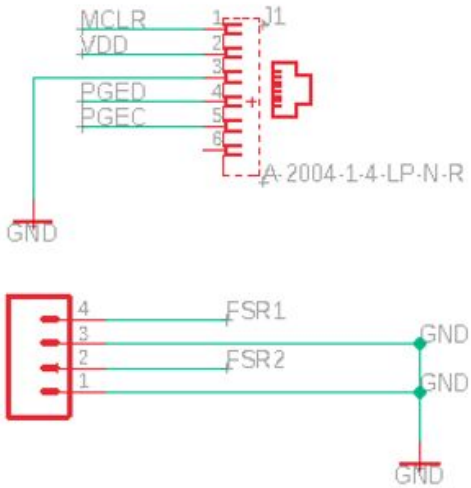

Figure 9. PCB Circuit Schematic: I/O Connections. (BC)

Figure 10. PCB Circuit Schematic: Microprocessor I/O. (BC)

The analog inputs AN3 and AN12 are connected to FSR2 and FSR1. The device is programmed using PGED PIN 25 and PGEC PIN 24 with a connection to an A-2004-1-4-LP-N-R jack. The jack will be used to program it and then removed to keep the form factor small. FSR1 and FSR2 are easy solder connections on the printed circuit board.

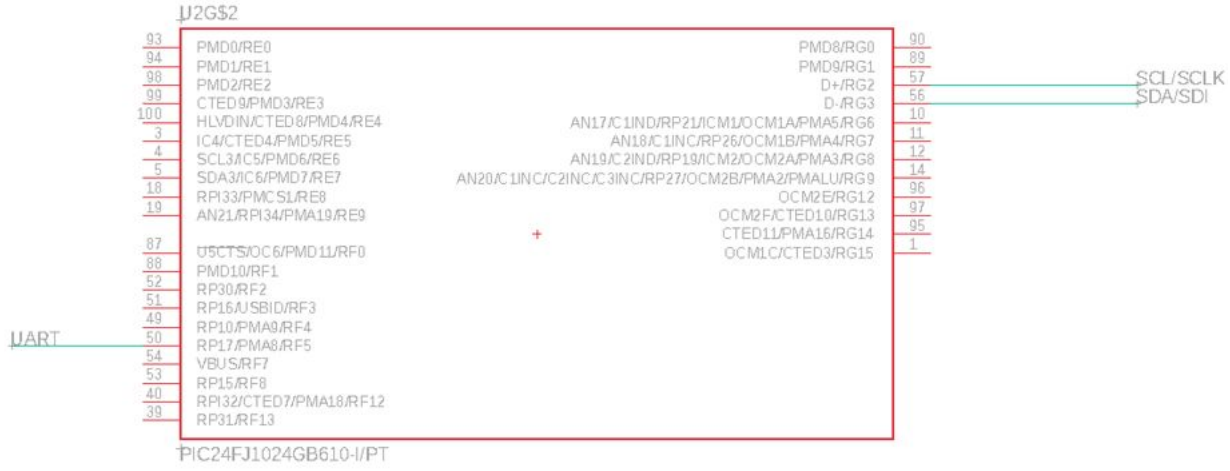

Figure 11. PCB Circuit Schematic: Microprocessor Communications. (BC)

The PIC24FJ transmits to the RN4870 bluetooth module using UART. The connections used are VDD, Ground, and TX on the PIC to RX on the RN4870. The current set up is one way communication (from the subsystem to the main RaspberryPi). [BC]

The PIC24FJ communicates with a MPU-9250 using I2C to get gyroscope and accelerometer data. SCL and SDA are connected to PINS 56 and 57.

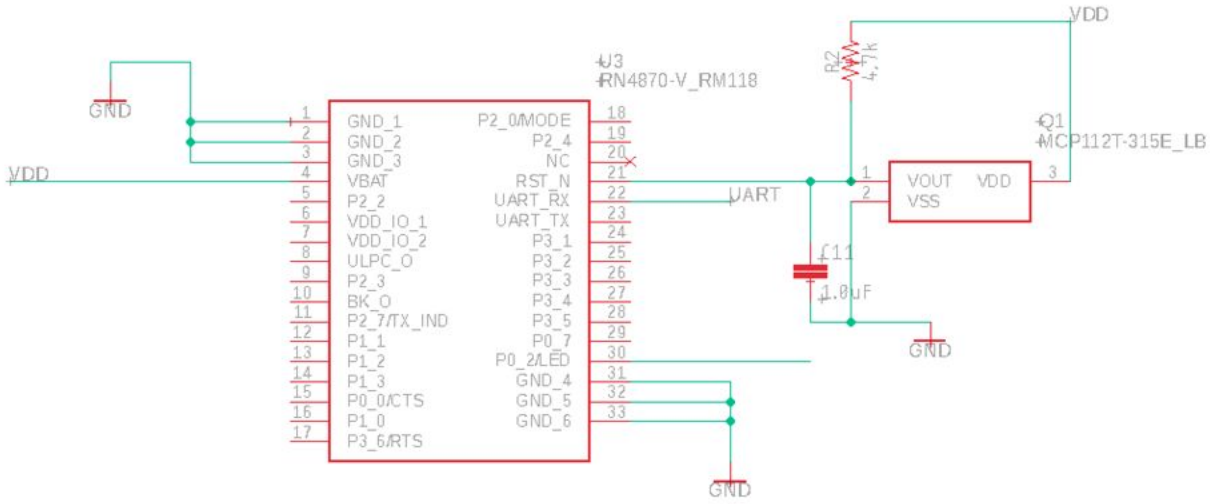

Figure 12. PCB Circuit Schematic: Bluetooth Module. (BC)

The bluetooth module is connected to the PIC24FJ using UART to receive communications. The device is powered with a MCP112T-315_LB to regulate power input.

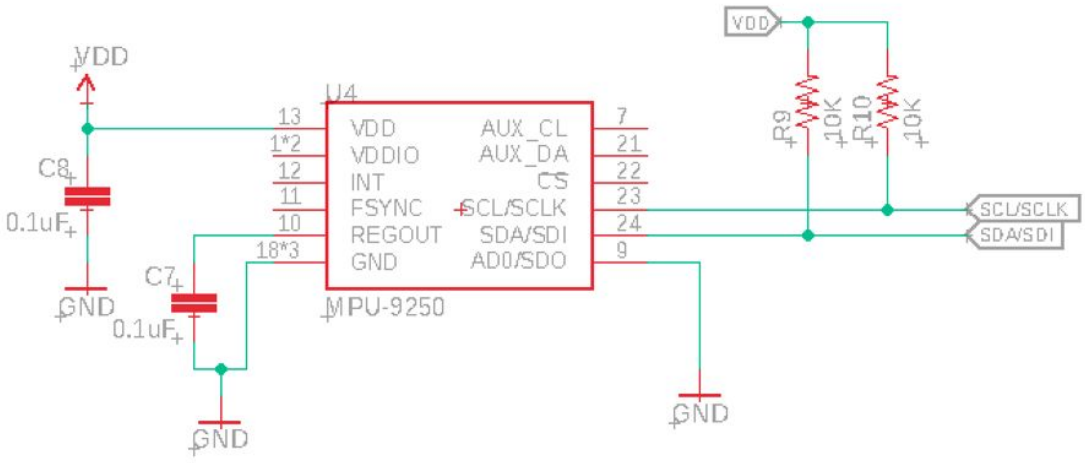

Figure 13. PCB Circuit Schematic: Accelerometer and Gyroscope. (BC)

The MPU-9250 uses I2C to communicate to the PIC24FJ. The SCL and SDA lines have 10KΩ pull up resistors to VDD.

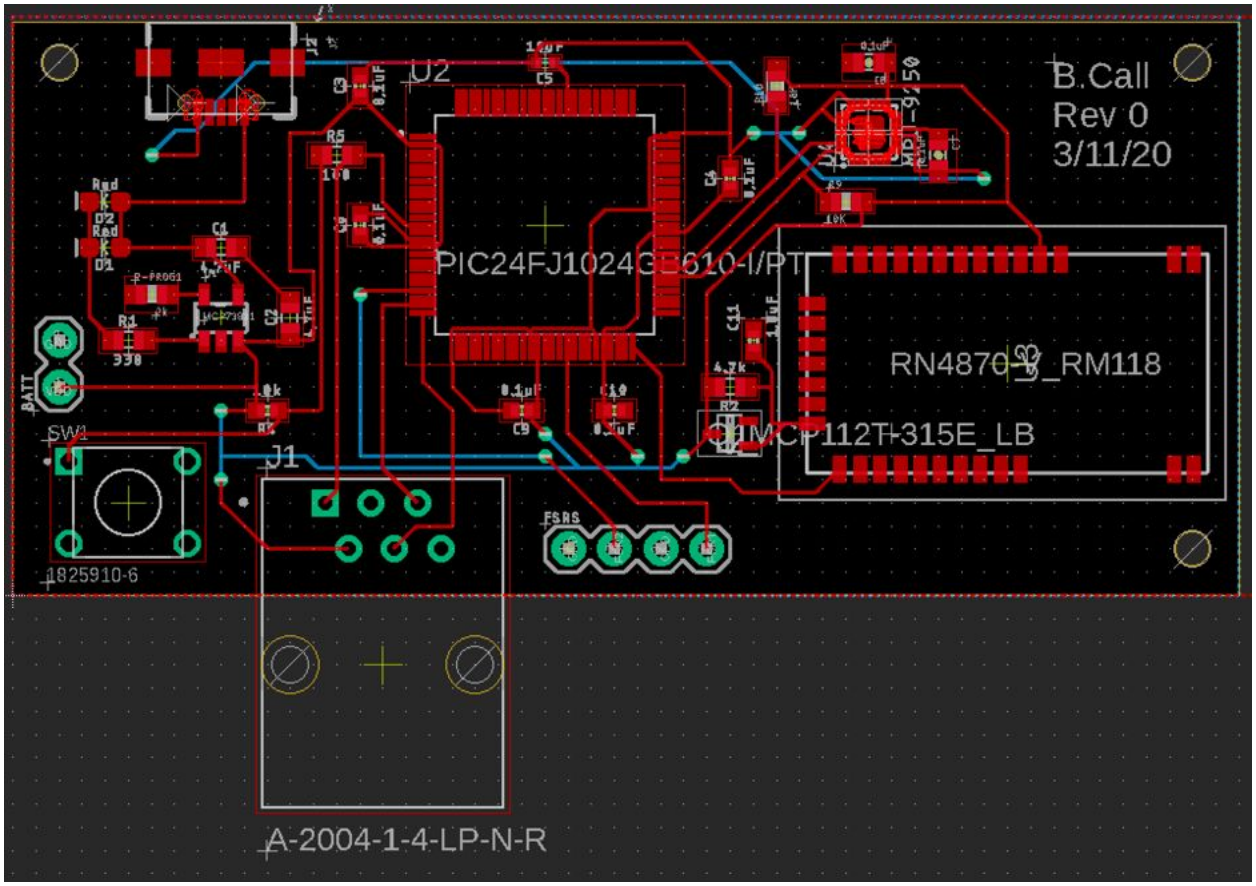

Figure 14. PCB Circuit Layout. (BC)

The printed circuit board layout is shown with the PIC24FJ in the center. The bluetooth module is isolated on the right. The left side has the power in, battery connection, and the charging regulator. The bottom has the programming connection. The accelerometer, gyroscope chip is located on the top right. The FSR connections are located in the bottom center. Additional layout work for the grounding and electromagnetic compatibility would have been done to ensure minimum spacing between communication components in the device. With UART, I2C, and bluetooth the device would be more susceptible to electromagnetic interference.

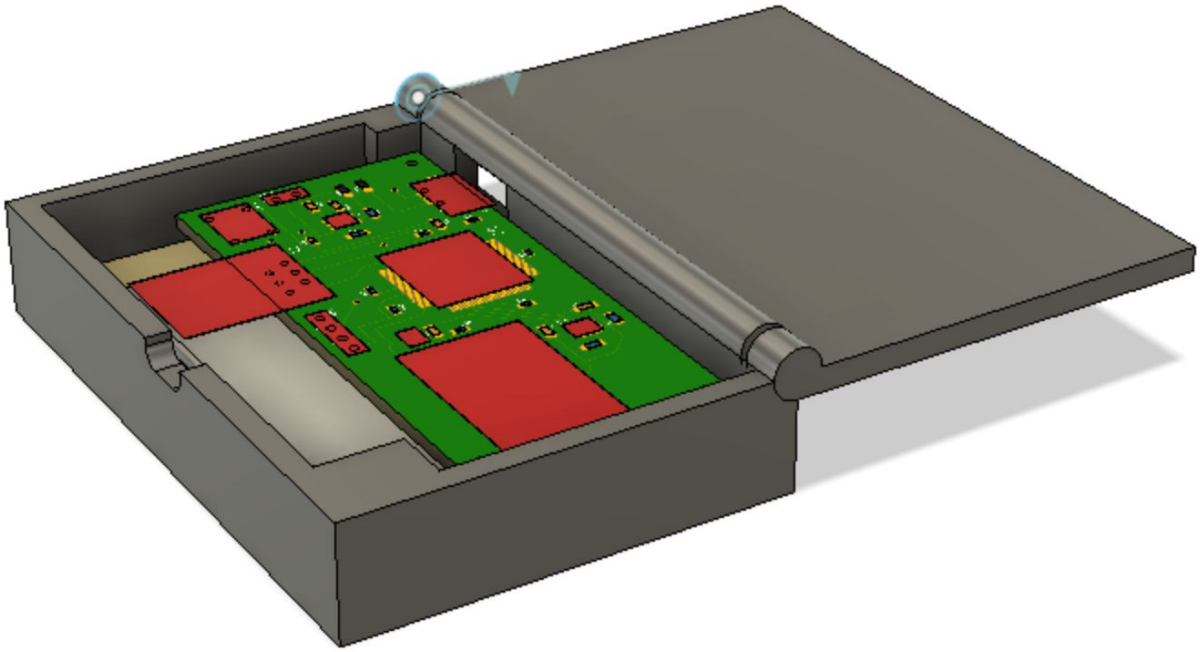

Figure 15. Steering Wheel Subsystem 3D Printed Case Design. (BC)

The printed circuit board will be placed in a 3D printed case to protect the components and to mount it to the steering wheel. It features openings for the charging connection and the outgoing cables to the FSRs and space for the battery to be mounted.

Voltage Regulator and OBD-II to UART Interpreter:

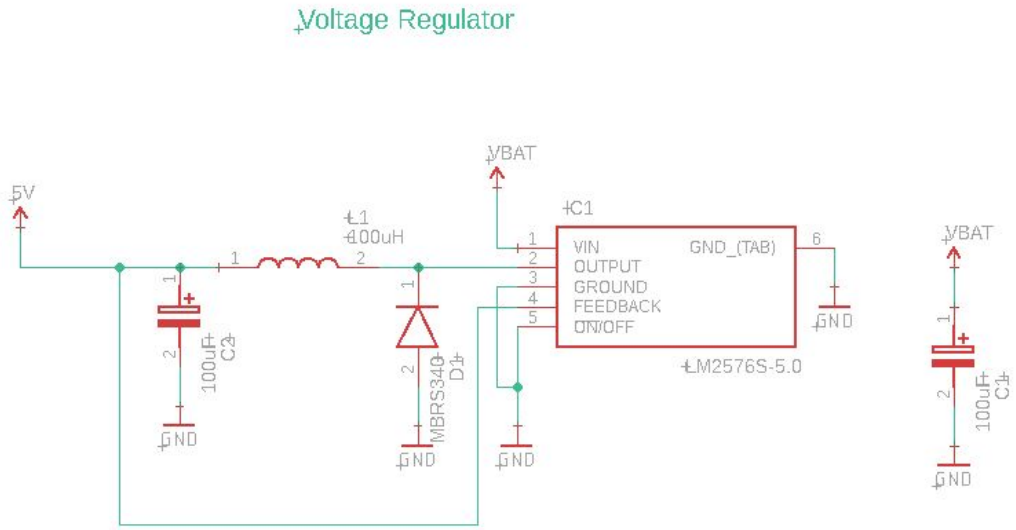

Figure 16. Schematic for the voltage regulator. (MDM)

The voltage regulator is used to power the Raspberry Pi from the car battery. The connection to the battery is through the VBAT pin of OBD-II port. The car battery supplies 12V to the input of the regulator which is fed to an LM2576 switching regulator which steps down the voltage from 12V to 5V at a current rating of up to 3A. This meets the power specifications of the Raspberry Pi. The output node 5V supplies 5V not only to the Pi but the rest of the circuits that require it as well. Capacitor C1 helps stabilize the input voltage. Diode D1, inductor L1, and capacitor C2 help stabilize the output. (MDM)

The Voltage Regulator was assembled on a breadboard at the lab bench. It was tested by connecting the battery pin of the OBD-II simulator, which supplies 12V, to the input and

connecting the output to the Raspberry Pi. A voltmeter was used to measure the voltage at the output of the regulator and approximately 5 volts was measured confirming it produced the correct output voltage. Next, it was connected to the Raspberry Pi for a more practical test. It successfully powered the Raspberry Pi. To test it further, a youtube video was played on the Pi in order to see if it would continue to power the Pi while its processing. It successfully powered the Pi throughout the test. (MDM)

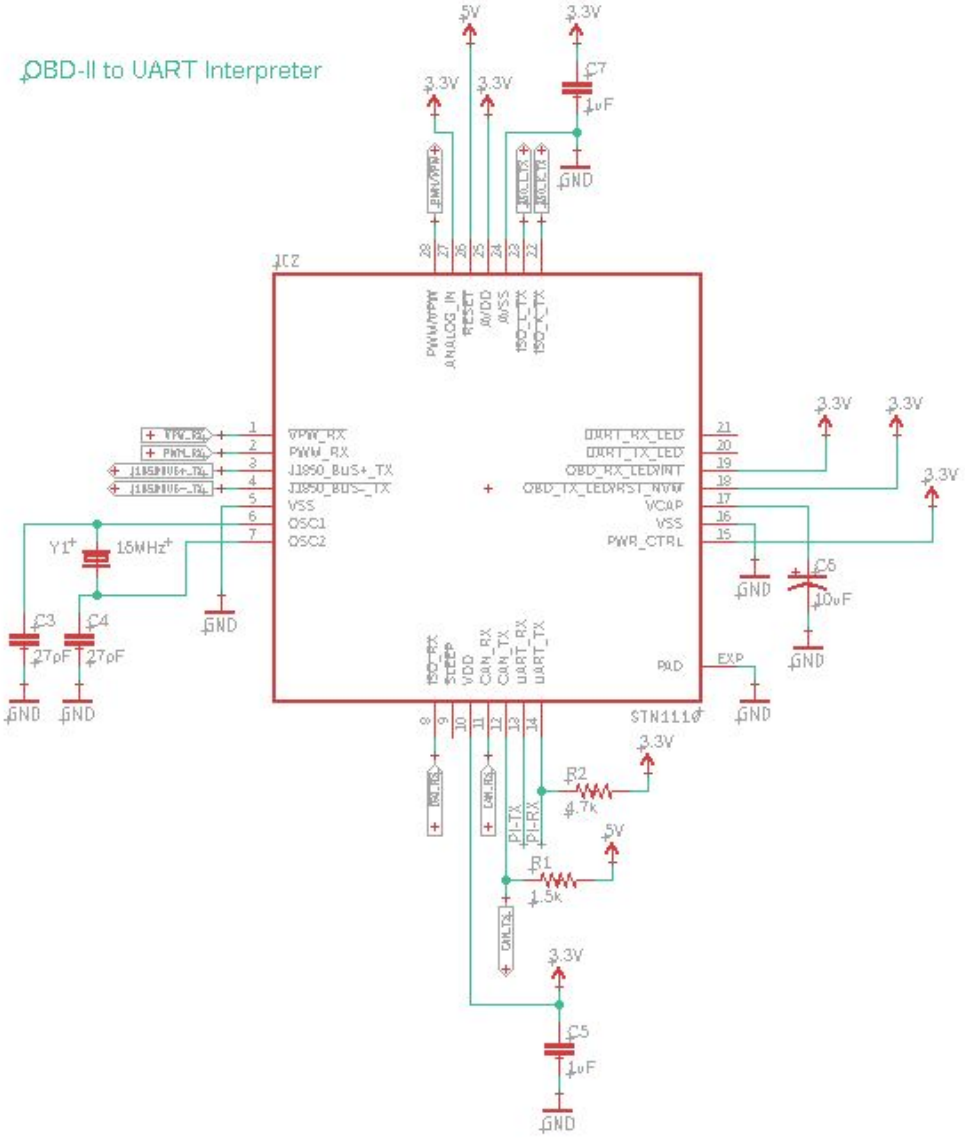

Figure 17. Schematic for the OBD-II to UART Interpreter. (MDM)

