

## ABSTRACT

### The Illinois Junior Academy of Science

This form/paper may not be taken without IJAS authorization.

CATEGORY	<u>Engineering</u>	STATE REGION #	<u>6</u>
SCHOOL	<u>Maine Township High School South</u>	IJAS SCHOOL #	<u>6011</u>
CITY/ZIP	<u>Park Ridge / 60068</u>	SPONSOR CELL PHONE #	<u>847-825-7711</u>
SPONSOR	<u>Kay Wagner</u>	SPONSOR E-MAIL	<u>kwagner@maine207.org</u>

MARK ONE:    EXPERIMENTAL INVESTIGATION ☐                      DESIGN INVESTIGATION ☒

NAME OF SCIENTIST*	<u>Joseph Habisohn</u>	GRADE	<u>11</u>
NAME OF SCIENTIST		GRADE	
NAME OF SCIENTIST		GRADE	
NAME OF SCIENTIST		GRADE	

PROJECT TITLE    Making a Braking Distance, 3-Second Rule Gauging Tool with a Raspberry Pi

**Purpose:** This research and design project was the development of a device prototype to increase the driver's awareness. It will tremendously help inexperienced drivers who don't have the instincts of their experienced counterparts. In addition, it can also serve as an aid during driving training to give the learner concrete feedback about the distances in front of them.

**Procedure:** The software was coded in Python directly on a Raspberry Pi 3B+ using the Thonny IDE. The Assembly Holder was then designed in Autodesk Onshape. It was designed to fit a TFT, the Raspberry Pi, and a Transparent Pane held at a 50-degree angle to the TFT. The Transparent Pane has a Reflective film attached, so the image is easier to see. The entire system was then placed in a vehicle driven by an experienced driver to test if the software works as intended.

**Conclusion:** As a whole, the device works as intended. All of the Peripherals connect, and the software starts up automatically when the Raspberry Pi turns on, in addition to the GUI updating in response to the LiDAR and the OBD-II. Therefore, if the device could be made more compact and more efficient, it would be a viable product to increase driver safety, including the young drivers who need the extra safety features when first learning. Finally, the device meets the set requirements, as it gets the vehicle's speed, gets the distance to the next vehicle, calculates the braking and three-second rule distance, and safely displays the information to the driver.

# SAFETY SHEET

The Illinois Junior Academy of Science

Possible hazards	Precautions taken to deal with each hazard
Testing HUD with a moving car.  Using a laser cutter to cut acrylic UV light (harms eyes), Fumes (methyl methacrylate).	An adult with ample driving experience drove the car. Laser glasses were worn, and the exhaust was operational.

Specific safety practices related to materials requiring endorsement sheets should be detailed on the specific endorsement sheet and not included on this safety sheet.

Please check off any other possible endorsements needed. Include these documents in your paper and on your board.

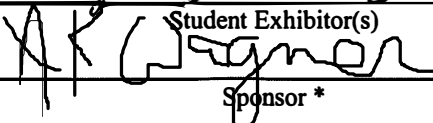
- ☒ Humans as Test Subjects –for any projects involving humans, including survey administration;  
☐ Microorganisms-for any projects involving bacteria, viruses, yeasts, fungi, or protozoa;  
☐ Non-Human Vertebrates -for any projects involving fish, amphibians, reptiles, birds or mammals;  
☐ Tissue Culture-for any projects involving growing eukaryotic tissues or cell cultures;  
☐ Letter from an institution where research was done or IJAS SRC, if an exception to the IJAS rules has been granted...

SIGNED: Joseph Habisohn



Student Exhibitor(s)

SIGNED: Kay Wagner



Sponsor \*

\*As a sponsor, I assume all responsibilities related to this project.

# Humans as Test Subjects Endorsement 2022-2024

Illinois Junior Academy of Science

These rules will be strictly enforced for the State Science Exposition.  
No region should send a project to the State Exposition that does not meet these regulations.

Students and sponsors doing a human vertebrate project must complete this form. The signature of the student or students and the sponsor indicates that the project was done within these rules and regulations. Failure to comply with these rules will mean the disqualification of the project at the state level. This form must follow the Safety Sheet in the project paper.

1. Humans must not be subjected to treatments that are considered hazardous and/or that could result in undue stress, injury, or death to the subject.
2. No primary or secondary cultures taken directly (mouth, throat, skin, etc.) or indirectly (eating utensils, countertops, doorknobs, toilets, etc.) will be allowed. However, cultures obtained from reputable biological suppliers or research facilities are suitable for student use.
3. Quantities of food and non-alcoholic beverages are limited to normal serving amounts or less and must be consumed in a reasonable amount of time. Normal serving amounts must be substantiated with reliable documentation. This documentation must be attached to the Humans as Test Subjects Endorsement form. No project may use over-the-counter, prescription, illegal drugs, or alcohol in order to measure their effect on a person.
4. The only human blood that may be used is that which is either purchased or obtained from a blood bank, hospital, or laboratory. No blood may be drawn by any person or from any person specifically for a science project. This rule does not preclude a student making use of data collected from blood tests not made exclusively for a science project.
5. Projects that involve exercise and its effect on pulse, respiration rate, blood pressure, and so on are allowed provided the exercise is not carried to the extreme. Electrical stimulation is not permitted. A valid, normal physical examination must be on file for each test subject. Documentation of same must be attached to the Humans as Test Subjects Endorsement form.
6. Projects that involve learning, ESP, motivation, hearing, vision, and surveys require the **Humans as Test Subjects** form.

The signatures of the student or students and sponsor below indicate that the project conforms to the above rules of the Illinois Junior Academy of Science.

Fill out the following charts:

Were humans given food? If so, was it a serving size or less?	No.
Were humans subjected to exercise? If so, is there evidence of a physical on file for each test subject?	No.
Briefly describe how humans were used in the investigation.	A human was needed to drive the car during all field tests.

Describe the possible risks to humans test subjects.	Describe how each risk was handled or avoided.
Driving a moving vehicle while using a display can be dangerous and is illegal for minors.	A very experienced driver (adult) drove the car while the researcher monitored the HUD during all field tests.

Kay Wagner

(Sponsor)\*

Joseph Habisohn

(Student)

March 17, 2023

(Date)

(Student)

\*As a sponsor, I assume all responsibilities related to this project.

This form **must** be displayed on the front of the exhibitor's display board. It may be reduced to half a sheet of paper 8.5 inches (vertical) X 5.5 inches (horizontal).



Check box if exception/approval letter from an institution where research was done, or the IJAS SRC is required and attached.

## Title Page

Making a Braking Distance, 3-Second Rule Gauging Tool with a Raspberry Pi

Joseph Habisohn

Maine South High School



## **Table of Contents**

Abstract	<b>1</b>
Safety Sheet	<b>2</b>
Humans as Test Subjects Endorsement	<b>3</b>
Title Page	<b>4</b>
Table of Contents	<b>5</b>
Acknowledgments:	<b>6</b>
Purpose	<b>7</b>
Background Research	<b>8</b>
List of Materials	<b>13</b>
Design Plan	<b>13</b>
First Iteration	13
Second Iteration	25
Final Iteration	36
Discussion	<b>52</b>
Conclusion	<b>54</b>
Bibliography	<b>55</b>

### **Acknowledgments:**

I would like to acknowledge several individuals for their guidance, support, and encouragement in this opportunity to participate in the scientific research and scholarship community. These persons are Mrs. Kay Wagner, P.Eng., B.Sc., M.A.Sc., M.A.T., of Maine South High School, for running the Science Research Class and giving me the chance to do this project; Michael V. Habisohn B.S. Engineering Physics, for advice on integrating the LiDAR with the Raspberry Pi and executing a Python file on boot; Dr. Ana Bell Ph.D., lecturer for on-campus courses 6.0001 and 6.0002 at MIT and Prof. Eric Grimson Ph.D., Professor of Computer Science and Chancellor for Academic Advancement, for teaching MIT OCW 6-0001 Introduction to Computer Science and Programming in Python.

## Purpose

Driving is often challenging for new drivers, and crashes are common. Ideas like stopping distance, the 3-Second Rule, and car lengths can be challenging for new drivers to quantify in physical space relative to their vehicle. Often, this only comes with experience, which many drivers never live to attain. On average, driving deaths and injuries are most common in 16 to 17-year-olds. Therefore it is crucial to enhance driver awareness with computing. This is because modern computers can do calculations much faster than humans. Such a device would give a new driver time to gain the instincts mentioned above. If there were a device that could predict if a crash could happen and warn the driver, many traffic collisions would be avoided. In addition, having a computer do some of the tasks that a driver would typically perform would allow the driver to stay more focused on the road. This would contribute to traffic safety and the driver's peace of mind. Finally, it can also aid during driving training by giving the learner concrete feedback about the distances in front of them. The end goal of this project is to design a working proof of concept for such a device.