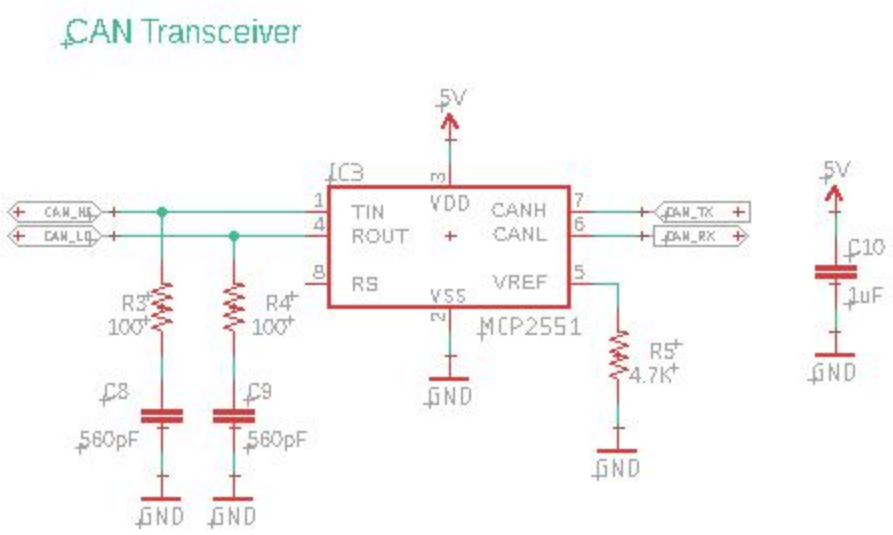

Figure 18. Schematic for the CAN transceiver. (MDM)

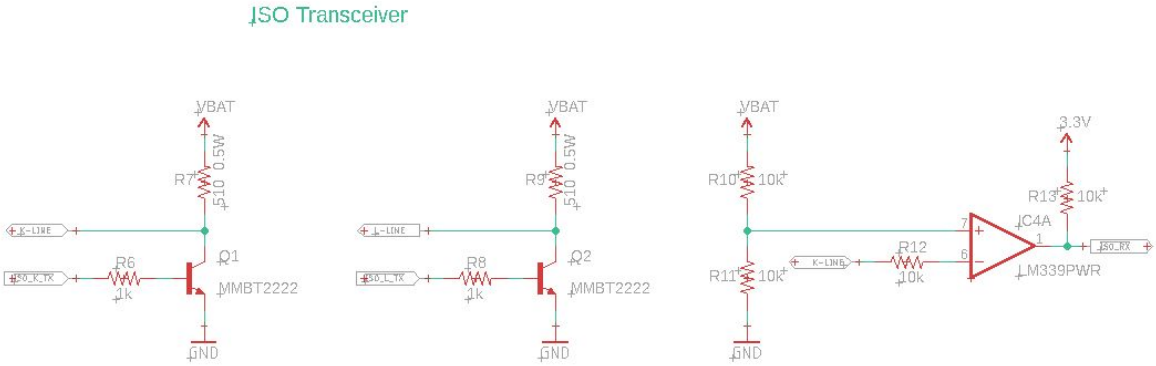

Figure 19. Schematic for the ISO transceiver. (MDM)

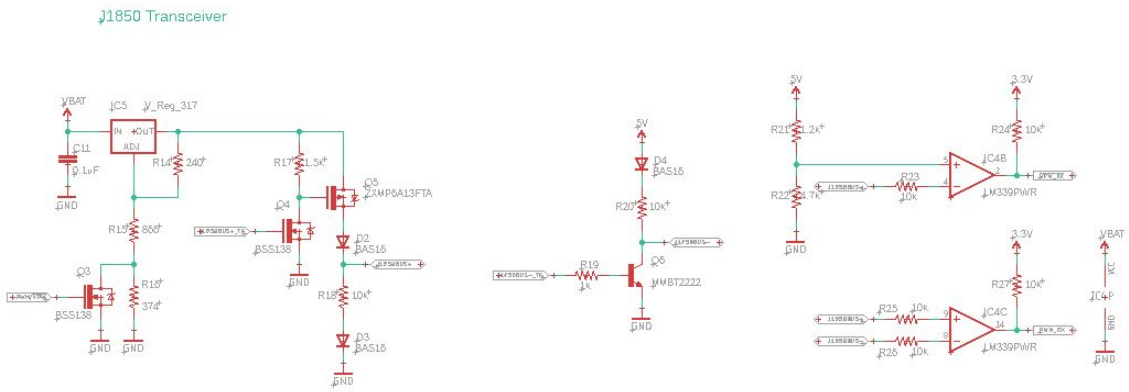

Figure 20. Schematic for the J1850 transceiver. (MDM)

The OBD-II to UART interpreter subsystem converts messages from CAN standard, ISO standard, and J1850 standard to UART standard using an STN1110 in combination with three transceivers. It also works in reverse and converts UART standard messages to CAN standard, ISO standard, and J1850 standard messages. This allows the Raspberry Pi to not only receive messages from the car's computer but to send commands as well. The three transceivers are composed of a CAN transceiver, an ISO transceiver, and a J1850 transceiver. Messages from CAN systems are sent from the OBD-II port to the CAN transceiver. Messages from ISO systems are sent from the OBD-II port to the ISO transceiver. Messages from J1850 systems are sentfrom the OBD-II port to the J1850 transsceiver. The transceivers convert the messages from differential signals to logic signals. The messages are then sent from their respective transceivers to the STN1110. There, they are then converted from CAN, ISO, or J1850 standard messages into UART standard messages. They are then sent from the STN1110 to the Raspberry Pi via UART connection for further processing. When the Raspberry Pi sends a command to the

car's computer, this process happens in reverse. The STN1110 and subsequent transceivers all have the ability to work in both directions. The 3.3V nodes are powered from one of the Raspberry Pi's various pins capable of supplying 3.3V. The 5V nodes are powered by the 12V to 5V voltage regulator. The 12V nodes are powered by the car battery through the VBAT OBD-II port pin. (MDM)

The OBD-II to UART interpreter circuit was assembled on a breadboard at the lab bench. To test it, it was hooked up to the OBD-II simulator and the Raspberry Pi with the appropriate connections. The parameter of vehicle speed was set to 50 km/h on the OBD-II simulator. An OBD-II command was then sent from a serial interface on the Raspberry Pi through the interpreter to the OBD-II simulator. This command requested the vehicle speed parameter. Once the command was sent, the OBD-II simulator responded immediately with a hexadecimal number on the Raspberry Pi's serial interface. This hexadecimal number contained the vehicle speed parameter of 50. To test it further, the speed parameter was changed to 60 km/h and the command sent again. The OBD-II simulator responded with the hexadecimal number for 60. The vehicle speed parameter was changed several more times and each time a command was sent requesting the data, it responded immediately with the accurate value. Then the test was repeated for a few other parameters. Each time the OBD-II simulator sent data to the Raspberry Pi quickly and the data values accurately reflected what the parameter values were set to. This confirmed the Raspberry Pi and OBD-II simulator were able to communicate back and forth and the OBD-II to UART interpreter ciruitry functioned as designed. (MDM)

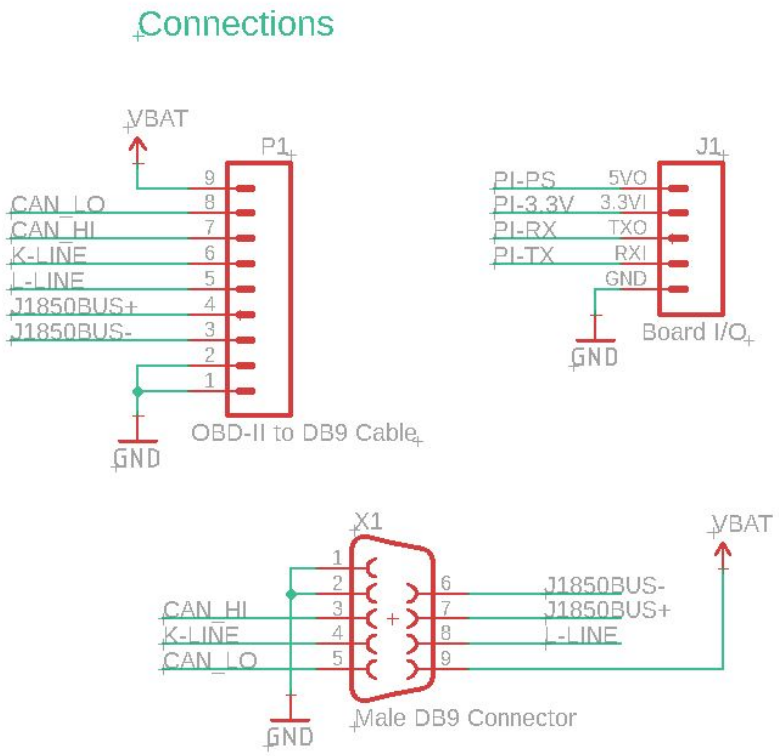

Figure 21. Schematic for the external connections of the OBD-II to UART interpreter subsystem. (MDM)

The schematic above shows the external connections that are made to the OBD-II to UART interpreter circuit. P1 is the cable that plugs into the car's OBD-II port connector and converts into a DB9 connector on the other side. The DB9 side of the cable plugs into the DB9 connector X1 which provides access to the pins for the rest of the circuitry. P1 and X1 show the pin out of the OBD-II port. J1 shows the other external connections of the interpreter and regulator circuits. 5VO is an output that connects from the output of the 12V to 5V regulator to the power input of the Raspberry Pi. 3.3VI is an input that connects from a 3.3V pin of the Pi

and supplies 3.3V to all the circuits that need it. TXO and RXI are the UART connections between the interpreter and the Pi. GND is a common ground connection between the interpreter and regulator circuits and the Raspberry Pi. (MDM)

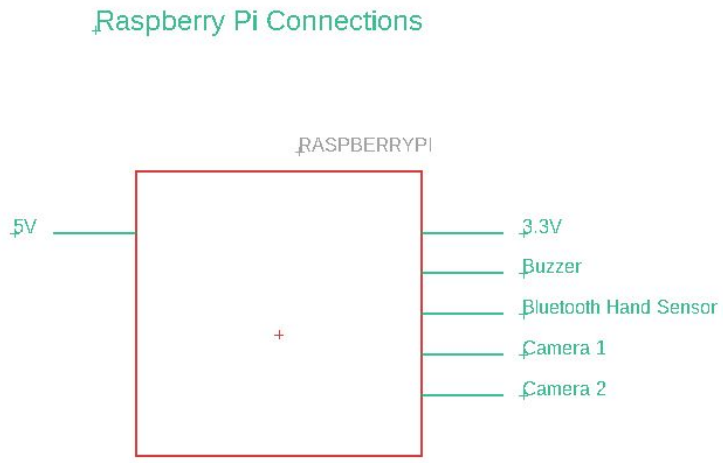

Figure 22. Schematic for the Raspberry Pi connections. (MDM)

The schematic above shows the main connection of the Raspberry Pi. The 5V node is the power supply input which comes from the 12V to 5V regulator. The 3.3V node supplies 3.3V to the OBD-II to UART interpreter where it is needed. The Buzzer node is a connection to a piezoelectric buzzer that will be used as a source of audio feedback to alert the driver when the system detects they are not paying attention. The Bluetooth Hand Sensor node is a bluetooth connection to the hand sensor that will be used to supply the Pi with data from the hand sensor

subsystem. The nodes Camera 1 and Camera 2 are connections to two respective webcams. The Pi will power the cameras which will be used in the eye tracker subsystem. (MDM)

Unfortunately, due to unpredictable circumstances the project was unable to be fully completed. The remaining steps would have been to create a PCB board for the voltage regulator and OBD-II to UART interpreter, order the necessary surface mounted components for the board according to the schematic, and create a housing to fit the circuitry in with openings for the connections that would have been needed. The subsystem as a whole would then have been integrated with the rest of the subsystems to create the completed project. (MDM)

5.2. Software Design

CANBus/OBD-II Level 0:

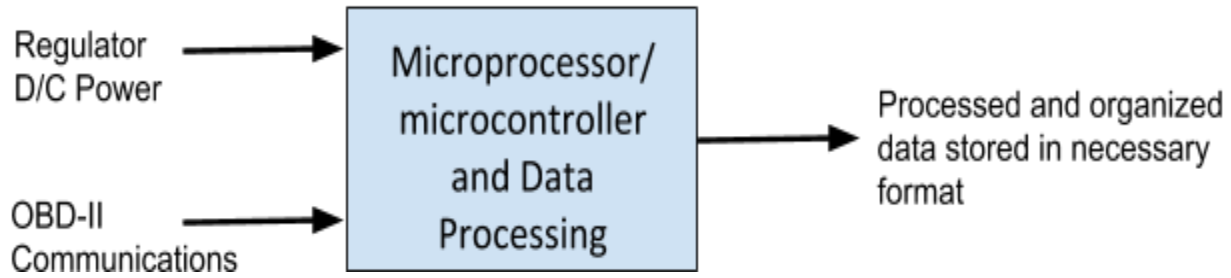

Figure 23. Block Diagram Level 0 Software Design of CANBus/OBD-II Communication.

Table 21. Functional Requirements Table for Level 0 Software Design of CANBus/OBD-II Communication.

Module	Microprocessor/Microcontroller and Data Processing
Designer	Matthew Mayfield
Inputs	D/C Power: either 5V or 12V OBD-II Communications: Mostly 0V to ~7V and transfer speeds of up to 1Mbps
Outputs	Fully processed and sorted data to be stored for data analysis
Description	Process all the OBD-II data and convert all received data to an easy to use format. Data is then stored for later documentation and data analysis

CANBus/OBD-II Level 1:

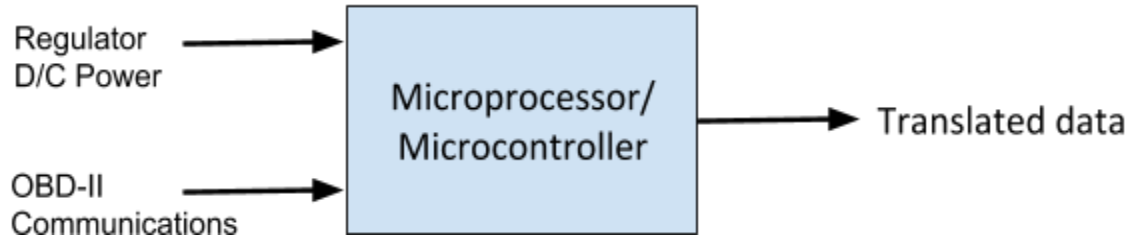

Figure 24. Block Diagram Table for Level 1 Software Design of CANBus/OBD-II Communication Translation Microcontroller/Microprocessor.

Table 22. Functional Requirements Table for Level 1 Software Design of CANBus/OBD-II Communication Translation Microcontroller/Microprocessor.

Module	Microprocessor/Microcontroller Unit
Designer	Matthew Mayfield
Inputs	D/C Power: either 5V or 12V OBD-II Communications: Mostly 0V to ~7V and transfer speeds of up to 1Mbps
Outputs	Various translated data points being sent at different times. Most likely plain text
Description	Communicate and convert the OBD-II data into an easy to use format for later data processing and analysis.

CANBus/OBD-II Level 1:

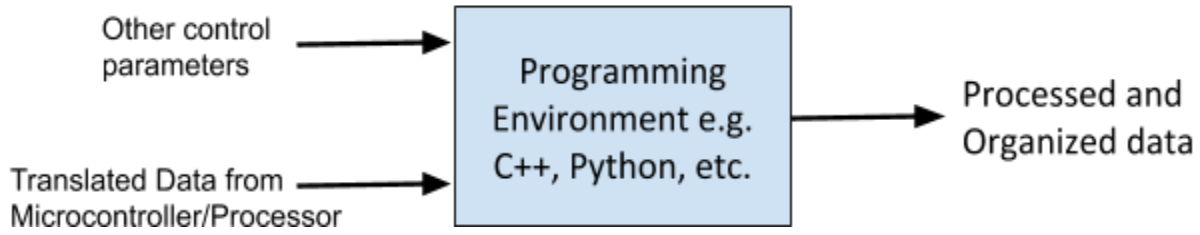

Figure 25. Block Diagram for Level 1 Software Design of CANBus/OBD-II Communication Translation Microcontroller/Microprocessor.

Table 23. Functional Requirements Table for Level 1 Software Design of CANBus/OBD-II Communication Translation Microcontroller/Microprocessor.