## Background Research

When driving a vehicle, it is crucial to maintain an optimal following distance when behind another vehicle. There are many rules of thumb to calculate that distance quickly; the most notable is the 3-Second Rule. According to the Illinois DMV, the protocol of the 3-Second rule is as follows: ‘...select a fixed object on the road ahead such as a sign, tree or overpass. When the vehicle in front of you passes the object, count “one-thousand-one, one-thousand-two, etc.” Your vehicle should not reach the object before the count of one-thousand-three. If this occurs, you are following too close.’ (The Secretary of State, 2022).

Another important distance a driver must keep track of is their braking distance, the distance that a car will coast when the brake pedal is pressed until the vehicle comes to rest. According to the National Association of City Transportation Officials (NACTO), the braking distance of a vehicle depends on the speed of the vehicle. For instance, a car moving at 26.8224 m/s (60 mph) has a braking distance of 52.43m (172 ft) (Tbl. 1) (NACTO, 2013).

The issue that many drivers face is that calculating these distances takes attention away

MPH	Ft./Sec.	Braking Deceleration Distance	Perception Reaction Distance	Total Stopping Distance
10	14.7	5	22	27
15	22	11	33	44
20	29.3	19	44	63
25	36	30	55	85
30	44	43	66	109
35	51.3	59	77	136
40	58.7	76	88	164
45	66	97	99	196
50	73.3	119	110	229
55	80.7	144	121	265
60	88	172	132	304
65	95.3	202	143	345
70	102.7	234	154	388
75	110	268	165	433
80	117.3	305	176	481
85	124.7	345	187	532
90	132	386	198	584

Table 1: Table of stopping and braking distances vs. their respective speeds. Taken from nacto.org (NACTO, 2013).



from the road and is inaccurate, or, in the case of braking distance, requires memorizing a table. In this case, a computer is much better suited for such calculations because a computer can do these calculations quickly and accurately. Such a computer would need to do a few things: get the vehicle's speed, get the distance to the next vehicle, and safely display the information to the driver.

The computer itself could be a phone, a laptop, an Arduino, or a Raspberry Pi. Making an application for a phone or a laptop is viable. However, two of the computers themselves have their own issues. For a laptop to run, it must be open, which is a significant distraction for the driver. Although a HUD could potentially reflect the phone's image into the driver's field of vision, it is significantly more challenging to make a phone app use sensor data than the other two options. Next, the Arduino has two disadvantages to the Raspberry Pi. Since the Raspberry Pi is a whole computer, the GUI will be much easier to read by the driver, and two, there are many more functionalities that can be added in the future since a Raspberry Pi is much more advanced than an Arduino.

There are multiple ways to get the vehicle's current speed. One, integrate the acceleration measured by an accelerometer over time, two, use a website, and three, get the current speed directly from the vehicle using the OBD-II (On-Board Diagnostic II) port. Using a website is viable because the Raspberry Pi has a web browser. However, the accuracy of such a website cannot be trusted, and an internet connection must be maintained the entire time. In addition, if the website buffers, the measurement from the site may no longer be the actual current speed. Therefore, using a website is ruled out, and just like the Raspberry Pi versus the Arduino, the OBD-II port can do much more than get the vehicle's speed, as the accelerometer can only do. The OBD-II port, a 16-pin connector integrated directly into the car, allows a

technician to view diagnostic information from the vehicle's onboard computer. This system turns on the warning lights on the dashboard (California Air Resources Board, 2019). In addition to diagnostic data, the OBD-II also outputs the current velocity of the car. The current velocity is the most essential value for both the braking distance and the 3-second following distance.

The industry's three most common ways to measure distance are Time-of-Flight (ToF), Phase-Shift (PS), and Triangulation. Triangulation sensors work by calculating the angle at which the laser is reflected back. This works because the laser is often not perfectly reflected straight back at that sensor. The transmitted laser and the reflected laser form a right triangle, and basic trigonometry is used to calculate the distance from the object (Acuity Laser, 2021). Unfortunately, triangulation sensors are only accurate at very small distances and are not viable for the long ranges seen on the road (Acuity Laser, 2021), while both ToF and PS are capable of long distances.

ToF is the most common type of distance measurement used in the automotive industry, specifically LiDAR (Light Detection And Ranging). A LiDAR sensor calculates the distance to another object by sending pulsed light out and timing how long it takes to reflect back to the sensor (Terabee, 2022). If it is a 3D or 2D sensor, it then generates a map of the scanned area (Mehendale & Neoge, 2020). The equation is  $D = \frac{c}{2} \cdot \Delta t$ , where  $D$  is the distance measured,  $c$  is the speed of light (299,792,458 m/s), and  $\Delta t$  is the time for the photons to reflect back to the sensor. PS sensors work similarly to ToF; however, they use a continuous laser beam instead of pulsed light, but the amplitude is modulated. The sensor determines the phase shift ( $\varphi$ ) between the transmitted and received beams, which is then used to calculate the distance ( $D$ ). The equation is  $D = \frac{c}{2f} \cdot \frac{\varphi}{2\pi}$ , where  $f$  is the modulation frequency and  $c$  is the speed of light

(299,792,458 m/s). Functionally, ToF and PS have different case uses. ToF is used for civil engineering when long distances are typical. (Suchocki, 2020). LiDAR is often used when researchers need to scan a vast area, such as shorelines and other natural and artificial environments (NOAA, 2012).

Finally, ToF sensors do not perform nearly as well as PS. PS sensors are high-speed and accurate, but they only have a medium range of 100m, though there are modern PS Sensors that have a range of 300m like that of their ToF counterparts (Suchocki, 2020). This 100m range is perfect for linear distance sensing for a vehicle. This is because a braking distance of 100m (328.084 ft) requires an initial velocity of approximately 38m/s (85 mph) (Tbl. 1).

There are many ways for the device to relay information to the driver, notably, the dashboard, the GPS screen, or a HUD. However, two of these options have a potentially fatal issue: they take the driver's eyes off the road. A HUD does not have this issue because it is not only in the driver's field of view but also does not obstruct the driver's vision. It does this by reflecting an image from a display onto the windshield with two mirrors (Fig. 1) (Youngworth, Tsao, and Chan, 2021). The effect of a windshield HUD was studied in the article "Employing Emerging Technologies to Develop and Evaluate In-Vehicle Intelligent Systems for Driver Support: Infotainment AR HUD Case Study," The researchers, using a driving simulation, recorded the number of collisions when the users used a HUD versus Heads-Down Display (HDD), i.e., the dashboard, was used. The researchers found that 13% of users avoided the collision when using the HDD, while 77% of users using a HUD were able to avoid the collision (Charissis et al., 2021).