Module	C++ Programming environment
Designer	Matthew Mayfield
Inputs	OBD-II Translated Data: Either basic text or an easy to translate/process language Potential other parameters that have yet to be determined
Outputs	Processed and organized group of data to be analyzed
Description	Receive translated OBD-II Data and to process and organize the data to be later used for analysis or other means. Maybe stored in a .csv file

Eye Tracking Camera Software Level 0 Block Diagram:

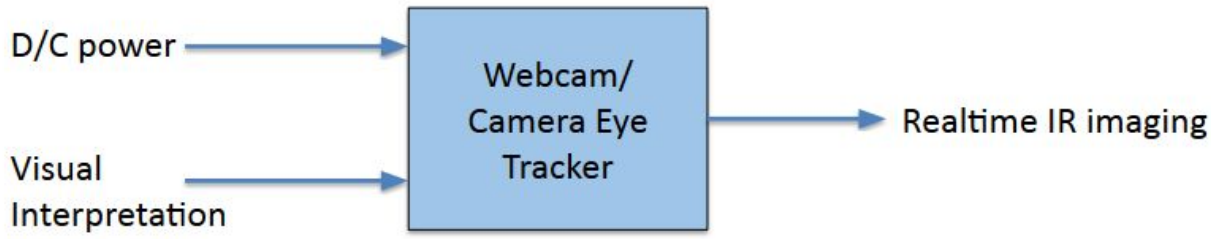

Figure 26. Block Diagram for Level 0 Software Design of Webcam/Eye Tracker. (MAK)

Table 24. Functional Requirements Table for Level 0 Software Design of Webcam/Eye Tracker. (MAK)

Module	Eye Tracking Webcam
Designer	Matthew Krispinsky
Inputs	D/C Power: 5V USB from Pi controller module Visual Interpretation: Visual IR feed of the driver. Depending on hardware, 15-30 FPS
Outputs	Real-time IR imaging of drivers' eyes
Description	Webcam will interpret drivers' eyes position throughout drive and feed Raspberry Pi data real-time

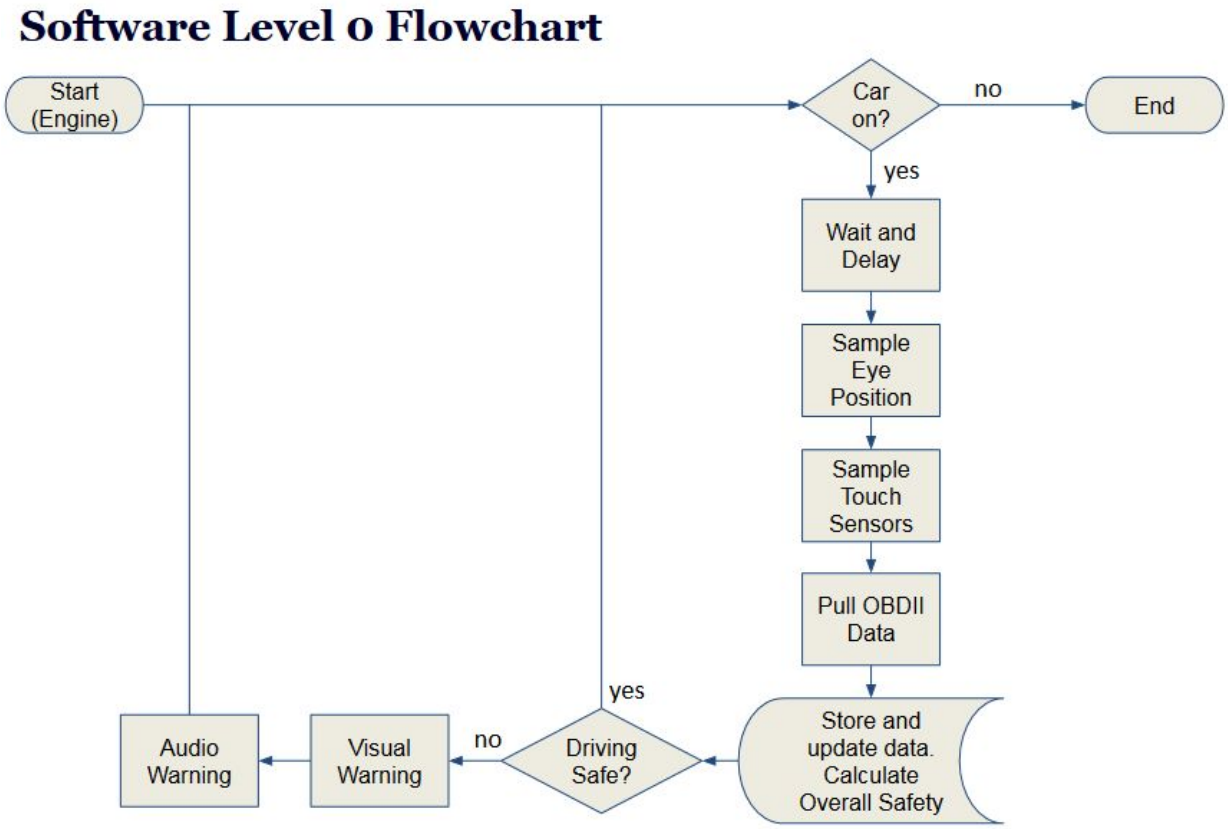

Figure 27. Level 0 Software Flowchart Design of Responsibility System. (MAK)

Software Level 1 Flowchart, Eye Tracking

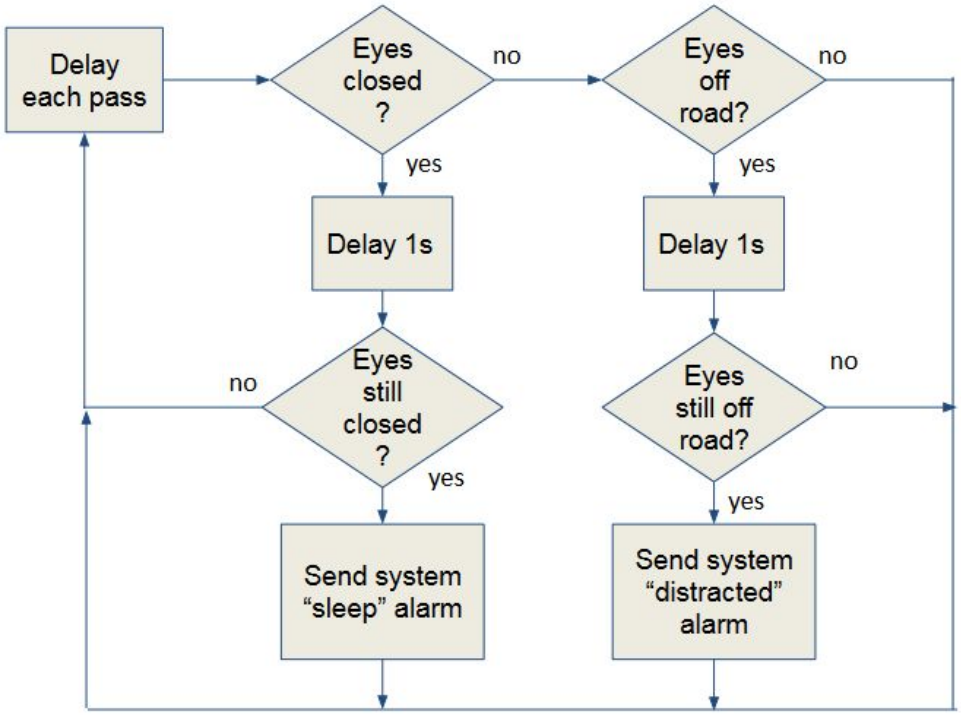

Figure 28. Level 1 Software Flowchart Design of Eye Tracker. (MAK)

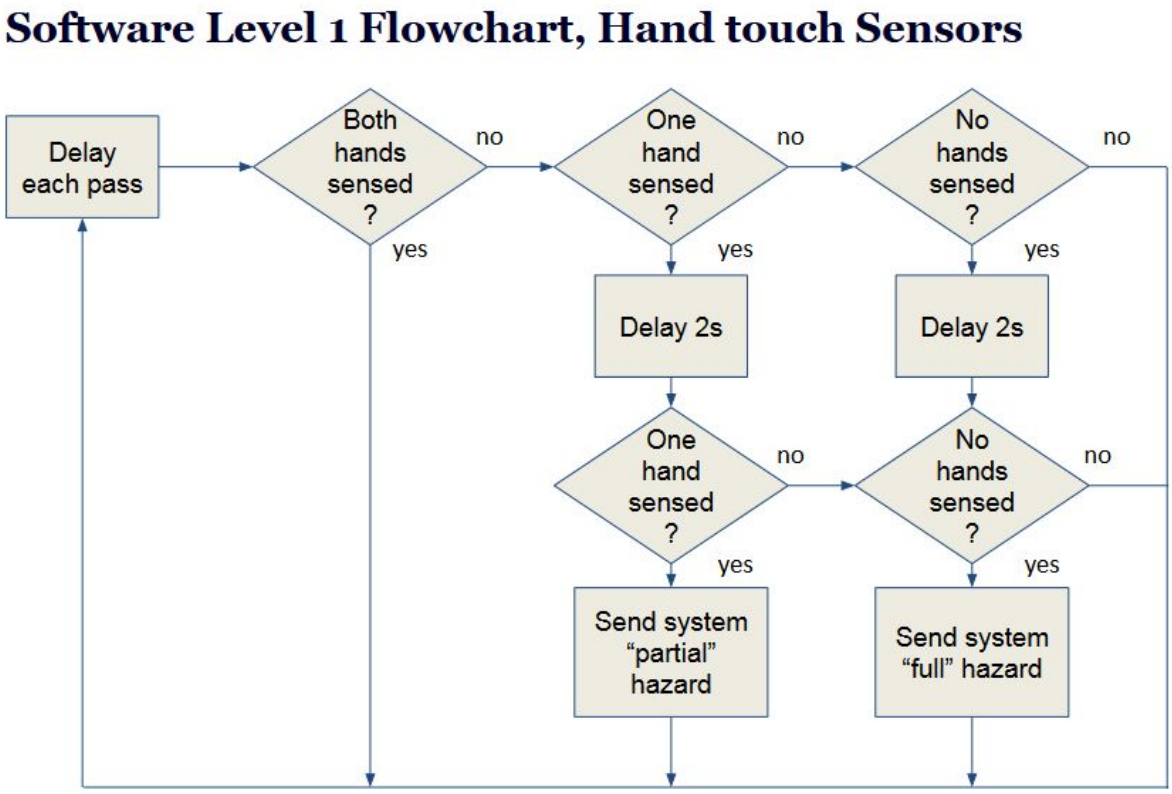

Figure 29. Level 1 Software Flowchart Design of Hand Touch Sensors. (MAK)

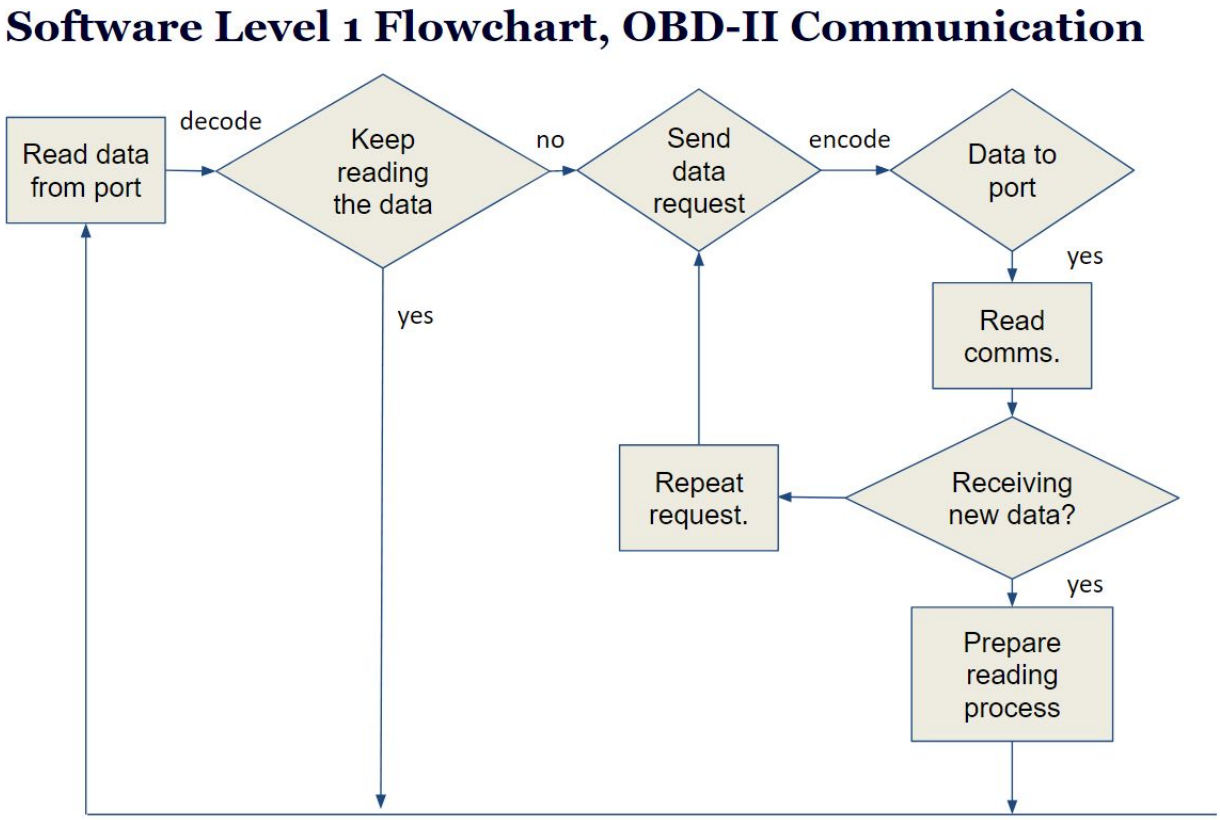

Figure 30. Level 1 Software Flowchart Design of OBD-II Communications. (MAM)

Figure 6, 7, 8 and Table 21, 22, 23 refer to the functional requirement diagram and tables for the OBD-II to UART data communications, the bluetooth data communications, and data interpretation. These are the basis for the main data collection and flagging for our system. Communications with the OBD-II port with the Raspberry Pi requires the use of a UART translation in-between for ease of implementation and use of all the features that the port can provide. The bluetooth data communications are vital for wireless communication with the steering wheel system, due to the moving nature of its location. Lastly the data interpretation and storage are necessary for later review of recorded trips using the entire system.

The bluetooth communication and data interpretation/storage code is written in python using the python bluetooth library as well as the datetime and csv libraries. The code begins by creating a .csv file using the current date and time, which would represent starting your vehicle for a trip. Then the program begins to set up the Raspberry Pi for a bluetooth communication and waits for a connection to be set up and pair to it. Once the connection is created, the code moves to reading the bluetooth communication data, which waits for some sort of terminating character before it interprets the data that it has received. Currently, that character is set as the space character, but it could be modified depending on other peoples usage. We can then interpret these now received communications after some basic stripping of other unnecessary transmitted information.

This is the bluetooth communication part of the code that is run from startup to shutdown, and is what gives the vital external sensor communication data that is required for properly recording the proper data at the moment of an issue or flag. It simply creates and opens the

bluetooth channel, and awaits the connection attempt from the external sensors, once paired, it constantly keeps checking what type of data the sensors are sending.[MAM]

```
#Importing the multiprocessing library
import multiprocessing
# Importing the Bluetooth Socket library
import bluetooth
# Importing the Date/Time library
import datetime
#Importing simplified time library to use timeout
import time
#Importing the csv library
import csv
#Importing serial communication library
import serial


def bluetoothcomm(UART_Q, SpeedCallSensor):

	#Setting up bluetooth connection
	host = ""
	port = 1	# Raspberry Pi uses port 1 for Bluetooth Communication

	# Creating Socket Bluetooth RFCOMM communication
	server = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
	print('Bluetooth Socket Created')
	#binding host to port
	try:
	server.bind((host, port))
	print("Bluetooth Binding Completed")
	except:
	print("Bluetooth Binding Failed")

	server.listen(1) # One connection at a time
	# Server accepts the clients request and assigns a mac address.
	client, address = server.accept()
	print("Connected To", address)
	print("Client:", client)

	with open(currentCombined + ".csv", 'a') as csvfile1: #create the CSV file with date
and time as name
	filewriter1 = csv.writer(csvfile1)
	try:
		while True:
		csvfile1.close()
		data = ""
		loop2 = True
		while loop2:
			# Receivng the data.
			btData = client.recv(1024) # 1024 is the buffer size.
			charData = str(btData) # convert the incoming character data as a string
```

```
                        charData = charData.lstrip('b') #removing pieces of unnecissary
transmitted information
                        charData = charData.lstrip("'")
                        charData = charData.rstrip("'")
                        if charData != ' ': #' ' is our string ending character.
                        data = data + charData
                        else:
                        loop2 = False
                print(data) #display the data on rasppi

                currentBlueDT = datetime.datetime.now() #datetime.datetime.now() calls the
current date and time in the forem
                currentBlueDate = str(currentBlueDT.strftime("%m/%d/%Y")) #current date in
mm/dd/yyyy format
                currentBlueTime = str(currentBlueDT.strftime("%H:%M:%S:%f")) #current time in
hh:mm:ss:uuuuuu

                if data == 'r': #Stand in command for data coming from wheel sensors
                        with open(currentCombined + ".csv", 'a') as csvfile1: #create the CSV
file with date and time as name
                        filewriter1 = csv.writer(csvfile1)
                        send_data = "Right Hand Sensor Triggered"
                        print(send_data)
                        row = [currentBlueDate, currentBlueTime, send_data]
                        filewriter1.writerow(row)
                        csvfile1.close()
                elif data == 'l':
                        with open(currentCombined + ".csv", 'a') as csvfile1: #create the CSV
file with date and time as name
                        filewriter1 = csv.writer(csvfile1)
                        send_data = "Left Hand Sensor Triggered"
                        print(send_data)
                        row = [currentBlueDate, currentBlueTime, send_data]
                        filewriter1.writerow(row)
                        csvfile1.close()
                elif data == 'rs': #stand in command for data coming from steering

                        with open(currentCombined + ".csv", 'a') as csvfile1: #create the CSV
file with date and time as name
                        filewriter1 = csv.writer(csvfile1)
                        send_data = "Right Hand Sensor with Speed Call Triggered"
                        print(send_data)
                        UART_Q.put('1')
                        while(SpeedCallSensor.empty()):
                                time.sleep(.05)
                        BluetoothSpeedData = SpeedCallSensor.get()
                        row = [currentBlueDate, currentBlueTime, send_data, BluetoothSpeedData]
                        row = [currentBlueDate, currentBlueTime, send_data]
                        filewriter1.writerow(row)
                        csvfile1.close()

                elif data == 'ls':
```

```
                        with open(currentCombined + ".csv", 'a') as csvfile1: #create the CSV
file with date and time as name
                        filewriter1 = csv.writer(csvfile1)
                        send_data = "Left Hand Sensor with Speed Call Triggered"
                        print(send_data)
                        UART_Q.put('1')
                        while(SpeedCallSensor.empty()):
                               time.sleep(.05)
                        BluetoothSpeedData = SpeedCallSensor.get()
                        row = [currentBlueDate, currentBlueTime, send_data, BluetoothSpeedData]
                        filewriter1.writerow(row)
                        csvfile1.close()

                elif data == 'quit': #end the program and close the file
                        send_data = "Quitting"
                        print(send_data)
                        loop1 = False
                else:
                        send_data = "Type r for right, l for left, and s for current speed"
                        print(send_data)

                # Sending the data to the client.
                client.send(send_data)
                csvfile1.close() #close csv file

        except:
        # Closing the client and server connection
                client.close()
                server.close()
```

This is the UART communication part of the code that is designated to talk through UART to the OBD to UART interpreter chip to send and receive commands and data with the vehicle OBD-II port. This iteration of the code focuses requesting speed data when another part of the complete program requests it.[MAM]

```
def UARTSerialComm(UART_Q, SpeedCallEye, SpeedCallSensor):

        #Setting UART communication standards needed for UART to CAN chip
        UARTser = serial.Serial('/dev/ttyAMA2', baudrate=9600, parity = serial.PARITY_NONE,
stopbits=serial.STOPBITS_ONE, bytesize=serial.EIGHTBITS, timeout=.3)

        while(True):

        while (UART_Q.empty()):
                time.sleep(.25) # wait 1/4 second
                #modifying/cleaning useful data from UART comms
                UARTser.write(str.encode('010d\r'))
```

```
                UARTserData = UARTser.read(size=64)
                UARTserStr = UARTserData.decode()
                UARTserStr = UARTserStr.lstrip("010d\r")
                UARTserStr = UARTserStr.rstrip(">\r\r ")
                SpeedKPH = UARTserStr.lstrip("41 ")
                SpeedKPH = SpeedKPH.lstrip("0")
                SpeedKPH = SpeedKPH.lstrip("D ")
                SpeedKPH = int(SpeedKPH, 16)
                SpeedMPH = SpeedKPH/1.609344
                currentUARTDT = datetime.datetime.now() #datetime.datetime.now() calls the
current date and time in the forem
                currentUARTDate = str(currentUARTDT.strftime("%m/%d/%Y")) #current date in
mm/dd/yyyy format
                currentUARTTime = str(currentUARTDT.strftime("%H:%M:%S:%f")) #current time in
hh:mm:ss:uuuuuu
                with open(currentCombined + "SpeedTracking" + ".csv", 'a') as csvfile2: #create
the CSV file with date and time as name
                filewriter2 = csv.writer(csvfile2)
                filewriter2.writerow([currentUARTTime, SpeedKPH, SpeedMPH])
                csvfile2.close()
        #taking in UART queue request to see which process called for data
        UARTSignal = UART_Q.get()

        if UARTSignal == '2':
                SpeedCallEye.put(SpeedKPH) #Eye tracking speed call

        elif UARTSignal == '1':
                SpeedCallSensor.put(SpeedKPH) #Bluetooth sensor speed call

        UARTser.close
```