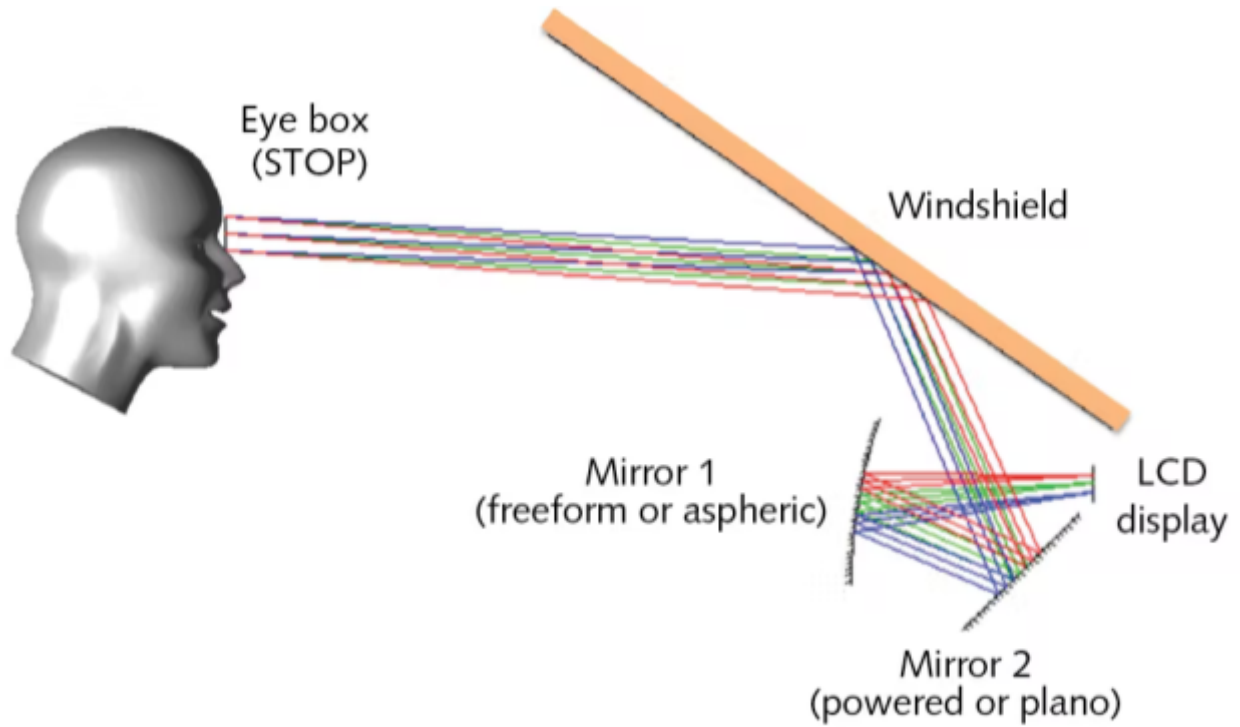


Figure 1: Visual representation of an integrated HUD in a vehicle. Taken from [laserfocusworld.com](http://laserfocusworld.com) (Youngworth, Tsao, and Chan, 2021).

## List of Materials

List of Materials				
Name	Quantity	Part #/Batch #	Size	Source
Hardware				
Raspberry Pi 3 B+	1	SC15184	94.996 x 74.93 x 27.94mm	Raspberry Pi
Raspberry Pi 3 B+ Air Cooled Case	1	n.d.	91.440 x 61.913 x 32.147mm	Came with Raspberry Pi
Hud Projection Film Two Pack	1	n.d.	150 x 130mm (2 sheets)	Amazon
JRT Laser Module, 100m	1	n.d.	44.12 x 43.05 x 31.30mm	Chengdu JRT Meter Technology Co., Ltd
TF03-UART LiDAR	1	PNF-1505A	44 x 43 x 32mm	Benewake/DFRobot
3D Printed Assembly Holder	1	n.d.	306.39 x 251.35 x 213.91mm	custom made
Transparent Pane	1	n.d.	309.42 x 181.92 x 6.35mm	custom made
TFT LCD	1	n.d.	1024x600 Pixels/160.4 x 124.14 x 29.07mm	n.d.
UART CAN SuitBoard V1.0 for TF02 and TF03	1	n.d.	53.65 x 28.84 x 9.45mm	n.d.
ELM327 OBD-II to USB adapter	1	FORD327	214.122 x 149.098 x 41.91mm	OBDResource
Cables				
Raspberry Pi 4 USB-C Power Supply	1	DCAR-RSP-3A5C	1.524m	Canakit
HDMI cable	1	n.d.	26.7cm	generic
HDMI to mini-HDMI adapter	1	n.d.	16.2cm	CableCreation
micro-USB	1	n.d.	15cm	generic
S2 mini-USB, came with UART CAN SuitBoard	1	n.d.	1.01m	n.d.
12v Power Inverter	1	CPS160PPB2U	92 x 41 x 92mm	CyberPower
Female to Male USB Extension Cord	1	n.d.	1.593m	generic
Male-Male Jumper Wires	4	n.d.	30.4cm	generic
Software				
Onshape				Autodesk
DFRobot_TFmini Library				Ceaser-S on GitHub
Numpy				Comes with Python
Serial				Comes with Python
python-OBD				Brendan Whitfield on GitHub
Tkinter				Comes with Python
Thonny Python 3.9 IDE				Comes with the Raspberry Pi

Table 2: List of Materials

## Design Plan

### *First Iteration*

The Raspberry Pi has 40 GPIO pins (Fig. 2), which can be used to connect the PS laser to the Raspberry Pi. In addition, it has 4 USB ports that can be used to connect to other peripherals. Conveniently, the GPIO pins and the USBs are serial communication ports and can be handled the same way in the code. This code handles all of the calculations and displays the information to a GUI, which is then displayed onto a 1024x600px Thin-Film-Transistor Liquid-Crystal-Display (TFT LCD, or TFT), which is also powered by the Raspberry Pi's USB

ports and connected to the Micro HDMI port via the HDMI to Micro HDMI adapter that comes with the Raspberry Pi. In addition to the TFT, an OBD-II to USB adapter is connected to one of the USB ports on the Raspberry Pi, and then it uses serial to communicate with the code. The Laser has four wires, ground (GND), 3.3v, and two signal wires.

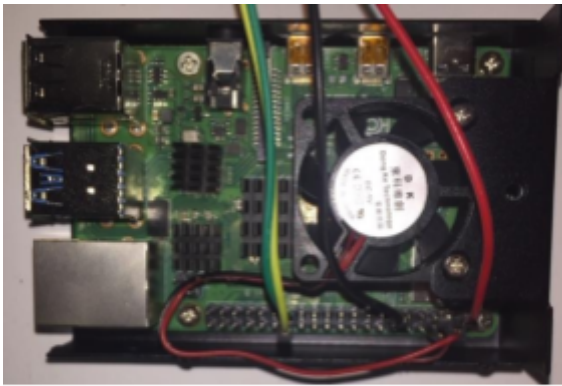


Figure 2: Top view of the GPIO pins on a Raspberry Pi 4 B.



Figure 3: Top view of the Laser and the Raspberry Pi 4 B.

Before serial communication can be used, the Raspberry Pi has to be configured for it. First, upgrade all of the packages and update the Raspberry Pi by running the following commands in the terminal (note: it may prompt with “Do you want to continue [Y/n].” In that case, type “y” and press enter):

```
sudo apt upgrade
sudo apt update
```

Second, run `sudo raspi-config` in the terminal to get into the Raspberry Pi config menu.

Third, navigate to Interface Options using the down arrow key and hit enter. Fourth, navigate to Serial Port and hit enter. For the first prompt (“Would you like a login shell to be accessible over serial?”), select <No> and hit enter, and for the second prompt (“Would you like serial port

hardware to be enabled?”), select <Yes> and hit enter. For the last prompt, select and enter <Ok>, then, using the right arrow key, select and enter <Finish>, shown in the first menu (Fig. 4). Finally, in the terminal, reboot the Raspberry Pi using `sudo reboot`.

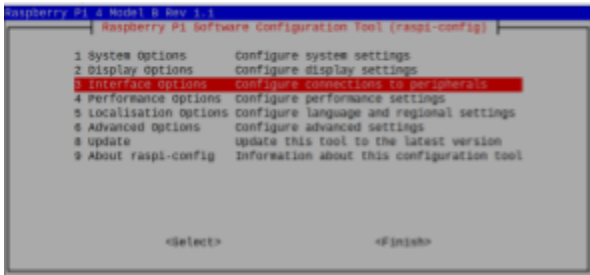


Figure 4a: Press the down arrow key twice and press enter.

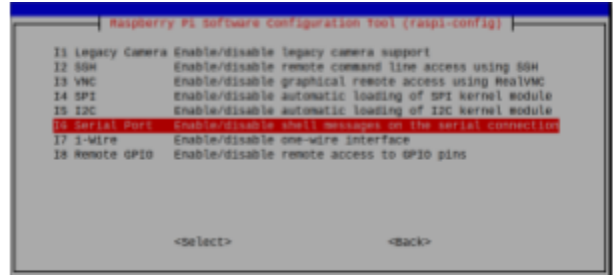


Figure 4b: Press the down arrow key 5 times and press enter.



Figure 4c: Use the right arrow key to select <No> and press enter.



Figure 4d: Use the left arrow key to select <Yes> and press enter.

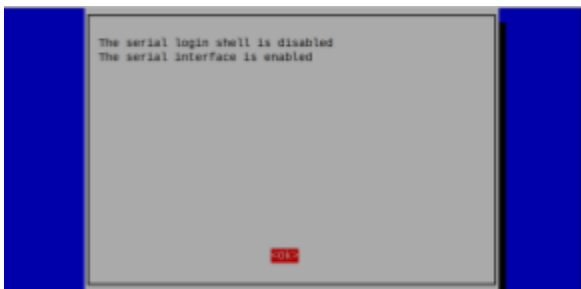


Figure 4e: Press enter.