This part of the code is to simulate the incoming eyetracking flags from the other python process that is designed for eye tracking, however, we did not get the opportunity to fully integrate the two different pieces of code. Thus, that is why the incoming flag is simulated with just a timer creating the flag. If fully integrated, the eye tracking software would be setting the flag.[MAM]

```
#defining "eyetracking" subprocess
def eyetracker(UART_Q, SpeedCallEye):

        while(True):

        time.sleep(2) #sleep for 2 seconds
        if True:
```

```
                print("Eyetracking warning has been triggered and will now be recorded")
                currentEyeDT = datetime.datetime.now() #datetime.datetime.now() calls the
current date and time in the forem
                currentEyeDate = str(currentEyeDT.strftime("%m/%d/%Y")) #current date in
mm/dd/yyyy format
                currentEyeTime = str(currentEyeDT.strftime("%H:%M:%S:%f")) #current time in
hh:mm:ss:uuuuuu
                with open(currentCombined + ".csv", 'a') as csvfile: #create the CSV file with
date and time as name
                filewriterEye = csv.writer(csvfile)
                rowEye = [currentEyeDate, currentEyeTime, "Eye Tracking Sensor was Triggered"]
                filewriterEye.writerow(rowEye)
                csvfile.close() #close csv file

                #Simulating a request to put speed into the eyetracking trigger
                UART_Q.put('2')
                while (UART_Q.empty == False):
                time.sleep(.05)

        #This is checking if there is any data inside of "SpeedCallEye" array, which is shared
with the UART comms subprocess
        if (SpeedCallEye.empty() == False):
                time.sleep(.3)
                print("Eyetracking warning with speed record has been triggered and will now be
recorded")
                currentEyeDT = datetime.datetime.now() #datetime.datetime.now() calls the
current date and time in the forem
                currentEyeDate = str(currentEyeDT.strftime("%m/%d/%Y")) #current date in
mm/dd/yyyy format
                currentEyeTime = str(currentEyeDT.strftime("%H:%M:%S:%f")) #current time in
hh:mm:ss:uuuuuu
                with open(currentCombined + ".csv", 'a') as csvfile: #create the CSV file with
date and time as name
                filewriterEye = csv.writer(csvfile)
                rowEye = [currentEyeDate, currentEyeTime, "Eye Tracking Sensor with speed
requested was Triggered", SpeedCallEye.get_nowait()]
                filewriterEye.writerow(rowEye)
                csvfile.close() #close csv file
```

This last bit of code is just the setup of running all these different parts of the code concurrently, that way BT communications and UART communications can run at the same time, while also another bit of code could be writing to a file. [MAM]

```
if __name__ == "__main__":
```

```
        #Since these variables are used for creating and referencing the file, they are
declared outside of the processes
        currentDT = datetime.datetime.now() #datetime.datetime.now() calls the current date and
time in the forem
        currentDate = str(currentDT.strftime("%m-%d-%Y")) #current date in mm/dd/yyyy format
        currentTime = str(currentDT.strftime("_%H-%M-%S-%f")) #current time in hh:mm:ss:uuuuuu
        currentCombined = currentDate + currentTime
        print (currentDate) #display current date and time
        print (currentTime)
        print (currentCombined)




        #Creating initial .csv file ahead of time before functions write to it
        with open(currentCombined + ".csv", 'w') as csvfile1: #create the CSV file with date
and time as name
        filewriter1 = csv.writer(csvfile1, delimiter=',', quotechar='|',
quoting=csv.QUOTE_MINIMAL)
        filewriter1.writerow(['Date','Time', 'Cause of Flag', 'Speed in KPH when Flag
Occured'])
        time.sleep(.25)
        csvfile1.close() #close the file. We will open it back up for appending later

        #Also creating initial .csv speedtracking file ahead of time before functions write to
it
        with open(currentCombined + "SpeedTracking" + ".csv", 'w', newline='\n') as csvfile2:
#create the CSV file with date and time as name
        filewriter2 = csv.writer(csvfile2, delimiter=',', quotechar='|',
quoting=csv.QUOTE_MINIMAL)
        filewriter2.writerow(['Time', 'Speed in KPH', 'Speed in MPH'])
        time.sleep(.25)
        csvfile2.close()

        #Creating a multiprocessing queue to request speed data from UART Serial Comm
        UART_Q = multiprocessing.Queue()
        SpeedCallEye = multiprocessing.Queue()
        SpeedCallSensor = multiprocessing.Queue()

        # creating new processes
        p1 = multiprocessing.Process(target=bluetoothcomm, args=(UART_Q, SpeedCallSensor,))
        p2 = multiprocessing.Process(target=UARTSerialComm,
args=(UART_Q,SpeedCallEye,SpeedCallSensor,))
        p3 = multiprocessing.Process(target=eyetracker, args=(UART_Q, SpeedCallEye,))

        # running processes
        p1.start()
        p2.start()
        p3.start()
```

The testing procedure for the code was actually tied pretty hand in hand with the creation and testing of the various transceivers and OBD to UART interpreter hardware. After they were initially put together and shown that communication was possible, they became the live test subjects for the code, which in turn helped with any modifications and changes that needed to be made so both the hardware and software of the OBD communication tools worked as good as possible. No live testing was performed with the eye tracking software and we only had minor testing with bluetooth communication with the external sensors and did not have anything concrete [MAM]

The eye tracking software utilized was written in Python and ran/debugged through the Spyder IDE. The Webcam eyetracker is open source software from the developer and thus was able to be modified at will.[11] The vision algorithm currently operates by checking basic parameters continuously. Firstly, the code takes into account the light threshold and subsequently finds the potential pupil areas based on said threshold by looking for the darkest point in the initial rectangle (which is in the center of the screen). The GUI updates this rectangle (that can have its size adjusted) and pupil position continuously and corrects the rectangle edges that go beyond image boundaries. While looping through and updating the rectangle boundaries and checking the settings/parameters, the code will also check if the pupil leaves the rectangle at any time. As soon as the pupil is outside, the code will return invalid parameters for the pupil position and output to the console both the timestamp of when they are missing and the fact that they are missing (after two seconds). If, however, the pupils are ever found within two seconds after they went missing, the timer resets and the code goes back to checking if the pupil is found or not (eyes open or closed). [MAK]

As it stands, most of the code for this portion of the design is a bad representation of the end product. For debugging purposes, a GUI will help getting the code up and running with the other subsystems. However, there is no planned monitor of any kind to be present in the vehicle thus rendering a GUI useless. Additionally, there are three“stages” throughout the flow of the code: setting the light threshold, setting the rectangle parameters, and finally tracking and confirming that the tracking is working. The end product will ideally have everything configured on start up with no input from the driver, thus removing the need for multiple stages. This however does bring up the concern of dynamically setting both the light threshold and the rectangle dimensions on startup which will have to be addressed later in the overall eye tracking design. [MAK]

```
    def find_pupil(self, thresholded, timestart, pupilrect=True):

            #timenow=time.time()

            current = datetime.now() #added timestamping
            year = current.strftime("%Y")
            month = current.strftime("%m")
            day = current.strftime("%d")
            second = current.strftime("%H:%M:%S.%f")

            """Get the pupil center, bounds, and size, based on the thresholded
            image; please note that the pupil bounds and size are very
            arbitrary: they provide information on the pupil within the
            thresholded image, meaning that they would appear larger if the
            camera is placed closer towards a subject, even though the
            subject's pupil does not dilate

            arguments
            thresholded           --     a pygame.surface.Surface instance, as
                                         returned by threshold_image

            keyword arguments
            pupilrect             --     a Boolean indicating whether pupil searching
                                         rect should be applied or not
                                         (default = False)

            returns
            pupilcenter, pupilsize, pupilbounds
                                  --     pupilcenter is an (x,y) position tuple that
```

```
                                        gives the pupil center with regards to the
                                        image (where the top left is (0,0))
                                        pupilsize is the amount of pixels that are
                                        considered to be part of the pupil in the
                                        thresholded image; when no pupilbounds can
                                        be found, this will return (-1,-1)
                                        pupilbounds is a (x,y,width,height) tuple,
                                        specifying the size of the largest square
                                        in which the pupil would fit
                """


                # cut out pupilrect (but only if pupil bounding rect option is on)
                if pupilrect:
                        # pupil rect boundaries
                        rectbounds = pygame.Rect(self.settings['pupilrect'])
                        # correct rect edges that go beyond image boundaries
                        if self.settings['pupilrect'].left < 0:
                                rectbounds.left = 0
                        if self.settings['pupilrect'].right > self.camres[0]:
                                rectbounds.right = self.camres[0]
                        if self.settings['pupilrect'].top < 0:
                                rectbounds.top = 0
                        if self.settings['pupilrect'].bottom > self.camres[1]:
                                rectbounds.bottom = self.camres[1]
                        # cut rect out of image
                        thresholded = thresholded.subsurface(rectbounds)
                        ox, oy = thresholded.get_offset()

                # find potential pupil areas based on threshold
                th =
(self.settings['threshold'],self.settings['threshold'],self.settings['threshold'])
                mask = pygame.mask.from_threshold(thresholded, self.settings['pupilcol'], th)

                # get largest connected area within mask (which should be the pupil)
                pupil = mask.connected_component()

                # get pupil center
                pupilcenter = pupil.centroid()

                # if we can only look within a rect around the pupil, do so
                if pupilrect:
                        # compensate for subsurface offset
                        pupilcenter = pupilcenter[0]+ox, pupilcenter[1]+oy
                        # check if the pupil position is within the rect
                        if (self.settings['pupilrect'].left < pupilcenter[0] <
self.settings['pupilrect'].right) and (self.settings['pupilrect'].top < pupilcenter[1] <
self.settings['pupilrect'].bottom):
                                # set new pupil and rect position
                                self.settings['pupilpos'] = pupilcenter
                                x = pupilcenter[0] - self.settings['pupilrect'][2]/2
                                y = pupilcenter[1] - self.settings['pupilrect'][3]/2
                                self.settings['pupilrect'] =
pygame.Rect(x,y,self.settings['pupilrect'][2],self.settings['pupilrect'][3])
```

```
                        timestart = time.time() # record time now, set to timestart

                # if the pupil is outside of the rect, return missing
                else:
                        self.settings['pupilpos'] = (-1,-1) #set pupil position to -1, -1
                        end = time.time() # record time now
                        #print(end-timestart) # debugging purposes
                        #print("Missing!", year, month, day, second) #added timestamping
                        if ((end - timestart) > 2.0):
                                # if 2 seconds have elapsed since pupils went missing,
print missing and time stamp
                                print("Missing!", year, month, day, second)
        else:
                self.settings['pupilpos'] = pupilcenter

        # get pupil bounds (sometimes failes, hence try-except)
        try:
                self.settings['pupilbounds'] = pupil.get_bounding_rects()[0]
                # if we're using a pupil rect, compensate offset
                if pupilrect:
                        self.settings['pupilbounds'].left += ox
                        self.settings['pupilbounds'].top += oy
        except:
                # if it fails, we simply use the old rect
                pass

        return self.settings['pupilpos'], pupil.count(), self.settings['pupilbounds'],
timestart
```

[MAK]

After debugging throughout the final semester of design and implementation, several goals were achieved. The first of which being the GUI fixes. Originally, the GUI was in place to allow the tester to see themselves as they tested the algorithm and eye tracking of the subsystem. After the changes, the program fed information such as time stamps of infringements (eyes closed for too long) and the value of variables to the console instead of displaying them to an otherwise useless GUI, considering there was no plan to have a monitor feedback of the driver for the driver to look at themselves. This was one of two major goals that were planned before the code was moved to the Raspberry Pi. [MAK]

The next objectives were to figure out how to make the detection of light available in the environment dynamic and how to make the region where the pupil will start at the beginning of

the program dynamic. As stated above, the two things that the algorithm takes into account are the light threshold of the image and the region of interest (ROI) for the camera to track, here being the drivers' pupils. Both of these parameters were originally inputted by the driver into the program for the algorithm to work properly. The latter was made dynamic and without the need of the driver's input quite easily. Considering both the camera and the driver's headrest were to be static throughout the entirety of the trip, the ROI was set directly in the center of the image it was capturing and encompassed both eyes. Although drivers all set seat levels to different settings, this change was made keeping in mind that the system was to be as proper and safe as possible in terms of driving standards. This meant a seat at 90 degree angle plus or minus 10 degrees. [MAK]

```
       def threshold_image(self, image):

              """Applies a threshold to an image and returns the thresholded
              image

              arguments
              image                 --     the image that should be thresholded, a
                                           pygame.surface.Surface instance

              returns
              thresholded           --     the thresholded image,
                                           a pygame.surface.Surface instance
              """

              # surface to apply threshold to surface
              thimg = pygame.surface.Surface(self.get_size(), 0, image)

              # perform thresholding
              th =
(self.settings['threshold'],self.settings['threshold'],self.settings['threshold'])
              pygame.transform.threshold(thimg, image, self.settings['pupilcol'], th,
self.settings['nonthresholdcol'], 1)

              return thimg
```

[MAK]

Making the light threshold dynamic proved to be more of a challenge. The steps to make it work appropriately were not successful in the early stages of testing. First, the idea was to take the image of the driver and poll the average color of the whole image and base the light threshold on the inverse of this value, continuously. This proved to not work but the idea was still valid. The next test was to use just a ROI somewhere that wasn't the drivers' eyes on the image and base the light threshold on the average color of this surface. This was closer to the desired dynamic result, but still not quite right. Finally, the last trial was to set up a second webcam. This proved difficult to implement in the code considering that "duplicating" the sections of the code and just using a second input was not applicable. However, after some trial and error, the result was successful. The second camera was introduced with the past trials in mind; set the camera up in the car where it will take an average color of a ROI that accurately illustrates how much light exists in the vehicle, send this value to the other camera as a light threshold, and finally track the drivers' eyes with this value in mind. This was the desired dynamic result. [MAK]

The next step was to integrate the eye tracking subsystem into the main Raspberry Pi module. The Pi was to run all the Python programs in unison with the eye tracking subsystem feeding the main module with continuous and live data. As mentioned above, simple flags were being used to simulate whether the driver was being attentive or not before the subsystems were integrated together. [MAK]

```
	def find_pupil(self, thresholded, timestart, pupilrect=True):

		#timenow=time.time()

		current = datetime.now() #added timestamping
		year = current.strftime("%Y")
		month = current.strftime("%m")
		day = current.strftime("%d")
```

```
		second = current.strftime("%H:%M:%S.%f")

		"""Get the pupil center, bounds, and size, based on the thresholded
		image; please note that the pupil bounds and size are very
		arbitrary: they provide information on the pupil within the
		thresholded image, meaning that they would appear larger if the
		camera is placed closer towards a subject, even though the
		subject's pupil does not dilate

		arguments
		thresholded		--	a pygame.surface.Surface instance, as
						returned by threshold_image

		keyword arguments
		pupilrect		--	a Boolean indicating whether pupil searching
						rect should be applied or not
						(default = False)

		returns
		pupilcenter, pupilsize, pupilbounds
					--	pupilcenter is an (x,y) position tuple that
						gives the pupil center with regards to the
						image (where the top left is (0,0))
						pupilsize is the amount of pixels that are
						considered to be part of the pupil in the
						thresholded image; when no pupilbounds can
						be found, this will return (-1,-1)
						pupilbounds is a (x,y,width,height) tuple,
						specifying the size of the largest square
						in which the pupil would fit
		"""


		# cut out pupilrect (but only if pupil bounding rect option is on)
		if pupilrect:
			# pupil rect boundaries
			rectbounds = pygame.Rect(self.settings['pupilrect'])
			# correct rect edges that go beyond image boundaries
			if self.settings['pupilrect'].left < 0:
				rectbounds.left = 0
			if self.settings['pupilrect'].right > self.camres[0]:
				rectbounds.right = self.camres[0]
			if self.settings['pupilrect'].top < 0:
				rectbounds.top = 0
			if self.settings['pupilrect'].bottom > self.camres[1]:
				rectbounds.bottom = self.camres[1]
			# cut rect out of image
			thresholded = thresholded.subsurface(rectbounds)
			ox, oy = thresholded.get_offset()

		# find potential pupil areas based on threshold
		th =
(self.settings['threshold'],self.settings['threshold'],self.settings['threshold'])
		mask = pygame.mask.from_threshold(thresholded, self.settings['pupilcol'], th)
```

```
		# get largest connected area within mask (which should be the pupil)
		pupil = mask.connected_component()

		# get pupil center
		pupilcenter = pupil.centroid()

		# if we can only look within a rect around the pupil, do so
		if pupilrect:
			# compensate for subsurface offset
			pupilcenter = pupilcenter[0]+ox, pupilcenter[1]+oy
			# check if the pupil position is within the rect
			if (self.settings['pupilrect'].left < pupilcenter[0] <
self.settings['pupilrect'].right) and (self.settings['pupilrect'].top < pupilcenter[1] <
self.settings['pupilrect'].bottom):
				# set new pupil and rect position
				self.settings['pupilpos'] = pupilcenter
				x = pupilcenter[0] - self.settings['pupilrect'][2]/2
				y = pupilcenter[1] - self.settings['pupilrect'][3]/2
				self.settings['pupilrect'] =
pygame.Rect(x,y,self.settings['pupilrect'][2],self.settings['pupilrect'][3])
				timestart = time.time() # record time now, set to timestart

			# if the pupil is outside of the rect, return missing
			else:
				self.settings['pupilpos'] = (-1,-1) #set pupil position to -1, -1
				end = time.time() # record time now
				#print(end-timestart) # debugging purposes
				#print("Missing!", year, month, day, second) #added timestamping
				if ((end - timestart) > 2.0):
					# if 2 seconds have elapsed since pupils went missing,
print missing and time stamp
					print("Missing!", year, month, day, second)
		else:
			self.settings['pupilpos'] = pupilcenter

		# get pupil bounds (sometimes failes, hence try-except)
		try:
			self.settings['pupilbounds'] = pupil.get_bounding_rects()[0]
			# if we're using a pupil rect, compensate offset
			if pupilrect:
				self.settings['pupilbounds'].left += ox
				self.settings['pupilbounds'].top += oy
		except:
			# if it fails, we simply use the old rect
			pass

		return self.settings['pupilpos'], pupil.count(), self.settings['pupilbounds'],
timestart
```

[MAK]

Had there been more time, this section where the flag was being set after two seconds of missing pupils would have been read by the main module. Unfortunately, both subsystems remained isolated from each other and real trials were not run. [MAK]

The PIC24FJ is programmed in C using MPLAB X IDE from microcontroller. The code shown controls the microcontroller, takes analog input from the force sensitive resistors, gets gyroscope/accelerometer data and transmits UART communication to the RN4870 bluetooth module. [BC]

The config.h file configures the microcontroller with detailed comments in the code section shown. The main design consideration in this section is the clock speed which is configured to 16Mhz using the primary oscillator with phase lock loop enabled. [BC]

Two timer functions are utilized, a microsecond timer and a millisecond timer. Both use timer 1 on the device, but with different configurations and scalers. The microsecond delay uses a prescale of 8 and scales the integer input by 2. The millisecond delay uses a prescale of 256 and multiplies the integer input by 63. [BC]

The analog to digital converter is configured using the function InitADC. The ReadADC function reads the analog channel that is input to the function. In this code the analog channels 3 and 12 are used. [BC]

The I2C is initialized using the function I2Cinit. This enables the module and puts it in 7-bit address mode. The I2C requires functions for start condition, repeated start, get byte, send byte, get last byte, get data and stop condition. The gyroscope and accelerometer data is able to be taken using the I2Cgetdata function with input of the register number. This function uses a series of other functions to use I2C to communicate. It enters the start condition then sends the

device write slave address 0xD0. Then it sends the register number for the data wanted. Then it enters the repeated start in order to enter receive mode sending the read slave address 0xD1. It then gets two bytes, the hibyte and lobyte. Finally, it enters the stop conditions. [B.C]

The gyroscope is configured using the function GyroConfig which is a series of I2C transmissions to set up the device. The data taken from the gyroscope is used to determine if the device is in motion. [B.C.]