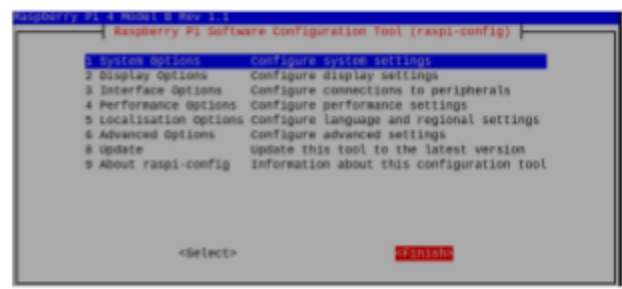


Figure 4f: Press the right arrow key twice and press enter.

Figure 4: Graphical instructions of how to enable serial ports.

The code is written in Python 3.9 through the Thonny IDE that comes with the Raspberry Pi, using the Serial, python-OBD (Whitfield, 2019), NumPy, and Tkinter Python packages. The first iteration of the software was written not to include communication with the OBD-II to allow for easier development; instead, it was simulated with a Tkinter slider (Scale

Widget). In addition, the TFT was not used in the initial iteration of the design. Next, it used the Threading Python package to allow the use of multi-threading with Tkinter. This is because Tkinter uses an event loop that does not allow other code to run. Finally, it did not use NumPy because it is an addition added in the second iteration of the design. Note that all three iterations of the software are available on GitHub:

<https://github.com/joebewon/Car-HUD-Software-Repository>.

---

```

1  import serial # Import the Python Serial package for serial
   communication
2  from tkinter import * # Import Tkinter for the GUI
3  import threading as thread # Import the Python Threading package
   for multithreading
4
5  # Set up serial connection to Laser
6  ser = serial.Serial(
7      port='/dev/ttyAMA1',
8      baudrate = 19200,
9      parity=serial.PARITY_NONE,
10     stopbits=serial.STOPBITS_ONE,
11     bytesize=serial.EIGHTBITS,
12     timeout=1
13 )
14
15 # GUI class based on APP class from
16 # stackoverflow.com/questions/459083 | Kevin's Response
17
18 # Run all GUI code in its own thread
19 # to allow both the rest of the code to run
20 class GUI(threading.Thread):
21     def __init__(self):
22         threading.Thread.__init__(self)
23         self.start()
24

```



```

25     def callback(self):
26         self.root.quit()
27
28     # Run the GUI
29     def run(self):
30         # Set up window
31         self.root = Tk()
32         self.root.protocol("WM_DELETE_WINDOW", self.callback)
33
34         width = self.root.winfo_screenwidth() # Get width of
screen
35         height = self.root.winfo_screenheight() # Get height of
screen
36
37         self.root.geometry("%dx%d" % (width, height)) # Set
size of window to fullscreen
38
39         # Set up Distance text
40         global distance # Put distance into the global scope
41         distance = Label(self.root, text="Starting Up",
font=("Arial", 25))
42         distance.grid(row=0, column=0)
43
44         # Set up Braking Distance Text
45         global braking_distance
46         braking_distance = Label(self.root, text=" ", font=("Arial",
25))
47         braking_distance.grid(row=0, column=2)
48
49         # Braking Distance indicator
50         global BD_indicator
51         BD_indicator = Canvas(self.root, height=76, width=228,
bg="gray")
52         BD_indicator.grid(row=1, column=2)
53
54         global BDlight
55         BDlight = BD_indicator.create_oval(8, 8, 68, 68,
outline="red", fill="gray")
56

```

```

57     global BDtext
58     BDtext = BD_indicator.create_text(114, 38, font=("Arial",
13), text="Error, Proceed with caution")
59
60     # TEMP: Make speed slider in m/s
61     global speed
62     speed = Scale(self.root, from_=0, to=40,
orient="horizontal")
63     speed.grid(row=2, column=0)
64
65     # 3 second rule header
66     global threeSHeader
67     threeSHeader = Label(self.root, text="Three second rule
distance:", font=("Arial", 25))
68     threeSHeader.grid(row=0, column=3)
69
70     # 3 second rule indicator
71     global TSR_indicator
72     TSR_indicator = Canvas(self.root, height=76, width=228,
bg="gray")
73     TSR_indicator.grid(row=1, column=3)
74
75     # create the TSR stop light
76     global TSRlight
77     TSRlight = TSR_indicator.create_oval(8, 8, 68, 68,
outline="gray", fill="gray")
78
79     global TSRtext
80     TSRtext = TSR_indicator.create_text(114, 38, font=("Arial",
13), text="Error, Proceed with caution")
81
82     # Start event loop
83     self.root.mainloop()
84
85     # print to GUI as (widget, new_text)
86     def printg(*args):
87         args[1].config(text=args[2])
88
89     # Change the options of the canvas oval item

```

```

90     # red = "red", yellow = "yellow", green = "#00E518"
91     def changeTSRLight(self, color):
92         if color.lower() == "red":
93             TSR_indicator.itemconfigure(TSRtext, fill="gray")
94             TSR_indicator.coords(TSRtext, 0, 0)
95             TSR_indicator.itemconfigure(TSRlight,
outline=color.lower(), fill=color.lower())
96             TSR_indicator.coords(TSRlight, 8, 8, 68, 68)
97
98         elif color.lower() == "yellow":
99             TSR_indicator.itemconfigure(TSRtext, fill="gray")
100             TSR_indicator.coords(TSRtext, 0, 0)
101             TSR_indicator.itemconfigure(TSRlight,
outline=color.lower(), fill=color.lower())
102             TSR_indicator.coords(TSRlight, 8 + 76, 8, 68 + 76,
68)
103
104         elif color.lower() == "#00E518" or color.lower() ==
"green":
105             TSR_indicator.itemconfigure(TSRtext, fill="gray")
106             TSR_indicator.coords(TSRtext, 0, 0)
107             TSR_indicator.itemconfigure(TSRlight,
outline="#00E518", fill="#00E518")
108             TSR_indicator.coords(TSRlight, 8 + 76 + 76, 8, 68
+ 76 + 76, 68)
109
110         elif color.lower() == "err":
111             TSR_indicator.itemconfigure(TSRlight,
outline="gray", fill="gray")
112             TSR_indicator.itemconfigure(TSRtext, fill="black")
113             TSR_indicator.coords(TSRtext, 114, 38)
114
115         else:
116             raise ValueError("Incorrect input: color must be
\"red\", \"yellow\", \"#00E518\", \"green\", or \"err\"")
117
118     def changeBDLight(self, color):
119         if color.lower() == "red":
120             BD_indicator.itemconfigure(BDtext, fill="gray")

```

```

121         BD_indicator.coords(BDtext, 0, 0)
122         BD_indicator.itemconfigure(BDlight,
outline=color.lower(), fill=color.lower())
123         BD_indicator.coords(BDlight, 8, 8, 68, 68)
124
125     elif color.lower() == "yellow":
126         BD_indicator.itemconfigure(BDtext, fill="gray")
127         BD_indicator.coords(BDtext, 0, 0)
128         BD_indicator.itemconfigure(BDlight,
outline=color.lower(), fill=color.lower())
129         BD_indicator.coords(BDlight, 8 + 76, 8, 68 + 76,
68)
130
131     elif color.lower() == "#00E518" or color.lower() ==
"green":
132         BD_indicator.itemconfigure(BDtext, fill="gray")
133         BD_indicator.coords(BDtext, 0, 0)
134         BD_indicator.itemconfigure(BDlight,
outline="#00E518", fill="#00E518")
135         BD_indicator.coords(BDlight, 8 + 76 + 76, 8, 68 +
76 + 76, 68)
136
137     elif color.lower() == "err":
138         BD_indicator.itemconfigure(BDlight,
outline="gray", fill="gray")
139         BD_indicator.itemconfigure(BDtext, fill="black")
140         BD_indicator.coords(BDtext, 114, 38)
141
142     else:
143         raise ValueError("Incorrect input: color must be
\"red\", \"yellow\", \"#00E518\", \"green\", or \"err\"")
144
145 gui = GUI() # Start the GUI
146
147 mu_tire_road = 0.7 # Coefficient of Friction of a dry road
148
149 # Error codes of the Laser
150 err = {":Er01!": "Power input to low, power voltage should be >=
2.0V.",

```

```

151     ":Er02!": "Internal error, don't care.",
152     ":Er03!": "Module temperature is too low (< -20C).",
153     ":Er04!": "Module temperature is too high (> +40C).",
154     ":Er05!": "Target out of range.",
155     ":Er06!": "Measure result invalid.",
156     ":Er07!": "Background light too strong.",
157     ":Er08!": "Laser signal too weak.",
158     ":Er09!": "Laser signal too strong.",
159     ":Er10!": "Hardware fault 1.",
160     ":Er11!": "Hardware fault 2.",
161     ":Er12!": "Hardware fault 3.",
162     ":Er13!": "Hardware fault 4.",
163     ":Er14!": "Hardware fault 5.",
164     ":Er15!": "Laser signal not stable.",
165     ":Er16!": "Hardware fault 6.",
166     ":Er17!": "Hardware fault 7."
167 }
168
169 # Function to send the command, calculate the values, and print
170 the data
171 def send(string):
172     ser.write(string) # Send command
173
174     x = str(ser.readline()) # Read Laser data
175     print(x)
176
177     # Parse data, ep. 1: there could be an error | can be
178     changed to suit other Lasers
179     potential_err = x[3:9] # Where the error key could be (see
180     err.keys())
181
182     # Check if an error was received
183     flag = False
184     for i in err.keys():
185         if potential_err == i:
186             flag = True
187
188     # If there was an error
189     if flag:

```

```

187         # Parse data, ep. 2: print the error | can be changed
to suit other Lasers
188         actual_err_c = x[2:9] + " " + err.get(x[3:9]) # Error
message printed to console
189         actual_err_g = x[4:9] + " " + err.get(x[3:9]) # Error
message printed to GUI
190
191         # Print error
192         print(actual_err_c) # to console
193         gui.printg(distance, actual_err_g) # to GUI
194
195         # Print braking distance
196         brakingDistance = ((speed.get() ** 2)) / (2 *
mu_tire_road * 9.81) # Calculate
197         print(brakingDistance) # to console
198         gui.printg(braking_distance, "Braking Distance:\n" +
str(brakingDistance) + "m") # to GUI
199
200         # Print three second rule distance
201         threeSDist = speed.get() * 3.0 # Calculate
202         print(threeSDist) # to console
203         gui.printg(threeSHeader, "Three second rule
distance:\n" + str(threeSDist) + "m") # to GUI
204         gui.changeTSRLight("err") # Make TSR_indicator show an
error message
205         gui.changeBDLight("err") # Make BD_indicator show an
error message
206
207         # otherwise
208         else:
209             # Parse data, ep. 3: There was no error | can be
210 changed to suit other Lasers
211             try:
212                 dist_c = x[2:x.index("\\")] # Distance printed to
console
213
214             except BaseException as error:
215                 print("Failed at send().if:else.try_1.\n" +
str(error))

```

```

216
217         try:
218             if x[5:6] == " ":
219                 dist = float(x[5:x.index("m")]) # Actual
distance value as a float
220                 dist_g = "Distance: " + x[5:x.index(",")] #
Distance printed to GUI
221
222             else:
223                 dist = float(x[4:x.index("m")]) # Actual
distance value as a float
224                 dist_g = "Distance: " + x[4:x.index(",")] #
Distance printed to GUI
225
226         except BaseException as error:
227             print("Failed at send().if:else.try_2.\n" +
228                 str(error))
229         try:
230             # Print distance
231             print(dist_c) # to console
232             gui.printg(distance, dist_g) # to GUI
233
234             # Print braking distance
235             brakingDistance = ((speed.get() ** 2)) / (2 *
236                 mu_tire_road * 9.81) # Calculate
237             print(brakingDistance) # to console
238             gui.printg(braking_distance, "Braking Distance:\n"
239                 + str(brakingDistance) + "m") # to GUI
240
241             # Print three second rule distance
242             threeSDist = speed.get() * 3.0 # Calculate
243             print(threeSDist) # to console
244             gui.printg(threeSHeader, "Three second rule
245                 distance:\n" + str(threeSDist) + "m") # to GUI

```

```

246         gui.changeTSRLight("green")
247
248         elif dist <= threeSDist + 10.0 and dist >
threeSDist:
249             gui.changeTSRLight("yellow")
250
251         else:
252             gui.changeTSRLight("red")
253
254         # Change BD_indicator depending on how close you
are to the next car
255         if dist > brakingDistance + 10.0:
256             gui.changeBDLight("green")
257
258         elif dist <= brakingDistance + 10.0 and dist >
brakingDistance:
259             gui.changeBDLight("yellow")
260
261         else:
262             gui.changeBDLight("red")
263
264     except BaseException as error:
265         print("Failed at send().if:else.try_3.\n" +
str(error))
266
267 # Dictionary of Laser commands
268 commands = {"laser_on": [0x4F], # Hex value of 0
269             "laser_off": [0x43], # Hex value of C
270             "laser_get_dist": [0x44], # Hex value of D
271             "laser_get_temp": [0x53] # Hex value of S
272             }
273 send(commands.get("laser_on")) # Turn Laser on
274 while 1:
275     send(commands.get("laser_get_dist")) # Continuously get the
measured distance

```

---

The initial design of the GUI displayed the current velocity, the distance measured by



the laser, braking distance, 3-Second Rule distance, two separate indicators for the last two distances. The indicators are modeled after a stoplight, and each light on the stoplight corresponds to how the measured distance ( $d$ ) compares to the calculated braking and 3-Second Rule distances ( $b$ ). The indicator is red when  $d < b$ , yellow when  $b + 10 \text{ meters} \geq d > b$  and, green when  $d > b + 10 \text{ meters}$ , and finally it displays “Error, Proceed with caution” when no data is being sent by the laser. (Fig. 5). At this point, everything is updated live according to the distance measured by the laser, but the current speed is handled by a Tkinter Scale Widget.

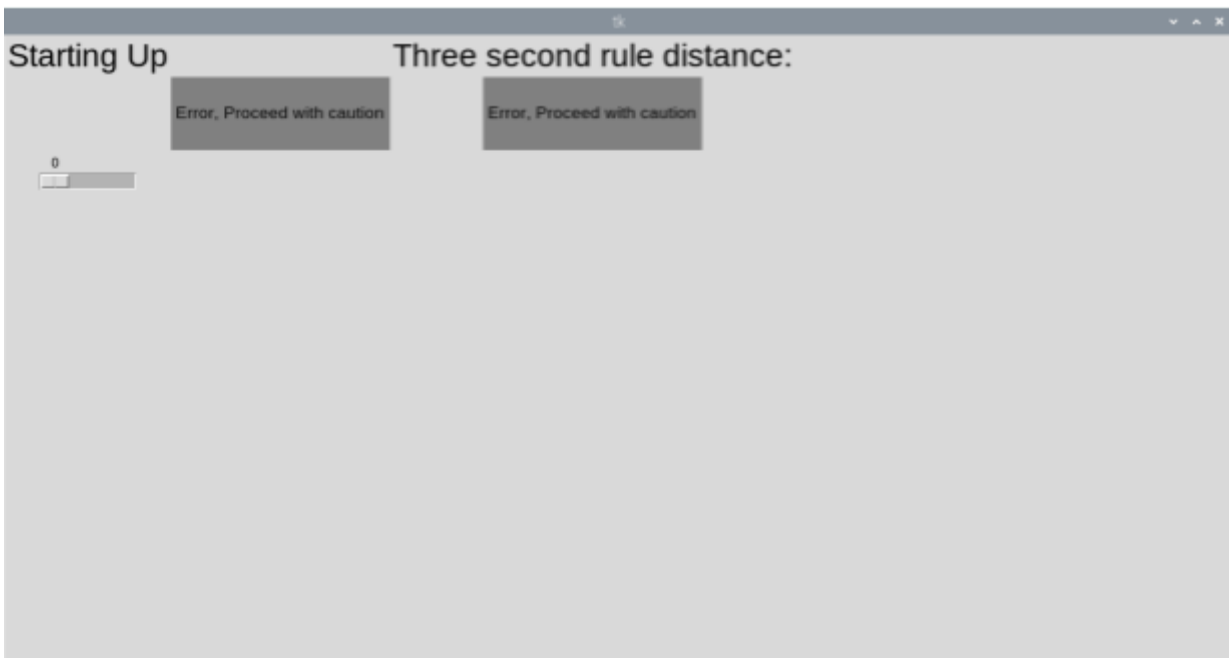


Figure 5: First iteration of the GUI. Notice the slider at the left that simulates the speed of the car.