The UART is initialized using the function InitU2. The baud rate used is 115200. Using the equation:

$$UxBRG = \frac{FCY}{4 * Baud\ Rate} - 1$$

The value for UxBRG is calculated to be 34. The UART mode is set to enable UART2, have idle state of 1, 8 bit data, no parity, and 1 stop bit. The transmit is also enabled. The putU2 function writes each character to the UART TX register. The UART is put onto PIN50 using Peripheral Pin Select Output Register 8. RPOR8 is set to '0x0500' because the output function number for the selectable output source UART2 Transmit is 5. Within the main function to transmit a string, a buffer is created using the sprintf function and then a loop is used to send incrementing characters. The hexadecimal values 0x0A and 0x0D are sent to move to a new line and move to the beginning of the line respectively after each string is transmitted. The information being transmitted in this string is a simple incrementing counter value, and the unscaled input from the A/D converters. [BC]

```
/*
 Brian Call
 Senior Design Project
 'config.h' file for senior design steering wheel subsystem
 PIC24FJ1024GB610 Configuration Bit Settings
 'C' source line config statements
*/
```

```
// FSEC
#pragma config BWRP = OFF               // Boot Segment Write-Protect bit (Boot Segment may be
written)
#pragma config BSS = DISABLED           // Boot Segment Code-Protect Level bits (No Protection
(other than BWRP))
#pragma config BSEN = OFF               // Boot Segment Control bit (No Boot Segment)
#pragma config GWRP = OFF               // General Segment Write-Protect bit (General Segment
may be written)
#pragma config GSS = DISABLED           // General Segment Code-Protect Level bits (No
Protection (other than GWRP))
#pragma config CWRP = OFF               // Configuration Segment Write-Protect bit
(Configuration Segment may be written)
#pragma config CSS = DISABLED           // Configuration Segment Code-Protect Level bits (No
Protection (other than CWRP))
#pragma config AIVTDIS = OFF            // Alternate Interrupt Vector Table bit (Disabled
AIVT)

// FBSLIM
#pragma config BSLIM = 0x1FFF           // Boot Segment Flash Page Address Limit bits (Enter
Hexadecimal value)

// FOSCSEL
#pragma config FNOSC = PRIPLL           // Oscillator Source Selection (Primary Oscillator
with PLL module (XT + PLL, HS + PLL, EC + PLL))
#pragma config PLLMODE = PLL4X
#pragma config IESO = OFF               // Two-speed Oscillator Start-up Enable bit (Start up
with user-selected oscillator source)

// FOSC
#pragma config POSCMD = XT
#pragma config OSCIOFCN = OFF           // OSC2 Pin Function bit (OSC2 is clock output)
#pragma config SOSCSEL = ON             // SOSC Power Selection Configuration bits (SOSC is
used in crystal (SOSCI/SOSCO) mode)
#pragma config PLLSS = PLL_PRI
#pragma config IOL1WAY = ON
#pragma config FCKSM = CSDCMD           // Clock Switching Mode bits (Both Clock switching and
Fail-safe Clock Monitor are disabled)

// FWDT
#pragma config WDTPS = PS32768          // Watchdog Timer Postscaler bits (1:32,768)
#pragma config FWPSA = PR128            // Watchdog Timer Prescaler bit (1:128)
#pragma config FWDTEN = ON              // Watchdog Timer Enable bits (WDT Enabled)
#pragma config WINDIS = OFF             // Watchdog Timer Window Enable bit (Watchdog Timer in
Non-Window mode)
#pragma config WDTWIN = WIN25           // Watchdog Timer Window Select bits (WDT Window is
25% of WDT period)
#pragma config WDTCMX = WDTCLK          // WDT MUX Source Select bits (WDT clock source is
determined by the WDTCLK Configuration bits)
#pragma config WDTCLK = LPRC            // WDT Clock Source Select bits (WDT uses LPRC)

// FPOR
#pragma config BOREN = ON               // Brown Out Enable bit (Brown Out Enable Bit)
#pragma config LPCFG = OFF              // Low power regulator control (No Retention Sleep)
```

```
#pragma config DNVPEN = ENABLE          // Downside Voltage Protection Enable bit (Downside
protection enabled using ZPBOR when BOR is inactive)

// FICD
#pragma config ICS = PGD1               // ICD Communication Channel Select bits (Communicate
on PGEC1 and PGED1)
#pragma config JTAGEN = OFF             // JTAG Enable bit (JTAG is disabled)
#pragma config BTSWP = OFF              // BOOTSWP Disable (BOOTSWP instruction disabled)

// FDEVOPT1
#pragma config ALTCMPI = DISABLE        // Alternate Comparator Input Enable bit (C1INC,
C2INC, and C3INC are on their standard pin locations)
#pragma config TMPRPIN = OFF            // Tamper Pin Enable bit (TMPRN pin function is
disabled)
#pragma config SOSCHP = ON              // SOSC High Power Enable bit (valid only when SOSCSEL
= 1 (Enable SOSC high power mode (default))
#pragma config ALTVREF = ALTREFEN       // Alternate Voltage Reference Location Enable bit
(VREF+ and CVREF+ on RA10, VREF- and CVREF- on RA9)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>



/*****************************************************************************************/

/*
 Brian Call
 Senior Design Project
 Steering wheel subsystem
 */
#include <string.h>
#include <stdio.h>
#include "config.h"
#define RTS _RF13 // Output, For potential hardware handshaking.
#define CTS _RF12 // Input, For potential hardware handshaking.


int abs(int v)
{
    return v * ((v<0)*(-1) + (v>0));
}

void us_delay(int n)
{
    T1CON = 0x8010; //
    TMR1 = 0; // reset timer to 0
    while (TMR1 < n * 2);
}

void ms_delay(int n)
{
```

```
    T1CON = 0x8030;
    TMR1 = 0;
    while (TMR1 < n * 63);
}

void InitU2(void)
{
    U2BRG = 34;
    U2MODE = 0x8008; // See data sheet, pg 148. Enable UART2, BRGH = 1,
                     // Idle state = 1, 8 data, No parity, 1 Stop bit
    U2STA = 0x0400; // See data sheet, pg. 150, Transmit Enable
    // Following lines pertain Hardware handshaking
    TRISFbits.TRISF13 = 1; // enable RTS , output
    RTS = 1; // default status , not ready to send
}

char putU2(char c)
{
    while (U2STAbits.UTXBF); // Wait if transmit buffer full.
    U2TXREG = c; // Write value to transmit FIFO
    return c;
}

char getU2(void)
{
    RTS = 0; // telling the other side !RTS
    while (!U2STAbits . URXDA); // wait
    RTS = 1; // telling the other side RTS
    return U2RXREG; // from receiving buffer
} //getU2

void InitADC(void)
{
    AD1CON2bits.PVCFG = 0x0 ;
    AD1CON3bits.ADCS = 0xFF ;
    AD1CON1bits.SSRC = 0x0;
    AD1CON3bits.SAMC = 0b10000;
    AD1CON1bits.FORM = 0b00;
    AD1CON2bits.SMPI = 0x0;
    AD1CON1bits.ADON = 1;
} // InitADC

int ReadADC(int ch)
{
    uint16_t i;
    AD1CHS = ch;
    // Get an ADC sample
    AD1CON1bits.SAMP = 1;           //Start sampling
    for(i=0;i<1000;i++)
    {
        Nop(); //Sample delay, conversion start automatically
    }
    AD1CON1bits.SAMP = 0;           //Start sampling
    for(i=0;i<1000;i++)
```

```
    {
        Nop(); //Sample delay, conversion start automatically
    }
    while(!AD1CON1bits.DONE);       //Wait for conversion to complete
    return ADC1BUF0;
} // ReadADC

void I2Cinit(int BRG) {
    I2C1BRG = BRG; //See PIC24FJ128GA010 data sheet, Table 16.1 pg. 139
    while (I2C1STATbits.P); // Check buss idle, see 5.1 of I2C document.
    // It works, not sure its needed.
    I2C1CONLbits.A10M = 0; // 7-bit address mode (Added 8-14-17)
    I2C1CONLbits.I2CEN = 1; // enable module

}

void I2CStart(void) {
    us_delay(10); // delay to be safe
    I2C1CONLbits.SEN = 1; // Initiate Start condition see pg. 21 of I2C man. DS70000195F
    while (I2C1CONLbits.SEN); // wait for Start condition complete See sec. 5.1
    us_delay(10); // delay to be safe
}

void I2CRepeatedStart(void){
    us_delay(10);
    I2C1CONLbits.RSEN = 1;
    while(I2C1CONLbits.RSEN);
    us_delay(10);
}

void I2CStop(void) {
    us_delay(10); // delay to be safe
    I2C1CONLbits.PEN = 1; // see 5.5 pg. 27 of Microchip I2C manual DS70000195F
    while (I2C1CONLbits.PEN); // This is at hardware level, & I suspect fast.
    us_delay(10); // delay to be safe
}

void I2Csendbyte(char data) {
    while (I2C1STATbits.TBF); //wait if buffer is full
    I2C1TRN = data; // pass data to transmission register
    us_delay(10); // delay to be safe
}



char I2CLastgetbyte(void) {
    I2C1CONLbits.RCEN = 1; // Set RCEN, Enables I2C Receive mode
    while (!I2C1STATbits.RBF); //wait for byte to shift into I2C1RCV register
    I2C1CONLbits.ACKDT = 1;
    I2C1CONLbits.ACKEN = 1; // Master sends NACK
    us_delay(10); // delay to be safe
    return (I2C1RCV);
}
```

```
char I2Cgetbyte(void) {
    I2C1CONLbits.RCEN = 1; // Set RCEN, Enables I2C Receive mode
    while (!I2C1STATbits.RBF); //wait for byte to shift into I2C1RCV register
    I2C1CONLbits.ACKDT = 0;
    I2C1CONLbits.ACKEN = 1; // Master sends ACK
    us_delay(10); // delay to be safe
    return (I2C1RCV);
}

float I2Cgetdata(char reg_number){
    int hibyte = 0;
    int lobyte = 0;
    us_delay(100);
    I2CStart(); //puts I2C in start condition
    us_delay(100);
    I2Csendbyte(0xD0);  //Send the write slave device address
    us_delay(100);
    I2Csendbyte(reg_number);  // Register Map Address
    us_delay(100);
    I2CRepeatedStart();
    us_delay(100);
    I2Csendbyte(0xD1);  // Send the read slave device address
    us_delay(100);
    hibyte = I2Cgetbyte();
    us_delay(100);
    lobyte = I2CLastgetbyte();
    us_delay(100);
    I2CStop();
    us_delay(250);
    return (((hibyte << 8) + lobyte )/131.0);
}

void GyroConfig(){
    us_delay(100);
    I2CStart(); //puts I2C in start condition
    us_delay(100);
    I2Csendbyte(0xD0);  //Send the write slave device address
    us_delay(100);
    I2Csendbyte(0x1B);  // Gyro_Config Register
    us_delay(100);
    I2CRepeatedStart();
    us_delay(100);
    I2Csendbyte(0xD0);  // Send the write slave device address
    us_delay(100);
    I2Csendbyte(0x00);
    us_delay(100);
    I2CStop();
}

void main(void)
{
    InitU2();               // initialize UART
    int count;              // define variable
```

```
    int i;
    int FSR1;
    int FSR2;
    char str[40];
    char str1[40];
    int Gyro_X = 0;
    int Gyro_Y = 0;
    int Gyro_Z = 0;

    count = 0;              // sets count to start at 0
    TRISA = 0x00;           // sets LEDs as outputs
    TRISFbits.TRISF5 = 0;   // sets pin 50 as output
    RPOR8 = 0x0500;         // puts UART on Pin 50
    us_delay(500);

    InitADC();  // initialize Analog to digital
    I2Cinit(17); //initialize i2c. Find BRG from page 139 was 0x9D

    //GyroConfig();

    while(1)
    {

        FSR1 = ReadADC(3);        // Reads ADC 3 for FSR1
        FSR2 = ReadADC(12);       // Reads ADC 12 for FSR2
        //LATA = FSR2;              // Puts FSR2 value onto LEDs for testing

        sprintf(str, "Count = %d;  FSR1=%d;  FSR2=%d", count, FSR1, FSR2);
        size_t len_str = strlen(str);  // get length of string to send

        for (i = 0; i < len_str; i++)  // loop to write string
        {
            putU2(str[i]);
        }
        putU2(0x0A);    // new line
        putU2(0x0D);    // beginning of the line

        count = count + 1;      // increase counter

//I2C transmission

        Gyro_X = I2Cgetdata(0x43);  // 0x43
        Gyro_Y = I2Cgetdata(0x45);  // 0x45
        Gyro_Z = I2Cgetdata(0x47);  // 0x47

        sprintf(str1, "X= %d ; Y = %d ; Z = %d ", Gyro_X, Gyro_Y, Gyro_Z);
        size_t len_str1 = strlen(str1);  // get length of string to send
        for (i = 0; i < len_str1; i++)  // loop to write string
        {
            putU2(str1[i]);
        }
        putU2(0x0A);    // new line
        putU2(0x0D);    // beginning of the line
```

```
        if((abs(Gyro_X) + abs(Gyro_Y) + abs(Gyro_Z) > 200))
        {
            LATA = 0xFF;
        }
        else{
            LATA = 0x00;
        }

        ms_delay(250);
    }
}
```

[BC]

6. Mechanical Sketch

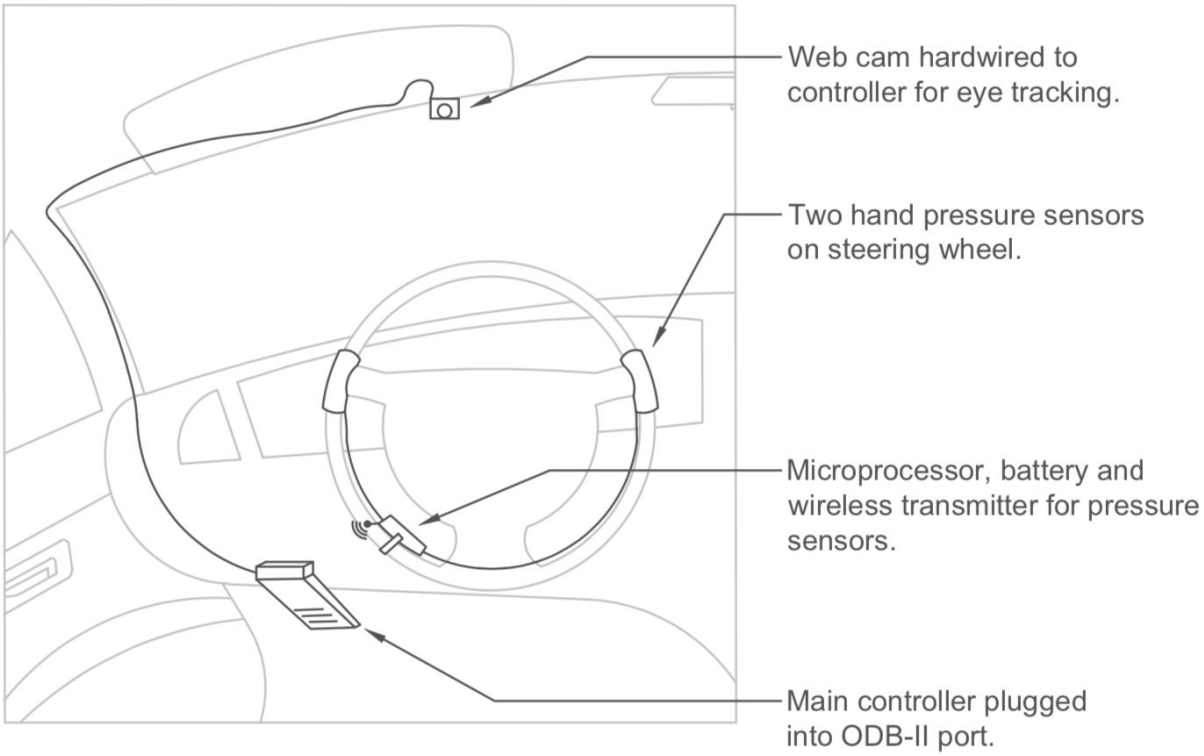

Figure 31. Mechanical Sketch. (BC, MK, MM, MDM)

7. Financial Budget

Table 25. Material Budget. (MDM)