### *Second Iteration*

There are a few flaws with the initial design of the device. First, the laser is incredibly slow, and its wires are much too short to be usable. Second, the laser is extremely unreliable and does not have the range needed for this application. Three, Tkinter is being run in its own



---

```
1  from numpy import ceil # Import the ceil() function from NumPy
2  import serial # Import the python serial package
3  import obd # Import the python-OBD package
4  from tkinter import * # Import Tkinter
5
6  connection = obd.OBD("/dev/ttyUSB2") # Connect to the OBD-II to
   USB adapter connected to the USB2 port
7
8  # Connect to the USB Lidar connected to the USB1 port
9  ser = serial.Serial(
10     port='/dev/ttyUSB1',
11     baudrate = 115200,
12     parity=serial.PARITY_NONE,
13     stopbits=serial.STOPBITS_ONE,
14     bytesize=serial.EIGHTBITS,
15     timeout=1
16 )
17
18 # Set up Tkinter
19 root = Tk()
20
21 guiFactor = 1.6 # Used to scale the entire gui by 1.6 times to
   fit the screen
22
23 width = root.winfo_screenwidth() # Get width of screen
24 height = root.winfo_screenheight() # Get height of screen
25
26 root.geometry("%dx%d" % (width, height)) # Set size of window to
   fullscreen
27
28 # Set up Distance text
29 distance = Label(root, text="Starting Up", font=("Arial",
   int(ceil(25 * guiFactor))), padx=5, pady=5)
30 distance.grid(row=0, column=0)
31
32 # Set up speed header
33 speed = Label(root, text="Starting Up", font=("Arial",
```

```

    int(ceil(25 * guiFactor))))
34 speed.grid(row=0, column=1, padx=5, pady=5)
35
36 # Set up Braking Distance Text
37 braking_distance = Label(root, text=" ", font=("Arial",
    int(ceil(25 * guiFactor))), padx=5, pady=5)
38 braking_distance.grid(row=1, column=1, padx=5, pady=5)
39
40 # Braking Distance indicator
41 BD_indicator = Canvas(root, height= int(ceil(76 * guiFactor)),
    width= int(ceil(228 * guiFactor)), bg="gray")
42 BD_indicator.grid(row=2, column=1, padx=5, pady=5)
43
44 BDlight = BD_indicator.create_oval(int(ceil(8 * guiFactor)),
    int(ceil(8 * guiFactor)), int(ceil(68 * guiFactor)), int(ceil(68
    * guiFactor))), outline="gray", fill="gray")
45 BDtext = BD_indicator.create_text( int(ceil(114 * guiFactor)),
    int(ceil(114 * guiFactor)), font=("Arial", int(ceil(13 *
    guiFactor)))), text="Error, Proceed with caution")
46
47 # 3 second rule header
48 threeSHeader = Label(root, text="Three second rule distance:",
    font=("Arial", int(ceil(25 * guiFactor))))
49 threeSHeader.grid(row=1, column=0, padx=5, pady=5)
50
51 # 3 second rule indicator
52 TSR_indicator = Canvas(root, height= int(ceil(76 * guiFactor)),
    width= int(ceil(228 * guiFactor)), bg="gray")
53 TSR_indicator.grid(row=2, column=0, padx=5, pady=5)
54
55 # create the TSR stop light
56 TSRlight = TSR_indicator.create_oval(int(ceil(8 * guiFactor)),
    int(ceil(8 * guiFactor)), int(ceil(68 * guiFactor)), int(ceil(68
    * guiFactor))), outline="gray", fill="gray")
57 TSRtext = TSR_indicator.create_text(114, 38, font=("Arial",
    int(ceil(13 * guiFactor)))), text="Error, Proceed with caution")
58
59 # print to GUI as printg(widget, new_text)
60 def printg(widget, string):

```

```

61         widget.config(text=string)
62
63     # Change the options of the canvas oval item
64     # red = "red", yellow = "yellow", green = "#00E518"
65     def changeTSRLight(color):
66         if color.lower() == "red":
67             TSR_indicator.itemconfigure(TSRtext, fill="gray")
68             TSR_indicator.coords(TSRtext, 0, 0)
69             TSR_indicator.itemconfigure(TSRlight,
outline=color.lower(), fill=color.lower())
70             TSR_indicator.coords(TSRlight, int(ceil(8 *
guiFactor)), int(ceil(8 * guiFactor)), int(ceil(68 *
guiFactor)), int(ceil(68 * guiFactor)))
71
72         elif color.lower() == "yellow":
73             TSR_indicator.itemconfigure(TSRtext, fill="gray")
74             TSR_indicator.coords(TSRtext, 0, 0)
75             TSR_indicator.itemconfigure(TSRlight,
outline=color.lower(), fill=color.lower())
76             TSR_indicator.coords(TSRlight, int(ceil((8 *
guiFactor) + (76 * guiFactor))), int(ceil(8 * guiFactor)),
int(ceil((68 * guiFactor) + (76 * guiFactor))), int(ceil(68 *
guiFactor)))
77
78         elif color.lower() == "#00E518" or color.lower() ==
"green":
79             TSR_indicator.itemconfigure(TSRtext, fill="gray")
80             TSR_indicator.coords(TSRtext, 0, 0)
81             TSR_indicator.itemconfigure(TSRlight,
outline="#00E518", fill="#00E518")
82             TSR_indicator.coords(TSRlight, int(ceil((8 *
guiFactor) + (76 * guiFactor) + (76 * guiFactor))), int(ceil(8 *
guiFactor)), int(ceil((68 * guiFactor) + (76 * guiFactor) + (76
* guiFactor))), int(ceil(68 * guiFactor)))
83
84         elif color.lower() == "err":
85             TSR_indicator.itemconfigure(TSRlight, outline="gray",
fill="gray")
86             TSR_indicator.itemconfigure(TSRtext, fill="black")

```

```

87         TSR_indicator.coords(TSRtext, int(ceil(114 *
guiFactor))), int(ceil(38 * guiFactor)))
88
89     else:
90         raise ValueError("Incorrect input: color must be
\"red\", \"yellow\", \"#00E518\", \"green\", or \"err\"")
91
92 def changeBDLight(color):
93     if color.lower() == "red":
94         BD_indicator.itemconfigure(BDtext, fill="gray")
95         BD_indicator.coords(BDtext, 0, 0)
96         BD_indicator.itemconfigure(BDlight,
outline=color.lower(), fill=color.lower())
97         BD_indicator.coords(BDlight, int(ceil(8 *
guiFactor))), int(ceil(8 * guiFactor))), int(ceil(68 *
guiFactor))), int(ceil(68 * guiFactor)))
98
99     elif color.lower() == "yellow":
100         BD_indicator.itemconfigure(BDtext, fill="gray")
101         BD_indicator.coords(BDtext, 0, 0)
102         BD_indicator.itemconfigure(BDlight,
outline=color.lower(), fill=color.lower())
103         BD_indicator.coords(BDlight, int(ceil((8 * guiFactor)
+ (76 * guiFactor))), int(ceil(8 * guiFactor))), int(ceil((68 *
guiFactor) + (76 * guiFactor))), int(ceil(68 * guiFactor)))
104
105     elif color.lower() == "#00E518" or color.lower() ==
"green":
106         BD_indicator.itemconfigure(BDtext, fill="gray")
107         BD_indicator.coords(BDtext, 0, 0)
108         BD_indicator.itemconfigure(BDlight,
outline="#00E518", fill="#00E518")
109         BD_indicator.coords(BDlight, int(ceil((8 * guiFactor)
+ (76 * guiFactor) + (76 * guiFactor))), int(ceil(8 *
guiFactor))), int(ceil((68 * guiFactor) + (76 * guiFactor) + (76
* guiFactor))), int(ceil(68 * guiFactor)))
110
111     elif color.lower() == "err":
112         BD_indicator.itemconfigure(BDlight, outline="gray",

```

```

        fill="gray")
113         BD_indicator.itemconfigure(BDtext, fill="black")
114         BD_indicator.coords(BDtext, int(ceil(114 *
guiFactor)), int(ceil(38 * guiFactor)))
115
116     else:
117         raise ValueError("Incorrect input: color must be
        \"red\", \"yellow\", \"#00E518\", \"green\", or \"err\"")
118
119 mu_tire_road = 0.7 # Coefficient of Friction of a dry road
120
121 # Send OBD command and parse the response
122 try:
123     velocity =
        connection.query(obd.commands.SPEED).value.to("m/s")
124     velocity = float(velocity.split(" ")[0])
125     printg(speed, "Current speed: " + format(velocity, ".2f")
        + "m/s") # to GUI
126 except:
127     printg(speed, "Current speed: " + format(velocity, ".2f")
        + "m/s") # to GUI
128
129 # Function to parse the data from the lidar, calculate the
130 values and print the data
131 def send():
132     # Get the distance from the lidar
133     dist = 0.0
134     TFbuff = [0,0,0,0,0,0,0,0,0]
135     checksum = 0
136     TFbuff[0] = ser.read() # Read Laser data
137     if TFbuff[0] == b'Y':
138         TFbuff[1] = ser.read()
139         if TFbuff[1] == b'Y':
140             for i in range(2,8):
141                 TFbuff[i] = ser.read()
142             TFbuff[8] = ser.read()
143             if checksum == 0:
144                 dist = int.from_bytes(TFbuff[2],"big") +
int.from_bytes(TFbuff[3],"big")*256 - 7

```

```

145             dist /= 100
146         print(dist)
147
148         # If there was an error
149         if dist > 170.00:
150             # Parse data, ep. 1: print the error | can be changed
to suit other Lasers
151
152             # Print error
153             printg(distance, "Distance: Error") # to GUI
154
155             # Print braking distance
156             brakingDistance = ((velocity ** 2)) / (2 *
mu_tire_road * 9.81) # Calculate
157             printg(braking_distance, "Braking Distance:\n" +
format(brakingDistance, ".2f") + "m") # to GUI
158
159             # Print three second rule distance
160             threeSDist = velocity * 3.0 # Calculate
161             printg(threeSHeader, "Three second rule distance:\n"
+ str(threeSDist) + "m") # to GUI
162
163             changeTSRLight("err") # Make TSR_indicator show an
error message
164             changeBDLight("err") # Make BD_indicator show an
error message
165
166             # otherwise
167         else:
168             # Parse data, ep. 2: There was no error | can be
changed to suit other Lasers
169             dist_g = "Distance: " + format(dist, ".2f") + "m" #
Distance printed to GUI
170
171             # Print distance
172             printg(distance, dist_g) # to GUI
173
174             # Print braking distance
175             brakingDistance = ((velocity ** 2)) / (2 *

```



```

mu_tire_road * 9.81) # Calculate
176     printg(braking_distance, "Braking Distance:\n" +
format(brakingDistance, ".2f") + "m") # to GUI
177
178     # Print three second rule distance
179     threeSDist = velocity * 3.0 # Calculate
180     printg(threeSHeader, "Three second rule distance:\n"
+ format(threeSDist, ".2f") + "m") # to GUI
181
182     # Change TSR_indicator depending on how close you are
to the next car
183     if dist > threeSDist + 10:
184         changeTSRLight("green")
185
186     elif dist <= threeSDist + 10.0 and dist > threeSDist:
187         changeTSRLight("yellow")
188
189     else:
190         changeTSRLight("red")
191
192     # Change BD_indicator depending on how close you are
to the next car
193     if dist > brakingDistance + 10.0:
194         changeBDLight("green")
195
196     elif dist <= brakingDistance + 10.0 and dist >
brakingDistance:
197         changeBDLight("yellow")
198
199     else:
200         changeBDLight("red")
201
202     root.after(5, send)
203
204 # Constantly get the measured distance, calculate the data, and
update the GUI
205 # Send OBD command and parse the response
206 try:
    velocity =

```

```

        connection.query(obd.commands.SPEED).value.to("m/s")
207         velocity = float(velocity.split(" ")[0])
208         printg(speed, "Current speed: " + format(velocity, ".2f")
+ "m/s") # to GUI
209 except:
210     printg(speed, "Current speed: " + format(velocity, ".2f")
+ "m/s") # to GUI
211
212 root.after(0, send)
213 root.mainloop() # Start event loop

```

---

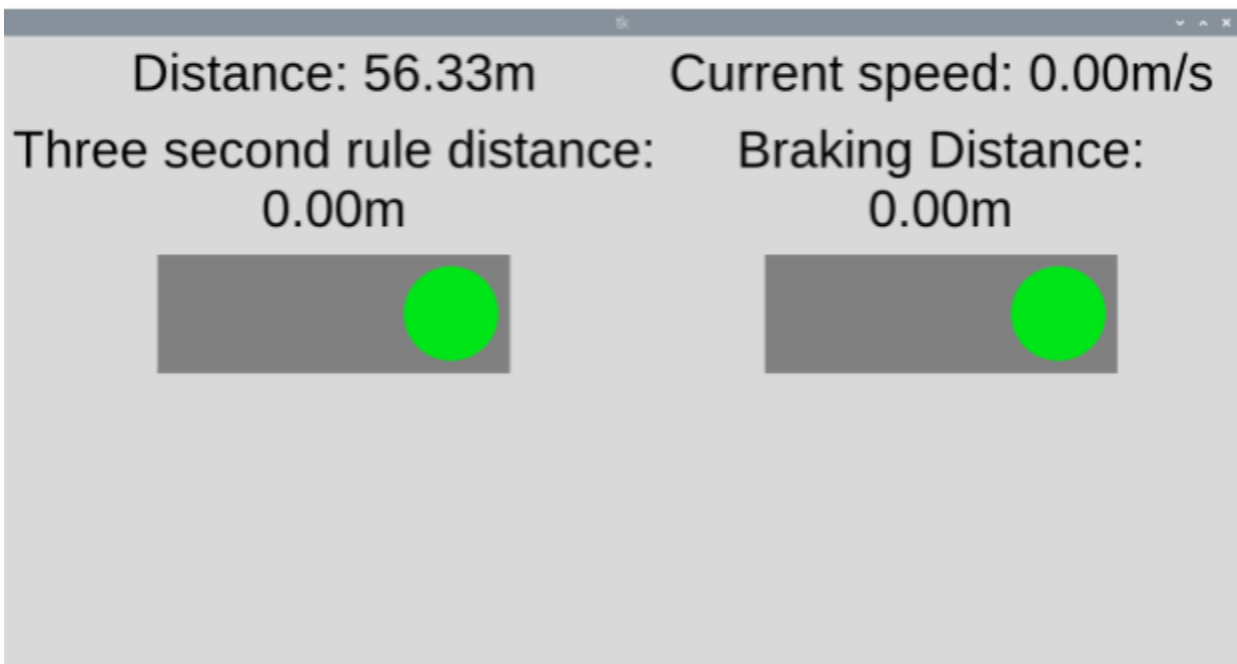


Figure 7: Second iteration of the GUI. Notice how speed is no longer a slider and the text is larger and centered

There were a few challenges faced while getting the code to work as intended. One was that the names of the USB ports had a chance to change every time a USB port was reconnected or when the Raspberry Pi booted. This ended up causing the code to be unable to connect to the Peripherals. This was fixed by looping through all of the possible names where the peripheral

could be connected. The second challenge was parsing the data from the LiDAR. The data sent back by the LiDAR is encoded in pure hexadecimal. This would not have been an issue because most devices have documentation describing how to parse the sent data. In fact, for this specific model, there were C++ and Python libraries for this specific model that would parse the data automatically. However, the Python library was nonexistent online and could not be imported. The C++ library (Cesar-S, 2019) can be found on GitHub, a platform where developers can host their code for collaboration and version control. Therefore the workaround was to rewrite the code written in C++ found in the repository into Python that would be implemented directly into the code.

The last challenge was making sure that the GUI was the correct size. This was fixed by scaling each Tkinter widget up by the same factor. The only issue is that Tkinter does not allow floats (numbers with decimals) to be passed into the functions being used by the code. This is fixed by importing the `ceil()` function from NumPy using `from numpy import ceil` alongside the `int()` function. After some trial and error, a factor of 1.6 is used to scale the GUI up. The `ceil()` function rounded the product up, and then the `int()` function was used to ensure that the number did not have a decimal ending by truncating the decimal part. This is because the `ceil()` function only rounds the number up and does not convert (typecast) the number to an integer; therefore, the `int()` function must be used to ensure that the value is typecasted to an integer before use, i.e., `int(1.0)` returns 1. The full use of the functions is `int(ceil(original_num * gui_factor))`.

### *Final Iteration*

The final thing that needs to be done is to integrate the Raspberry Pi, OBD-II, LiDAR, and TFT into a single, easy-to-use unit. Therefore, a holder was designed in AutoDesk Onshape to hold the Raspberry Pi and the TFT. In addition, a simple HUD must be designed to reflect the image into the driver's field of view. Finally, it was found while testing the second iteration in a moving vehicle that the LiDAR has very poor performance inside the car, pointing out the windshield while it works perfectly outside the vehicle. Therefore, the solution is to duct tape the LiDAR to the front of the car and compensate for the distance between the front of the car and the sensor. To do this, a USB extension cord was used because the LiDAR's mini-USB cable was not long enough to reach outside the car (Fig. 8).