			Unit	Total
Qty.	Part Num.	Description	Cost	Cost
1	3651	STN1110 Multiprotocol OBD-II to UART Interpreter IC SPDIP-28 package	$9.99	$9.99
1	CF14JT1K00CT-ND	RES 1K OHM 1/4W 5% AXIAL	0.10	0.10
1	CF14JT1K50CT-ND	RES 1.5K OHM 1/4W 5% AXIAL	0.10	0.10
2	CF14JT10K0CT-ND	RES 10K OHM 1/4W 5% AXIAL	0.10	0.20
1	CF14JT10R0CT-ND	RES 10 OHM 1/4W 5% AXIAL	0.10	0.10
2	CF14JT100RCT-ND	RES 100 OHM 1/4W 5% AXIAL	0.10	0.20
1	CF14JT4K70CT-ND	RES 4.7K OHM 1/4W 5% AXIAL	0.10	0.10
1	CF14JT100KCT-ND	RES 100K OHM 1/4W 5% AXIAL	0.10	0.10
1	490-16957-1-ND	CAP CER 10UF 50V X7S RADIAL	1.39	1.39
2	490-7360-1-ND	CAP CER 27PF 50V C0G/NP0 RADIAL	0.53	1.06
3	490-7517-1-ND	CAP CER 1UF 50V X7R RADIAL	0.83	2.49
2	490-8703-ND	CAP CER 560PF 50V NP0 RADIAL	0.48	0.96
1	493-17638-1-ND	CAP ALUM 100UF 10% 25V THRU HOLE	0.54	0.54
1	493-15293-ND	CAP ALUM 1000UF 20% 16V RADIAL	0.69	0.69
1	595-1728-ND	FIXED IND 100UH 7A 37 MOHM TH	2.44	2.44
1	497-11370-1-ND	DIODE SCHOTTKY 40V 3A DO201AD	0.47	0.47
1	B0087ZT5E2	5 x 16.000 MHz 16MHz Crystal HC-49S Low Profile	2.08	2.08
1	576-1518-5-ND	IC REG BUCK 5V 3A TO220-5	1.84	1.84
1	1568-1227-ND	OBD-II TO DB9 CABLE 10087	9.95	9.95
1	445-2525-1-ND	AUDIO PIEZO TRANSDUCER 30V TH	0.68	0.68
1		Raspberry Pi 3 B+	35.00	35.00
1	GF-HD-1080-AK	Full HD Webcam 1080P/1536P, Widescreen Video Calling	39.99	39.99
1	EMU-MK2	Freematics OBD-II Emulator MK2	289.90	289.90
1	MCP2551-I/P	IC TRANCIEVER HALF 1/1 8DIP	1.09	1.09
2	SF15-150	Force Sensitive Resistor	13.59	27.18
6	296-6607-1-ND	IC QUAD DIFF COMPARATOR 14-TSSOP	0.38	2.28
4	2N3904FS-ND	TRANS NPN 40V 0.2A TO-92	0.20	0.80
2	S510HCT-ND	RES 510 OHM 1/2W 5% CF MINI	0.10	0.20
3	CF14JT1K00CT-ND	RES 1K OHM 1/4W 5% AXIAL	0.10	0.30
13	CF14JT10K0CT-ND	RES 10K OHM 1/4W 5% AXIAL	0.10	1.30
2	LM317LDR2GOSCT-ND	IC REG LIN POS ADJ 100MA 8SOIC	0.37	0.74
1	2N7002NCT-ND	MOSFET N-CH 60V 115MA SOT-23	0.33	0.33
2	MMBT3904-FDICT-ND	TRANS NPN 40V 0.2A SMD SOT23-3	0.12	0.24
1	MMBT3906-FDICT-ND	TRANS PNP 40V 0.2A SOT23-3	0.12	0.12
3	1N4148FS-ND	DIODE GEN PURP 100V 200MA DO35	0.10	0.30
1	CMF374QFCT-ND	RES 374 OHM 1/4W 1% AXIAL	0.67	0.67

1	RNF14FTD866RCT-ND	RES 866 OHM 1/4W 1% AXIAL	0.10	0.10
1	CF14JT240RCT-ND	RES 240 OHM 1/4W 5% AXIAL	0.10	0.10
1	CF14JT2K20CT-ND	RES 2.2K OHM 1/4W 5% AXIAL	0.10	0.10
1	CF14JT2K00CT-ND	RES 2K OHM 1/4W 5% AXIAL	0.10	0.10
1	ALSR1J-8.0K-ND	RES 8K OHM 1W 5% AXIAL	1.86	1.86
1	RN4870-I/RM128	Microchip Bluetooth Low Energy Module	7.03	7.03
1	PIC24FJ64GA002-I/SS	28 PIN, 32/64KB FLASH 8KB RAM, 16 BIT CORE, GENERAL PURPOSE	2.83	2.83
2	RQ73C1E10KBTDF	10k thin film resistor type RQ73 Series	0.90	1.80
2	279-RQ73C1E113RBTDF	100 - 470 Ohm	0.90	1.80
6	C0603X104K5RAC3316	0.1uF 50V Ceramic Capacitor	0.19	1.14
1	DTP603450	1000mAh Rechargeable Battery	9.95	9.95
2	MCP73831T-2ADI/OT	Single Cell Li-ion Charge Management Controller	0.59	1.18
2	47589-0001	Micro USB Connector	0.77	1.54
4	C1206C475K5RACAUTO	4.7 uF 50 V ceramic Capacitor	0.47	1.88
2	RQ73C1E2K0BTDF	2k ohm Resistor	0.90	1.80
2	RQ73C1J475RBTDF	470 ohm Resistor	0.81	1.62
4	SML-LX0603SRW-TR	Red LED	0.39	1.56
1	MPU9250	Gyro and Accelerometer	8.40	8.40
1	1528-1597-ND	ADAPT USB A RCPT TO MICRO B PLUG	2.95	2.95
3	A879AR-ND	SOCKET ADAPTER SOT-23 TO 6DIP	2.38	7.14
4	A882AR-ND	SOCKET ADAPTER TSSOP TO 14DIP	3.03	12.12
2	A880AR-ND	SOCKET ADAPTER SOIC TO 8DIP	2.57	5.14
1	GF-HD-1080-AK	Full HD Webcam 1080P/1536P, Widescreen Video Calling	39.99	39.99
3	2N7002NCT-ND	MOSFET N-CH 60V 115MA SOT-23	0.33	0.99
2	A879AR-ND	SOCKET ADAPTER SOT-23 TO 6DIP	2.38	4.76
1	LM2576T-ADJ	Switching Voltage Regulators SIMPLE SWITCHER® 40V, 3A Low Component Count Step-Down Regulator 5-TO-220 -40 to 125	3.88	3.88
1	LM2576S-ADJ/NOPB	Switching Voltage Regulators 3A STEP-DOWN VLTG REG	3.33	3.33
1	573300D00000G	Heat Sinks Surface Mount Stamped Heatsink for D2Pak, TO-263 for D2PAK, TO-263, Horizontal Mounting, 16 n Thermal Resistance, Bulk Packaging, 26.16mm	0.73	0.73
1	2301843-1	D-Sub Standard Connectors AMPL PLUG HD20, R/A 9P, B/L,4-40 INS	1.91	1.91
3	RN73R1JTTD2001B25	Thin Film Resistors - SMD 2K ohm 0.1% 25 ppm	0.42	1.26
3	RQ73C1J6K49BTDF	Thin Film Resistors - SMD RQ 0603 6K49 0.1% 10PPM 1K RL	0.78	2.34
1	MSS1583-104KEB	Fixed Inductors MSS1583 SMT Power 0.103Ohms 100uH	2.43	2.43
1	EMZR100ARA102MHA0G	Aluminum Electrolytic Capacitors - SMD 1000uF 10V 20%	0.84	0.84
1	EEH-ZC1V101P	Aluminum Organic Polymer Capacitors 35VDC 100uF 20% AEC-Q200	1.78	1.78
3	MBRS340	Schottky Diodes & Rectifiers 3.0a Power Rectifier Schottky	0.45	1.35

			Total	**$573.65**

8. Team Information

Brian Call, Electrical Engineering, ESI (Y)

Matthew Krispinsky, Computer Engineering, ESI (Y)

Matt Marsek, Electrical Engineering, ESI (Y)

Matthew Mayfield, Computer Engineer, ESI (Y)

9. Parts List

Table 26. Parts list. (MDM)

Qty.	Refdes	Part Num.	Description
1	IC1	3651	STN1110 Multiprotocol OBD-II to UART Interpreter IC SPDIP-28 package
1	R3	CF14JT1K00CT-ND	RES 1K OHM 1/4W 5% AXIAL
1	R1	CF14JT1K50CT-ND	RES 1.5K OHM 1/4W 5% AXIAL
2		CF14JT10K0CT-ND	RES 10K OHM 1/4W 5% AXIAL
1	R1	CF14JT10R0CT-ND	RES 10 OHM 1/4W 5% AXIAL
2	R3, R4	CF14JT100RCT-ND	RES 100 OHM 1/4W 5% AXIAL
1	R2, R5	CF14JT4K70CT-ND	RES 4.7K OHM 1/4W 5% AXIAL
1	R2, R-S1	CF14JT100KCT-ND	RES 100K OHM 1/4W 5% AXIAL
1	C6	490-16957-1-ND	CAP CER 10UF 50V X7S RADIAL
2	C3, C4	490-7360-1-ND	CAP CER 27PF 50V C0G/NP0 RADIAL
3	C5, C7, C10, C11	490-7517-1-ND	CAP CER 1UF 50V X7R RADIAL
2	C8, C9	490-8703-ND	CAP CER 560PF 50V NP0 RADIAL
1		493-17638-1-ND	CAP ALUM 100UF 10% 25V THRU HOLE
1		493-15293-ND	CAP ALUM 1000UF 20% 16V RADIAL
1		595-1728-ND	FIXED IND 100UH 7A 37 MOHM TH
1		497-11370-1-ND	DIODE SCHOTTKY 40V 3A DO201AD
1	Y1	B0087ZT5E2	5 x 16.000 MHz 16MHz Crystal HC-49S Low Profile
1	IC1	576-1518-5-ND	IC REG BUCK 5V 3A TO220-5
1	P1	1568-1227-ND	OBD-II TO DB9 CABLE 10087
1		445-2525-1-ND	AUDIO PIEZO TRANSDUCER 30V TH
1			Raspberry Pi 3 B+
1	CAM1	GF-HD-1080-AK	Full HD Webcam 1080P/1536P, Widescreen Video Calling
1		EMU-MK2	Freematics OBD-II Emulator MK2
1	IC3	MCP2551-I/P	IC TRANCIEVER HALF 1/1 8DIP
2	FSR1, FRS2	SF15-150	Force Sensitive Resistor
6	IC4A, IC4B, IC4C, IC4P	296-6607-1-ND	IC QUAD DIFF COMPARATOR 14-TSSOP
4	Q1, Q2	2N3904FS-ND	TRANS NPN 40V 0.2A TO-92
2	R7, R9	S510HCT-ND	RES 510 OHM 1/2W 5% CF MINI
3	R6, R8, R17, R19	CF14JT1K00CT-ND	RES 1K OHM 1/4W 5% AXIAL

13	R10, R11, R12, R13, R18, R20, R23, R24, R25, R26	CF14JT10K0CT-ND	RES 10K OHM 1/4W 5% AXIAL
2	IC5	LM317LDR2GOSCT-ND	IC REG LIN POS ADJ 100MA 8SOIC
1	Q3	2N7002NCT-ND	MOSFET N-CH 60V 115MA SOT-23
2	Q4, Q6	MMBT3904-FDICT-ND	TRANS NPN 40V 0.2A SMD SOT23-3
1	Q5	MMBT3906-FDICT-ND	TRANS PNP 40V 0.2A SOT23-3
3	D2, D3, D4	1N4148FS-ND	DIODE GEN PURP 100V 200MA DO35
1	R16	CMF374QFCT-ND	RES 374 OHM 1/4W 1% AXIAL
1	R15	RNF14FTD866RCT-ND	RES 866 OHM 1/4W 1% AXIAL
1	R14	CF14JT240RCT-ND	RES 240 OHM 1/4W 5% AXIAL
1	R20	CF14JT2K20CT-ND	RES 2.2K OHM 1/4W 5% AXIAL
1	R21	CF14JT2K00CT-ND	RES 2K OHM 1/4W 5% AXIAL
1	R22	ALSR1J-8.0K-ND	RES 8K OHM 1W 5% AXIAL
1	BLE-MOD	RN4870-I/RM128	Microchip Bluetooth Low Energy Module
1	PIC24	PIC24FJ64GA002-I/SS	28 PIN, 32/64KB FLASH 8KB RAM, 16 BIT CORE, GENERAL PURPOSE
2	R-A1	RQ73C1E10KBTDF	10k thin film resistor type RQ73 Series
2	R-A2	279-RQ73C1E113RBTDF	100 - 470 Ohm
6	CA-1	C0603X104K5RAC3316	0.1uF 50V Ceramic Capacitor
1	BATT	DTP603450	1000mAh Rechargeable Battery
2		MCP73831T-2ADI/OT	Single Cell Li-ion Charge Management Controller
2		47589-0001	Micro USB Connector
4	C-B1	C1206C475K5RACAUTO	4.7 uF 50 V ceramic Capacitor
2	R-B1	RQ73C1E2K0BTDF	2k ohm Resistor
2	R-B2	RQ73C1J475RBTDF	470 ohm Resistor
4	LED-B1	SML-LX0603SRW-TR	Red LED
1	G1	MPU9250	Gyro and Accelerometer
1		1528-1597-ND	ADAPT USB A RCPT TO MICRO B PLUG
3		A879AR-ND	SOCKET ADAPTER SOT-23 TO 6DIP
4		A882AR-ND	SOCKET ADAPTER TSSOP TO 14DIP
2		A880AR-ND	SOCKET ADAPTER SOIC TO 8DIP
1	CAM2	GF-HD-1080-AK	Full HD Webcam 1080P/1536P, Widescreen Video Calling
3		2N7002NCT-ND	MOSFET N-CH 60V 115MA SOT-23
2		A879AR-ND	SOCKET ADAPTER SOT-23 TO 6DIP
1		LM2576T-ADJ	Switching Voltage Regulators SIMPLE SWITCHER® 40V, 3A Low Component Count Step-Down Regulator 5-TO-220 -40 to 125
1		LM2576S-ADJ/NOPB	Switching Voltage Regulators 3A STEP-DOWN VLTG REG
1		573300D00000G	Heat Sinks Surface Mount Stamped Heatsink for D2Pak, TO-263 for D2PAK, TO-263, Horizontal Mounting, 16 n

			Thermal Resistance, Bulk Packaging, 26.16mm
1	X1	2301843-1	D-Sub Standard Connectors AMPL PLUG HD20, R/A 9P, B/L,4-40 INS
3		RN73R1JTTD2001B25	Thin Film Resistors - SMD 2K ohm 0.1% 25 ppm
3		RQ73C1J6K49BTDF	Thin Film Resistors - SMD RQ 0603 6K49 0.1% 10PPM 1K RL
1	L1	MSS1583-104KEB	Fixed Inductors MSS1583 SMT Power 0.103Ohms 100uH
1	C1	EMZR100ARA102MHA0G	Aluminum Electrolytic Capacitors - SMD 1000uF 10V 20%
1	C2	EEH-ZC1V101P	Aluminum Organic Polymer Capacitors 35VDC 100uF 20% AEC-Q200
3	D1	MBRS340	Schottky Diodes & Rectifiers 3.0a Power Rectifier Schottky

10. Project Schedules

Table 27. Final Design Gantt Chart Resources and Project Schedule. (MAK)

Task Name	Duration	Start	Finish	Predecessors	Resource Names
SDPII Implementation 2020	**103 days**	**Mon 1/13/20**	**Fri 4/24/20**		
Revise Gantt Chart	14 days	**Mon 1/13/20**	Sun 1/26/20		Matthew Krispinsky
Implement Project Design	**96 days**	**Mon 1/13/20**	**Fri 4/17/20**		
Hardware Implementation	**49 days**	**Mon 1/13/20**	**Sun 3/1/20**		
Breadboard Components	**14 days**	**Mon 1/13/20**	**Sun 1/26/20**		
Accelerometer	14 days	Mon 1/13/20	Sun 1/26/20		Brian Call
Battery/Charging	14 days	Fri 2/14/20	Thu 2/27/20		Brian Call
Transceivers	14 days	Mon 1/13/20	Sun 1/26/20		Matt Marsek
Layout and Generate PCB(s)	**14 days**	**Mon 2/10/20**	**Sun 2/23/20**	**6**	
Steering Wheel	**14 days**	**Mon 2/10/20**	**Sun 2/23/20**		
Layout and PIN Assignment	14 days	Fri 2/14/20	Thu 2/27/20		Brian Call
Eagle PCB Design	14 days	Fri 2/28/20	Thu 3/12/20		Brian Call
OBD2 to UART	14 days	Mon 2/10/20	Sun 2/23/20		Matt Marsek
Voltage Regulator	14 days	Mon 2/24/20	Sun 3/8/20		Matt Marsek
Assemble Hardware	7 days	Mon 2/24/20	Sun 3/1/20	10	Brian Call,Matt Marsek
Test Hardware	7 days	Mon 3/2/20	Sun 3/8/20	15	Brian Call,Matt Marsek
Revise Hardware	7 days	Mon 3/2/20	Sun 3/8/20	15	Brian Call,Matt Marsek
MIDTERM: Demonstrate Hardware	5 days	Mon 3/9/20	Fri 3/13/20	16	Brian Call,Matt Marsek
SDC & FA Hardware Approval	0 days	Fri 2/28/20	Fri 2/28/20		
Software Implementation	**49 days**	**Mon 1/13/20**	**Sun 3/1/20**	**19**	
Develop Software	**28 days**	**Mon 1/13/20**	**Sun 2/9/20**		
Develop Vision Software	**28 days**	**Mon 1/13/20**	**Sun 2/9/20**		
Remove GUI Elements	14 days	Mon 1/13/20	Sun 1/26/20		Matthew Krispinsky
Make Light Threshold Dynamic	14 days	Mon 1/27/20	Sun 2/9/20		Matthew Krispinsky
Develop Accelerometer Software	**14 days**	**Sun 1/26/20**	**Sat 2/8/20**		
I2C Communication	14 days	Thu 1/30/20	Wed 2/12/20		Brian Call
Gyro/Accelerometer Data Processing	14 days	Wed 2/12/20	Tue 2/25/20		Brian Call
Develop Raspberry Pi Software	**28 days**	**Mon 1/13/20**	**Sun 2/9/20**		
Establish Simulatious Bluetooth and UART communication	7 days	Tue 2/4/20	Mon 2/10/20		Matthew Mayfield
Complete communication code	7 days	Tue 2/4/20	Mon 2/10/20		Matthew Mayfield
Establish Connection from Camera to Pi					Matthew Krispinsky
Test Software	16 days	Mon 2/10/20	Tue 2/25/20	22	Brian Call,Matthew Krispinsky,Matthew Mayfield
Revise Software	14 days	Mon 2/10/20	Sun 2/23/20	21	Brian Call,Matthew Krispinsky,Matthew Mayfield
MIDTERM: Demonstrate Software	5 days	Mon 2/24/20	Fri 2/28/20	33	Brian Call,Matthew Krispinsky,Matthew Mayfield
SDC & FA Software Approval	0 days	Fri 2/28/20	Fri 2/28/20		
System Integration	**50 days**	**Sat 2/29/20**	**Sat 4/18/20**		
Assemble Complete System	14 days	Sat 2/29/20	Fri 3/13/20	34	
Test Complete System	21 days	Sat 3/21/20	Fri 4/10/20	37	
Revise Complete System	21 days	Sat 3/21/20	Fri 4/10/20	37	
Demonstration of Complete System	7 days	Sat 4/11/20	Fri 4/17/20	39	Brian Call,Matt Marsek,Matthew Krispinsky,Matthew Mayfield
Develop Final Report	**103 days**	**Mon 1/13/20**	**Fri 4/24/20**		
Write Final Report	103 days	Mon 1/13/20	Fri 4/24/20		Brian Call,Matt Marsek,Matthew Krispinsky,Matthew Mayfield
Submit Final Report	0 days	Fri 4/24/20	Fri 4/24/20	42	Brian Call,Matt Marsek,Matthew Krispinsky,Matthew Mayfield
Spring Recess	7 days	Mon 3/23/20	Sun 3/29/20		Matt Marsek,Brian Call,Matthew Krispinsky,Matthew Mayfield
Project Demonstration and Presentation	0 days	Mon 4/20/20	Mon 4/20/20		Brian Call,Matt Marsek,Matthew Krispinsky,Matthew Mayfield

11. Conclusions and Recommendations

The design of this project focused on an easy to use device that improves driver safety while operating a motor vehicle. The technologies utilized in this project include Raspberry Pi computing, embedded systems, video processing, eye tracking, data storage, analog sensors, CAN communications and others. The main control unit is a Raspberry Pi computer. This device handles the main computations, video processing, UART inputs, bluetooth inputs, and auditory outputs. The signal from the OBD-II port in CAN standard, ISO standard, or J1850 standard is converted to UART. The bluetooth is wireless paired to the bluetooth on the steering wheel hand sensor subsystem. This subsystem uses an embedded PIC24FJ microcontroller to analyze analog input and transmit results via bluetooth.

As a team, we had disagreements of certain implementations or ideas, but as a whole we came together and finalized what the direction our finalized product and design would have been. The overall dynamics of the teams and our specified roles stayed relatively smooth and consistent throughout the year, with no real issues from any member on any front. Everyone performed their roles while also helping each other when needed.

The coding portion of the project, while unfinished in certain parts and aspects, we began to see the greater potential that the results of our efforts were bringing, in terms of reliability, stability, and usability. For the hardware side, the physical prototypes were fully functional with minor hiccups, but were not in a finalized state due to not having used PCBs due to current events. What was achieved in the time that was left was remarkably good, for the most part. There was almost a fully operational prototype with all subsystems being properly integrated. The future of the project and the work we would have done is unknown at this point, some

members have shown some interest in continuing the project as a side hobby, but as a whole, the project will probably never be in a “consumer level of completeness”.

Recommendations for future students for projects like this one presented in this paper are as follows: One, do not be afraid to try ideas outside of the box, you will never know what you could accomplish if you never try them. Two, keep the number of features and ideas in a reasonable amount due to the time restrictions you have for a Senior project. Three, it is not a race to the top or a talent show, take your time and work as hard as you can towards your goal, but do not be deterred by other people/groups and their projects. Lastly, have fun with it while being safe, it is a senior project in a lab environment with fellow classmates, don't stress yourself too much if something does not work properly.

Recommendations for future design of this device include additional analysis of vehicle information, additional improvements to power efficiency, and additional reliability across nonideal conditions of eye tracking. The algorithms used to define an ‘attentive’ driver will be able to be improved over time by tracking data. With this additional information could be taken from the onboard computer system via the OBD-II port and added to the algorithm. The steering wheel subsystem is battery powered making any improvements to power consumption add to the amount of time able to be used between charging. The improvements could be made in both the embedded programming utilizing sleep and low power modes, or in the bluetooth connection and number of communications. Having accurate and efficient eye tracking can depend on light levels and position of the user. Additional improvements can be made in the camera and dynamic range of this system.

12. References

[1] EyeTechDS. “EyeTech Eye Tracking Technology for Automotive Distracted and Distracted Driving Story by Channel 3”. Filmed [January 2016]. Youtube video, 2:10. Posted [January 2016]. https://www.youtube.com/watch?v=_aBZlQaIvmY.

[2] Ungvarsky, Janine. 2017. “Eye Tracking.” Salem Press Encyclopedia. http://ezproxy.uakron.edu:2048/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=ers&AN=125600217&site=eds-live.

[3] Valentin Amortila, Elena Mereuta, Silvia Veresiu, Madalina Rus, Costel. “HumelnicuPositioning study of driver's hands in certain areas of the steering wheel.” MATEC Web Conf 178, (2018). doi: 10.1051/matecconf/201817806014.

[4] O. Stan, L. Miclea and A. Centea, "Eye-Gaze Tracking Method Driven by Raspberry PI Applicable in Automotive Traffic Safety," *2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation*, Madrid, 2014, pp. 126-130.

[5] A. Mitrović, S. Djukanović and M. Radonjić, "Implementation of signal classification using frequency spectrum features on the raspberry Pi platform," 2017 25th Telecommunication Forum (TELFOR), Belgrade, 2017, pp. 1-4.

[6] A Barón, P Green, "Safety and Usability of Speech Interfaces for In-Vehicle Tasks while Driving: A Brief Literature Review", *Technical Report 2006-5*, 2006.

[7] Rahim, H., Yusop, Z., Bin Syed Hassan, S. and Chia Kim Seng (2010). Grasp hand approach to detect the attentiveness and fatigue of driver via vibration system. *2010 6th International Colloquium on Signal Processing & its Applications*.

[8] Lisseman, J., Mogg, T (2014). *United States Patent No. US 8725230B2*. Retrieved from https://patents.google.com/patent/US8725230

[9] Perme, T. (2007). Introduction to capacitive sensing. *Microchip Technology Inc*.

[10] “Does My Car Have OBD-II?” *The OBD II Home Page,* B&B Electronics, 2011, http://www.obdii.com/connector.html.

[11] Open source Python software for webcam eyetracker. Copyright (C) 2014 Edwin S. Dalmaijer. https://github.com/esdalmaijer/webcam-eyetracker

13. Appendices

```
# CamTracker GUI test

from camtracker import Setup
from camtracker import CamEyeTracker
from camtracker import available_devices
#import time

# initialize setup
setup = Setup()
eyetrack = CamEyeTracker()

available = available_devices()
print(available)

# run GUI
tracker = setup.start()

# in DEBUG mode, images of the calibration are saved as strings in a textfile,
# after the calibration, the textfile will be read and PNG images will be
# produced based on the content of the file
DEBUG = False
BUFFSEP = 'edwinisdebeste'


# # # # #
# imports

import os.path
import time
from datetime import datetime

# Try to import VideoCapture Library
# Requires VideoCapture & PIL libraries
vcAvailable = False
import imp
try:
  imp.find_module('VideoCapture')
  vcAvailable = True
  import VideoCapture
except ImportError:
  print "VideoCapture module not available"

try:
       import pygame
       import pygame.camera
       pygame.init()
       pygame.camera.init()
except:
       raise Exception("Error in camtracker: PyGame could not be imported and initialized!
:(")
```

```
# # # # #
# functions

def available_devices():

       """Returns a list of available device names or numbers; each name or
       number can be used to pass as Setup's device keyword argument

       arguments
       None

       keyword arguments
       None

       returns
       devlist       --     a list of device names or numbers, e.g.
                                   ['/dev/video0','/dev/video1'] or [0,1]
       """

       return pygame.camera.list_cameras()


# # ###
# classes

class Setup:

       """The Setup class provides means to calibrate your webcam to function
       as an eye tracker"""

       def __init__(self, device=None, camres=(640,480), disptype='window',
dispres=(1024,768), display=None):

              """Initializes a Setup instance

              arguments
              None

              keyword arguments
              device        --     a string or an integer, indicating either
                                   device name (e.g. '/dev/video0'), or a
                                   device number (e.g. 0); None can be passed
                                   too, in this case Setup will autodetect a
                                   useable device (default = None)
              camres        --     the resolution of the webcam, e.g.
                                   (640,480) (default = (640,480))
              disptype             --     a string indicating what kind of
                                   calibration display should be presented;
                                   choose from 'window' (PyGame windowed),
                                   'fullscreen' (PyGame fullscreen)
                                   (default = 'window')
              dispres       --     the resolution of the display, e.g.
```

```
                                        (1280,1024) (default = 1024,768)
                display         --      pass None to let the Setup create its own
                                        display, otherwise pass a display that
                                        matches the disptype you provided (under
                                        the disptype argument) to let the Setup use
                                        that display; example: set disptype to
                                        'fullscreen', then pass a
                                        pygame.surface.Surface instance that is
                                        returned by pygame.display.set_mode:
                                        calibration will then be presented on the
                                        passed pygame.surface.Surface instance
                """

                # DEBUG #
                if DEBUG:
                        self.savefile = open('savefile.txt','w')
                # # # # #

                # create new Display if none was passed, or use the provided display
                if display == None:
                        if disptype == 'window':
                                self.disp = pygame.display.set_mode(dispres, pygame.RESIZABLE)
                        elif disptype == 'fullscreen':
                                self.disp = pygame.display.set_mode(dispres,
pygame.FULLSCREEN|pygame.HWSURFACE|pygame.DOUBLEBUF)
                        else:
                                raise Exception("Error in camtracker.Setup.__init__: disptype
'%s' was not recognized; please use 'window', 'fullscreen'")
                # if a display was specified, use that
                else:
                        self.disp = display
                        dispres = self.disp.get_size()

                # select a device if none was selected
                if device == None:
                        available = available_devices()
                        if available == []:
                                raise Exception("Error in camtracker.Setup.__init__: no available
camera devices found (did you forget to plug it in?)")
                        else:
                                device = available[0]

                # create new camera
                self.tracker = CamEyeTracker(device=device, camres=camres)

                # find font: first look in directory, if that fails we fall back to default
                try:
                        fontname =
os.path.join(os.path.split(os.path.abspath(__file__))[0],'resources','roboto_regular-webfont.t
tf')
                except:
                        fontname = pygame.font.get_default_font()
                        print("WARNING: camtracker.Setup.__init__: could not find
'roboto_regular-webfont.ttf' in the resources directory!")
```

```
                # create a Font instance
                self.font = pygame.font.Font(fontname, 24)
                self.sfont = pygame.font.Font(fontname, 12)

                # set some properties
                self.disptype = disptype
                self.dispsize = dispres
                self.fgc = (255,255,255)
                self.bgc = (0,0,0)

                # fill display with background colour
                self.disp.fill(self.bgc)

                # set some more properties
                self.img = pygame.surface.Surface(self.tracker.get_size())  # empty surface,
gets filled out with camera images
                self.settings = {'pupilcol':(0,0,0), \
                                        'threshold':100, \
                                        'nonthresholdcol':(100,100,255,255), \
                                        'pupilpos': (camres[0]/2,camres[1]/2), \

'pupilrect':pygame.Rect(camres[0]/2-50,camres[1]/2-25,100,50), \
                                        'pupilbounds': [0,0,0,0], \
                                        '':None
                                        }


        def start(self):

                """Starts running the GUI

                arguments
                None

                keyword arguments
                None

                returns
                None
                """

                # show welcoming screen (loading...)
                self.show_welcome(loading=True)

                # DEBUG #
                if DEBUG:
                        self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
                # # # # #

                # create GUI
                self.setup_GUI()

                # replace 'loading' on welcoming screen with 'press any key to start'
                self.show_welcome(loading=False)
```

```
		# DEBUG #
		if DEBUG:
			self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
		# # # # #

		# wait for keypress
		noinput = True
		while noinput:
			for event in pygame.event.get():
				if event.type == pygame.KEYDOWN:
					noinput = False

		# show welcoming screen (and we're loading again)
		self.show_welcome(loading=True)

		# DEBUG #
		if DEBUG:
			self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
		# # # # #

		# draw general GUI (no stage information yet)
		self.draw_stage(stagenr=None)

		# DEBUG #
		if DEBUG:
			self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
		# # # # #

		# mouse visibility
		pygame.mouse.set_visible(True)

		# start setup (should return a CamEyeTracker instance)
		tracker = self.run_GUI()

		return tracker


	def show_welcome(self, loading=False):

		"""Shows a welcoming screen with package and author information,
		either depecting "Loading, please wait..." or "Press any key to
		start", depending on the loading argument

		arguments
		None

		keyword arguments
		loading		--	Boolean indicating whether welcoming screen
					should say "Loading, please wait..." or
					"Press any key to start"
		returns
		None			--	directly draws on self.disp
		"""
```

```
		# welcome text
		welcometext = \
		"""

		"""

		"""Welcome to the Webcam EyeTracker calibration interface!

		author: Edwin Dalmaijer
		version: 0.1 (12-10-2013)



		"""

		# reset display
		self.disp.fill(self.bgc)

		# loading message
		if loading:
			welcometext += "Loading, please wait..."
		else:
			welcometext += "Press any key to start!"

		# remove tabs from text
		welcometext = welcometext.replace("\t","")

		# draw lines on display
		x = self.dispsize[0]/2; y = self.dispsize[0]/2
		lines = welcometext.split("\n")
		nlines = len(lines)
		for lnr in range(nlines):
			# render text
			linesize = self.font.size(lines[lnr])
			rendered = self.font.render(lines[lnr], True, self.fgc) #
Font.render(text, antialias, color, background=None)
			# position
			pos = (x-linesize[0]/2, y + (lnr - nlines/2)*linesize[1])
			# draw to disp
			self.disp.blit(rendered, pos)

		# update!
		pygame.display.flip()


	def setup_GUI(self):

		"""Sets up a GUI interface within a PyGame Surface

		arguments
		None

		keyword arguments
```

```
                None

                returns
                None                    --      returns nothing, but draws on self.disp and
                                                sets self.guisurface
                """

                # directory
                resdir = os.path.join(os.path.split(os.path.abspath(__file__))[0],'resources')
                if not os.path.exists(resdir):
                        raise Exception("Error in camtracker.Setup.setup_GUI: could not find
'resources' directory to access button images; was it relocated or renamed, or is the
installation of camtracker incorrect?")

                # find image paths
                imgpaths = {}
                buttnames = ['1','2','3','up','down','t','space','r','escape']
                buttstates = ['active','inactive']
                for bn in buttnames:
                        imgpaths[bn] = {}
                        for bs in buttstates:
                                filename = "%s_%s.png" % (bn,bs)
                                imgpaths[bn][bs] = os.path.join(resdir,filename)
                                if not os.path.isfile(imgpaths[bn][bs]):
                                        print("WARNING: image file '%s' was not found in
resources!" % filename)
                                        imgpaths[bn][bs] = os.path.join(resdir,"blank_%s.png" %
bs)

                # image positions (image CENTERS!)
                buttsize = (50,50)
                camres = self.tracker.get_size()
                buttpos = {}
                y = self.dispsize[1]/2 + int(camres[1]*0.6)
                buttpos['1'] = int(self.dispsize[0]*(2/6.0) - buttsize[0]/2), y
                buttpos['2'] = int(self.dispsize[0]*(3/6.0) - buttsize[0]/2), y
                buttpos['3'] = int(self.dispsize[0]*(4/6.0) - buttsize[0]/2), y
                buttpos['space'] = int(self.dispsize[0]*(5/6.0) - buttsize[0]/2), y

                leftx = self.dispsize[0]/2 - (camres[0]/2 + buttsize[0]) # center of the
buttons on the right
                rightx = self.dispsize[0]/2 + camres[0]/2 + buttsize[0] # center of the buttons
on the left
                buttpos['up'] = rightx, self.dispsize[1]/2-buttsize[1] # above snapshot half,
to the right
                buttpos['down'] = rightx, self.dispsize[1]/2+buttsize[1] # below snapshot half,
to the right
                buttpos['t'] = leftx, self.dispsize[1]/2+camres[1]/2-buttsize[1]/2 # same level
as snapshot bottom, to the left
                buttpos['r'] = leftx, self.dispsize[1]/2 # halfway snapshot (==halfway
display), to the left
                buttpos['escape'] = buttsize[0], buttsize[1] # top left

                # new dict for button properties (image, position, and rect)
```

```
		self.buttons = {}
		# loop through button names
		for bn in imgpaths.keys():
			# new dict for this button name
			self.buttons[bn] = {}
			# recalculate position
			buttpos[bn] = buttpos[bn][0]-buttsize[0]/2, buttpos[bn][1]-buttsize[1]/2
			# loop through button states
			for bs in imgpaths[bn].keys():
				# new dict for this button name and this button state
				self.buttons[bn][bs] = {}
				# load button image
				self.buttons[bn][bs]['img'] = pygame.image.load(imgpaths[bn][bs])
				# save position and rect
				self.buttons[bn][bs]['pos'] = buttpos[bn]
				self.buttons[bn][bs]['rect'] = buttpos[bn][0], buttpos[bn][1],
buttsize[0], buttsize[1]

		# save buttsize
		self.buttsize = buttsize


	def draw_button(self, image, pos):

		"""Draws a button on the display

		arguments
		image		--	a pygame.surface.Surface instance, depicting
					a button
		pos		--	a (x,y) position coordinate, indicating the
					top left corner of the button

		keyword arguments
		None

		returns
		None		--	directly draws on self.disp
		"""

		self.disp.blit(image, pos)


	def draw_stage(self, stagenr=None):

		"""Draws the GUI window for the passed stage nr

		arguments
		None

		keyword arguments
		stagenr	--	None for only the basic buttons, or a stage
					number for the basic buttons, as well as the
					stage specific buttons
```

```
		returns
		None			--	directly draws on self.disp
		"""

		# clear display
		self.disp.fill(self.bgc)

		# universal buttons
		buttonstodraw = ['1','2','3','space','escape','t','r']
		activetodraw = []

		# stage specific buttons
		if stagenr == 1:
			title = "set pupil detection threshold"
			buttonstodraw.extend(['up','down'])
			activetodraw.extend(['1'])
		elif stagenr == 2:
			title = "select pupil and set pupil detection bounds"
			buttonstodraw.extend(['up','down'])
			activetodraw.extend(['2'])
		elif stagenr == 3:
			title = "confirmation"
			buttonstodraw.extend(['up','down'])
			activetodraw.extend(['3'])
		else:
			title = "loading, please wait..."

		# draw inactive buttons
		for buttname in buttonstodraw:

self.draw_button(self.buttons[buttname]['inactive']['img'],self.buttons[buttname]['inactive']['
'pos'])

		# draw active buttons
		for buttname in activetodraw:

self.draw_button(self.buttons[buttname]['active']['img'],self.buttons[buttname]['active']['pos
'])

		# draw title
		titsize = self.font.size(title) # author note: LOL, 'titsize'!
		titpos = self.dispsize[0]/2-titsize[0]/2,
self.dispsize[1]/2-(self.tracker.get_size()[1]/2+titsize[1])
		titsurf = self.font.render(title, True, self.fgc)
		self.disp.blit(titsurf,titpos)


	def run_GUI(self):

		"""Perform a setup to set all settings using a GUI

		arguments
		None
```

```
                keyword arguments
                None

                returns
                None                    --      returns nothing, but does fill in
                                                the self.settings dict
                """
                start=time.time() #starting time

                # # # # #
                # variables

                # general
                stage = 1 # stage is updated by handle_input functions

                # stage specific
                stagevars = {}

                stagevars[0] = {}
                stagevars[0]['show_threshimg'] = False # False for showing snapshots, True for
showing thresholded snapshots
                stagevars[0]['use_prect'] = True # False for no pupil search limits, True for
pupul rect

                stagevars[1] =  {}
                stagevars[1]['thresholdchange'] = None # None, 'up', or 'down'

                stagevars[2] = {}
                stagevars[2]['clickpos'] = 0,0 # becomes a (x,y) tuple, indicating click
position within the webcam's snapshots (to determine pupil rect)
                stagevars[2]['prectsize'] = 100,50 # pupilrectsize
                stagevars[2]['prect'] =
pygame.Rect(stagevars[2]['clickpos'][0],stagevars[2]['clickpos'][1],stagevars[2]['prectsize'][
0],stagevars[2]['prectsize'][1]) # rect around pupil, in which the pupil is expected to be
                stagevars[2]['vprectchange'] = None  # None, 'up', or 'down'
                stagevars[2]['hprectchange'] = None  # None, 'right', or 'left'

                stagevars[3] = {}
                stagevars[3]['confirmed'] = False

                # set Booleans
                running = True                # turns False upon quiting the GUI

                # set image variables
                imgsize = self.img.get_size()
                blitpos = (self.dispsize[0]/2-imgsize[0]/2, self.dispsize[1]/2-imgsize[1]/2)

                # # # # #
                # run GUI
                while running:

                        # # # # #
                        # general
```

```
                        # draw stage
                        self.draw_stage(stagenr=stage)

                        # get new snapshot, thresholded image, and pupil measures (only use
pupil bounding rect after stage 1)
                        useprect = stagevars[0]['use_prect'] and stage > 1
                        self.img, self.thresholded, pupilpos, pupilsize, pupilbounds, start =
self.tracker.give_me_all(start, pupilrect=useprect)

                        # update settings
                        self.settings = self.tracker.settings

                        # check if the thresholded image button is active
                        if stagevars[0]['show_threshimg']:
                                # draw active button
                                self.draw_button(self.buttons['t']['active']['img'],
self.buttons['t']['active']['pos'])
                        # if threshold button is not active, draw inactive button
                        else:
                                self.draw_button(self.buttons['t']['inactive']['img'],
self.buttons['t']['inactive']['pos'])

                        # check if the thresholded image button is active
                        if stagevars[0]['use_prect']:
                                # draw active button
                                self.draw_button(self.buttons['r']['active']['img'],
self.buttons['r']['active']['pos'])
                        # if threshold button is not active, draw inactive button
                        else:
                                self.draw_button(self.buttons['r']['inactive']['img'],
self.buttons['r']['inactive']['pos'])

                        # check for input
                        inp, inptype = self.check_input()

                        # handle input, according to the stage (this changes the stagevars!)
                        stage, stagevars = self.handle_input(inptype, inp, stage, stagevars)

                        # # # # #
                        # stage specific

                        # stage 1: setting pupil threshold
                        if stage == 1:
                                # set camera threshold
                                if stagevars[1]['thresholdchange'] != None:
                                        if stagevars[1]['thresholdchange'] == 'up' and
self.settings['threshold'] < 255:
                                                self.settings['threshold'] += 1
                                        elif stagevars[1]['thresholdchange'] == 'down' and
self.settings['threshold'] > 0:
                                                self.settings['threshold'] -= 1
                                        stagevars[1]['thresholdchange'] = None

                        # stage 2: select eye by clicking on it
```

```
                    if stage == 2:
                            # check if input is a mouse click
                            if type(inp) in [tuple,list]:
                                    # check if mouse position is in image
                                    mpos = pygame.mouse.get_pos()
                                    hposok = mpos[0] > blitpos[0] and mpos[0] <
blitpos[0]+imgsize[0]
                                    vposok = mpos[1] > blitpos[1] and mpos[1] <
blitpos[1]+imgsize[1]
                                    if hposok and vposok:
                                            # set pupil position
                                            stagevars[2]['clickpos'] = inp[0]-blitpos[0],
inp[1]-blitpos[1]
                                            self.settings['pupilpos'] =
stagevars[2]['clickpos'][:]
                                            # set pupil rect
                                            x = stagevars[2]['clickpos'][0] -
stagevars[2]['prectsize'][0]/2
                                            y = stagevars[2]['clickpos'][1] -
stagevars[2]['prectsize'][1]/2
                                            stagevars[2]['prect'] =
pygame.Rect(x,y,stagevars[2]['prectsize'][0],stagevars[2]['prectsize'][1])
                                            self.settings['pupilrect'] = stagevars[2]['prect']

                            # if input was a key or button press
                            elif stagevars[2]['vprectchange'] or
stagevars[2]['hprectchange']:
                                    # change pupil rect size
                                    if stagevars[2]['vprectchange'] != None:
                                            if stagevars[2]['vprectchange'] == 'up':
                                                    stagevars[2]['prectsize'] =
stagevars[2]['prectsize'][0], stagevars[2]['prectsize'][1] + 1
                                            elif stagevars[2]['vprectchange'] == 'down':
                                                    stagevars[2]['prectsize'] =
stagevars[2]['prectsize'][0], stagevars[2]['prectsize'][1] - 1
                                            stagevars[2]['vprectchange'] = None
                                    if stagevars[2]['hprectchange'] != None:
                                            if stagevars[2]['hprectchange'] == 'right':
                                                    stagevars[2]['prectsize'] =
stagevars[2]['prectsize'][0] + 1, stagevars[2]['prectsize'][1]
                                            elif stagevars[2]['hprectchange'] == 'left':
                                                    stagevars[2]['prectsize'] =
stagevars[2]['prectsize'][0] - 1, stagevars[2]['prectsize'][1]
                                            stagevars[2]['hprectchange'] = None
                                    # set pupil rect
                                    x = self.settings['pupilrect'][0]
                                    y = self.settings['pupilrect'][1]
                                    stagevars[2]['prect'] =
pygame.Rect(x,y,stagevars[2]['prectsize'][0],stagevars[2]['prectsize'][1])
                                    self.settings['pupilrect'] = stagevars[2]['prect']

                            # draw pupil rect
                            pygame.draw.rect(self.img, (0,0,255), self.settings['pupilrect'],
2)
```

```
                        pygame.draw.rect(self.thresholded, (0,0,255),
self.settings['pupilrect'], 2)

                # stage 3: confirmation
                if stage == 3:
                        # set camera threshold
                        if stagevars[1]['thresholdchange'] != None:
                                if stagevars[1]['thresholdchange'] == 'up' and
self.settings['threshold'] < 255:
                                        self.settings['threshold'] += 1
                                elif stagevars[1]['thresholdchange'] == 'down' and
self.settings['threshold'] > 0:
                                        self.settings['threshold'] -= 1
                                stagevars[1]['thresholdchange'] = None
                        # draw pupil center and pupilbounds in image
                        try: pygame.draw.rect(self.img, (0,255,0),pupilbounds,1);
pygame.draw.rect(self.thresholded, (0,255,0),pupilbounds,1)
                        except: print("pupilbounds=%s" % pupilbounds)
                        try: pygame.draw.circle(self.img, (255,0,0),pupilpos,3,0);
pygame.draw.circle(self.thresholded, (255,0,0),pupilpos,3,0)
                        except: print("pupilpos=%s" % pupilpos)
                        # is settings are confirmed, stop running
                        if stagevars[3]['confirmed']:
                                running = False

                # draw values
                starty = self.dispsize[1]/2 - imgsize[1]/2
                vtx = self.dispsize[0]/2 - imgsize[0]/2 - 10 # 10 isa
                vals = ['pupil colour',str(self.settings['pupilcol']), 'threshold',
str(self.settings['threshold']), 'pupil position', str(self.settings['pupilpos']), 'pupil
rect', str(self.settings['pupilrect'])]
                for i in range(len(vals)):
                        # draw title
                        tsize = self.sfont.size(vals[i])
                        tpos = vtx-tsize[0], starty+i*20
                        tsurf = self.sfont.render(vals[i], True, self.fgc)
                        self.disp.blit(tsurf,tpos)

                # draw new image
                if stagevars[0]['show_threshimg']:
                        self.disp.blit(self.thresholded, blitpos)
                else:
                        self.disp.blit(self.img, blitpos)

                # update display
                pygame.display.flip()

                # apply settings
                self.tracker.settings = self.settings

                # DEBUG #
                if DEBUG:

self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
```

```
                        # # # # #

                # DEBUG #
                if DEBUG:
                        # close savefile
                        self.savefile.close()
                        # message
                        print("processing images...")
                        # open savefile
                        savefile = open('savefile.txt','r')
                        # read ALL contents in once, then close file again
                        raw = savefile.read()
                        savefile.close()
                        # split based on newlines (this leaves one empty entry, because of the
final newline)
                        raw = raw.split(BUFFSEP)
                        # process strings and save image
                        for framenr in range(len(raw)-1):
                                img = pygame.image.fromstring(raw[framenr],self.dispsize,'RGB')
                        pygame.image.save(img,'data/frame%d.png' % framenr)
                # # # # #

                return self.tracker


        def check_input(self):

                """Checks if there is any keyboard or mouse input, then returns
                input (keyname or clickposition) and inptype ('mouseclick' or
                'keypress') or None, None when no input is registered

                arguments
                None

                keyword arguments
                None

                returns
                inp, inptype   --     inp is a keyname (string) when a key has
                                                been pressed, or a click position
                                                ((x,y) tuple) when a mouse button has
                                                been pressed
                                        inptype is a string, either 'mouseclick',
                                                or 'keypress'
                                        if no input is registered, returnvalues
                                        will be None, None
                """

                # if nothing happens, None should be returned
                inp = None
                inptype = None

                # check events in queue
                for event in pygame.event.get():
```

```
                        # mouseclicks
                        if event.type == pygame.MOUSEBUTTONDOWN:
                                inp = pygame.mouse.get_pos()
                                inptype = 'mouseclick'
                        # keypresses
                        elif event.type == pygame.KEYDOWN:
                                inp = pygame.key.name(event.key)
                                inptype = 'keypress'

                return inp, inptype


        def handle_input(self, inptype, inp, stage, stagevars):

                """Checks the input, compares this with what is possible in the
                current stage, then returns adjusted stage and adjusted stage
                variables

                arguments
                inptype         --      string indicating input type, should be
                                                either 'mouseclick' or 'keypress'
                inp                     --      input, should be either
                """

                # # # # #
                # mouseclicks to keypress value

                if inptype == 'mouseclick':

                        # click position
                        pos = inp[:]

                        # loop through buttons
                        for bn in self.buttons.keys():
                                # check if click position is on a button
                                r = self.buttons[bn]['inactive']['rect']
                                if pos[0] > r[0] and pos[0] < r[0]+r[2] and pos[1] > r[1] and
pos[1] < r[1]+r[3]:
                                        # change input to button name
                                        inp = bn
                                        # break from loop (we don't want to loop through all the
other buttons once we've found the clicked one)
                                        break

                # # # # #
                # keypress (or simulated keypress) handling

                # stage 1
                if stage == 1:
                        # up should increase threshold, down should decrease threshold
                        if inp in ['up','down']:
                                stagevars[1]['thresholdchange'] = inp

                # stage 2
```

```
		elif stage == 2:
			# up should increase pupil rect size, down should decrease pupil rect
size
			if inp in ['up','down']:
				stagevars[2]['vprectchange'] = inp
			elif inp in ['left','right']:
				stagevars[2]['hprectchange'] = inp

		# stage 3
		elif stage == 3:
			# up should increase threshold, down should decrease threshold
			if inp in ['up','down']:
				stagevars[1]['thresholdchange'] = inp
			# space should confirm settings
			if inp == 'space':
				stagevars[3]['confirmed'] = True

		# space should move to next stage (but not in stage 3)
		if inp == 'space' and stage < 3:
			stage += 1

		# number keys should make the stage jump to that number
		if inp in ['1','2','3']:
			stage = int(inp)

		# T should toggle between displays
		if inp == 't':
			if stagevars[0]['show_threshimg']:
				stagevars[0]['show_threshimg'] = False
			else:
				stagevars[0]['show_threshimg'] = True

		# R should toggle between using pupil rect or not
		if inp == 'r':
			if stagevars[0]['use_prect']:
				stagevars[0]['use_prect'] = False
			else:
				stagevars[0]['use_prect'] = True

		# escape should close down
		if inp == 'escape':
			pygame.display.quit()
			raise Exception("camtracker.Setup: Escape was pressed")

		# return the changed variables
		return stage, stagevars


class CamEyeTracker:

	"""The CamEyeTracker class uses your webcam as an eye tracker"""

	def __init__(self, device=None, camres=(640,480)):
```

```
		"""Initializes a CamEyeTracker instance

		arguments
		None

		keyword arguments
		device		--	a string or an integer, indicating either
					device name (e.g. '/dev/video0'), or a
					device number (e.g. 0); None can be passed
					too, in this case Setup will autodetect a
					useable device (default = None)
		camres		--	the resolution of the webcam, e.g.
					(640,480) (default = (640,480))
		"""

		global vcAvailable
		if vcAvailable == False:
			# select a device if none was selected
			if device == None:
				available = available_devices()
				if available == []:
					raise Exception("Error in
camtracker.CamEyeTracker.__init__: no available camera devices found (did you forget to plug
it in?)")
				else:
					device = available[0]

			# start the webcam
			self.cam = pygame.camera.Camera(device, camres, 'RGB')
			self.cam.start()
		else:
			self.cam = VideoCapture.Device()

		# get the webcam resolution (get_size not available on all systems)
		try:
			self.camres = self.cam.get_size()
		except:
			self.camres = camres

		# default settings
		self.settings = {'pupilcol':(0,0,0), \
					'threshold':100, \
					'nonthresholdcol':(100,100,255,255), \
					'pupilpos':(-1,-1), \

'pupilrect':pygame.Rect(self.camres[0]/2-50,self.camres[1]/2-25,100,50), \
					'pupilbounds': [0,0,0,0], \
					'':None
					}


	def get_size(self):

		"""Returns a (w,h) tuple of the image size
```

```
        arguments
        None

        keyword arguments
        None

        returns
        imgsize        --      a (width,height) tuple indicating the size
                                of the images produced by the webcam
        """
        return self.camres


    def get_snapshot(self):

        """Returns a snapshot, without doing any any processing

        arguments
        None

        keyword arguments
        None

        returns
        snapshot               --      a pygame.surface.Surface instance,
                                       containing a snapshot taken with the webcam
        """
        global vcAvailable
        if vcAvailable:
               image = self.cam.getImage()
               mode = image.mode
               size = image.size
               data = image.tostring()
               return pygame.image.fromstring(data, size, mode)
        else:
               return self.cam.get_image()


    def threshold_image(self, image):

        """Applies a threshold to an image and returns the thresholded
        image

        arguments
        image                  --      the image that should be thresholded, a
                                       pygame.surface.Surface instance

        returns
        thresholded            --      the thresholded image,
                                       a pygame.surface.Surface instance
        """

        # surface to apply threshold to surface
```

```
                thimg = pygame.surface.Surface(self.get_size(), 0, image)

                # perform thresholding
                th =
(self.settings['threshold'],self.settings['threshold'],self.settings['threshold'])
                pygame.transform.threshold(thimg, image, self.settings['pupilcol'], th,
self.settings['nonthresholdcol'], 1)

                return thimg


        def find_pupil(self, thresholded, timestart, pupilrect=True):

                #timenow=time.time()

                current = datetime.now() #added timestamping
                year = current.strftime("%Y")
                month = current.strftime("%m")
                day = current.strftime("%d")
                second = current.strftime("%H:%M:%S.%f")

                """Get the pupil center, bounds, and size, based on the thresholded
                image; please note that the pupil bounds and size are very
                arbitrary: they provide information on the pupil within the
                thresholded image, meaning that they would appear larger if the
                camera is placed closer towards a subject, even though the
                subject's pupil does not dilate

                arguments
                thresholded           --      a pygame.surface.Surface instance, as
                                              returned by threshold_image

                keyword arguments
                pupilrect             --      a Boolean indicating whether pupil searching
                                              rect should be applied or not
                                              (default = False)

                returns
                pupilcenter, pupilsize, pupilbounds
                                      --      pupilcenter is an (x,y) position tuple that
                                              gives the pupil center with regards to the
                                              image (where the top left is (0,0))
                                              pupilsize is the amount of pixels that are
                                              considered to be part of the pupil in the
                                              thresholded image; when no pupilbounds can
                                              be found, this will return (-1,-1)
                                              pupilbounds is a (x,y,width,height) tuple,
                                              specifying the size of the largest square
                                              in which the pupil would fit
                """


                # cut out pupilrect (but only if pupil bounding rect option is on)
                if pupilrect:
```

```
                        # pupil rect boundaries
                        rectbounds = pygame.Rect(self.settings['pupilrect'])
                        # correct rect edges that go beyond image boundaries
                        if self.settings['pupilrect'].left < 0:
                                rectbounds.left = 0
                        if self.settings['pupilrect'].right > self.camres[0]:
                                rectbounds.right = self.camres[0]
                        if self.settings['pupilrect'].top < 0:
                                rectbounds.top = 0
                        if self.settings['pupilrect'].bottom > self.camres[1]:
                                rectbounds.bottom = self.camres[1]
                        # cut rect out of image
                        thresholded = thresholded.subsurface(rectbounds)
                        ox, oy = thresholded.get_offset()

                # find potential pupil areas based on threshold
                th =
(self.settings['threshold'],self.settings['threshold'],self.settings['threshold'])
                mask = pygame.mask.from_threshold(thresholded, self.settings['pupilcol'], th)

                # get largest connected area within mask (which should be the pupil)
                pupil = mask.connected_component()

                # get pupil center
                pupilcenter = pupil.centroid()

                # if we can only look within a rect around the pupil, do so
                if pupilrect:
                        # compensate for subsurface offset
                        pupilcenter = pupilcenter[0]+ox, pupilcenter[1]+oy
                        # check if the pupil position is within the rect
                        if (self.settings['pupilrect'].left < pupilcenter[0] <
self.settings['pupilrect'].right) and (self.settings['pupilrect'].top < pupilcenter[1] <
self.settings['pupilrect'].bottom):
                                # set new pupil and rect position
                                self.settings['pupilpos'] = pupilcenter
                                x = pupilcenter[0] - self.settings['pupilrect'][2]/2
                                y = pupilcenter[1] - self.settings['pupilrect'][3]/2
                                self.settings['pupilrect'] =
pygame.Rect(x,y,self.settings['pupilrect'][2],self.settings['pupilrect'][3])
                                timestart = time.time() # record time now, set to timestart

                        # if the pupil is outside of the rect, return missing
                        else:
                                self.settings['pupilpos'] = (-1,-1) #set pupil position to -1, -1
                                end = time.time() # record time now
                                #print(end-timestart) # debugging purposes
                                #print("Missing!", year, month, day, second) #added timestamping
                                if ((end - timestart) > 2.0):
                                        # if 2 seconds have elapsed since pupils went missing,
print missing and time stamp
                                        print("Missing!", year, month, day, second)
                else:
                        self.settings['pupilpos'] = pupilcenter
```

```
		# get pupil bounds (sometimes failes, hence try-except)
		try:
			self.settings['pupilbounds'] = pupil.get_bounding_rects()[0]
			# if we're using a pupil rect, compensate offset
			if pupilrect:
				self.settings['pupilbounds'].left += ox
				self.settings['pupilbounds'].top += oy
		except:
			# if it fails, we simply use the old rect
			pass

		return self.settings['pupilpos'], pupil.count(), self.settings['pupilbounds'],
timestart


	def give_me_all(self, timestart, pupilrect=False):

		"""Returns snapshot, thresholded image, pupil position, pupil area,
		and pupil bounds

		arguments
		None

		keyword arguments
		pupilrect		--	a Boolean indicating whether pupil searching
					rect should be applied or not
					(default = False)

		returns
		snapshot, thresholded, pupilcenter, pupilbounds, pupilsize
			snapshot	--	a pygame.surface.Surface instance,
						containing a snapshot taken with the webcam
			thresholded	--	the thresholded image,
						a pygame.surface.Surface instance
			pupilcenter	--	pupilcenter is an (x,y) position tuple that
						gives the pupil center with regards to the
						image (where the top left is (0,0))
			pupilsize	--	pupilsize is the amount of pixels that are
						considered to be part of the pupil in the
						thresholded image; when no pupilbounds can
						be found, this will return (-1,-1)
			pupilbounds	--	pupilbounds is a (x,y,width,height) tuple,
						specifying the size of the largest square
						in which the pupil would fit
		"""

		img = self.get_snapshot()
		thimg = self.threshold_image(img)
		ppos, parea, pbounds, timestart = self.find_pupil(thimg, timestart, pupilrect)

		return img, thimg, ppos, parea, pbounds, timestart
```

```
    def close(self):

            """Shuts down connection to the webcam and closes logfile

            arguments
            None

            keyword arguments
            None

            returns
            None
            """

            # close camera
            self.cam.stop()
```