The case of the whole assembly is 3D-printed out of PLA. It is designed with two features in mind. The display area is transparent, and the image can be seen during the day. Its design is based on two prior works by the Always Be Curious (ABCyt) Youtube Channel, "DIY: Making a Car HUD display" (Seth, 2019) and The Instructables Workshop "Holographic Car Heads-Up Display (HUD)" (Jain, 2019). The design implements the use of a Projection Film used by the ABCyt. However, their design places the film on the windshield, which is not ideal. Therefore, the assembly uses a clear piece of 309.42 x 181.92 x 6.35mm laser-cut acrylic at a 50° angle inspired by the use of a CD case by The Instructables workshop. See the technical drawing for more details.



Figure 8: Expanded view of all the wires and peripherals connected to the Raspberry Pi with the 3D printed Assembly Holder.

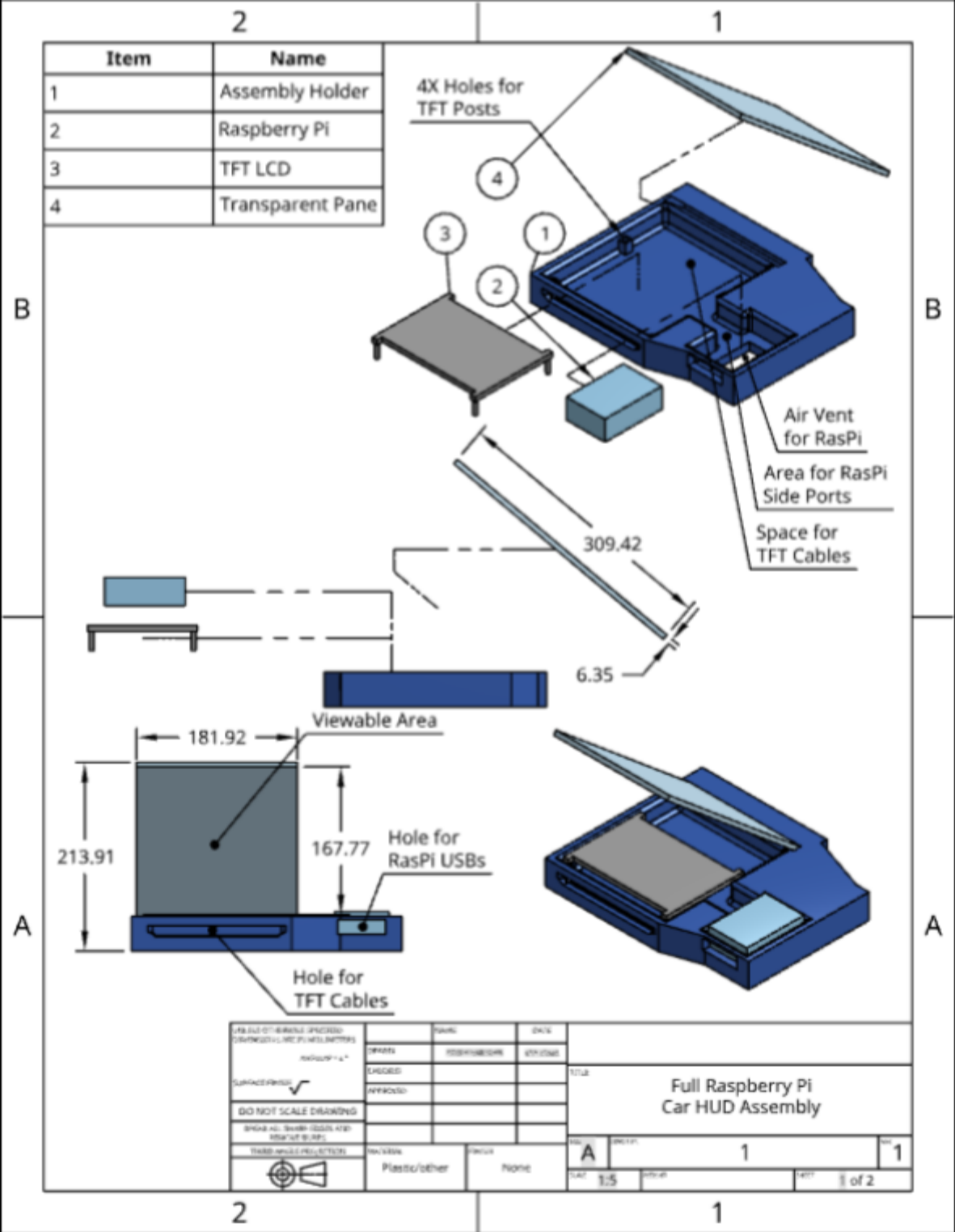


Figure 9: Page 1 of the technical drawing of Holder

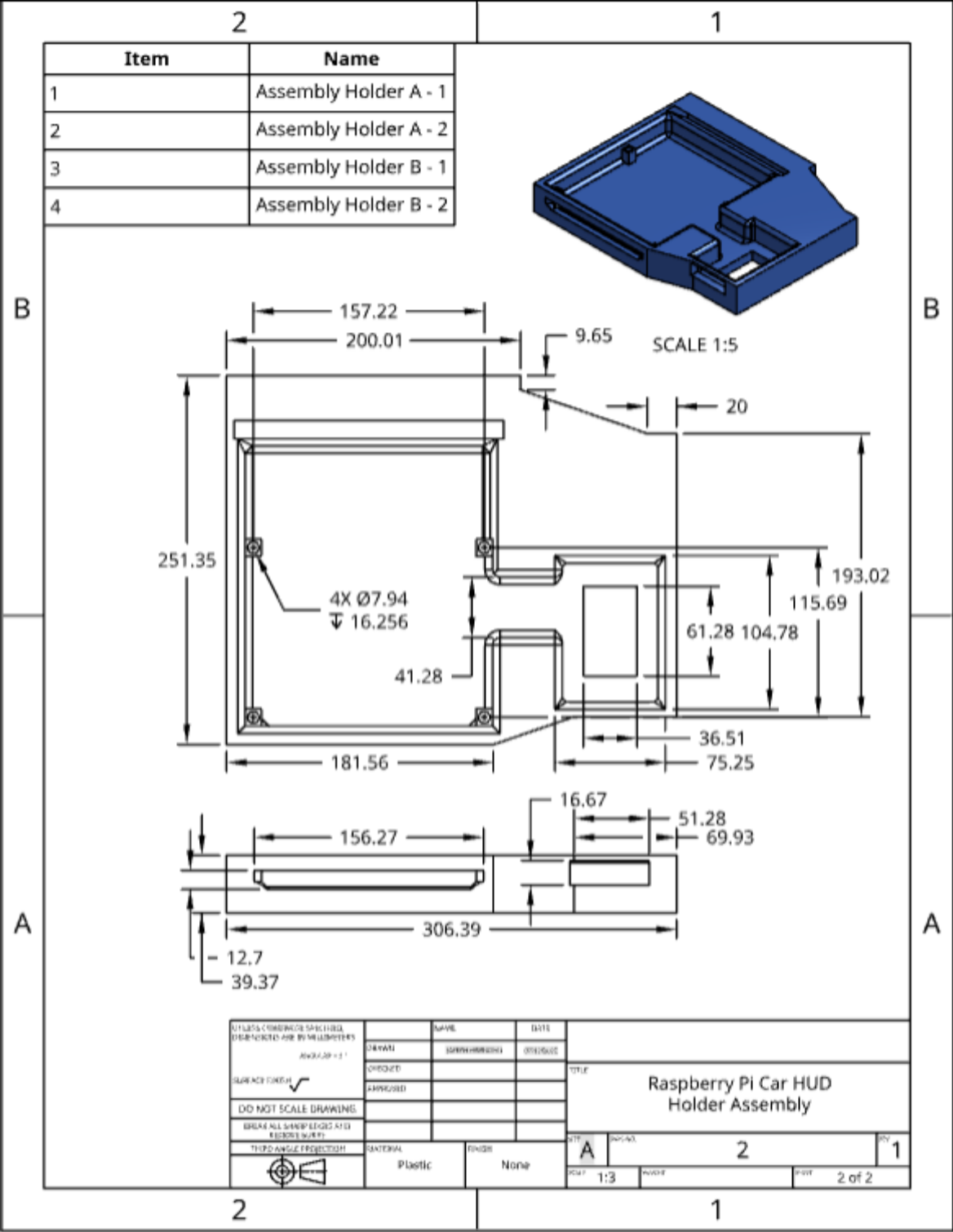


Figure 10: Page 2 of the technical drawing of Holder

The next problem that needs to be solved is the screen's orientation. For the TFT to fit in the holder mentioned above, it needs the HDMI and power port facing the windshield. This is because space is added to allow the ends of the cables to fit; see technical sheet 1 (Fig. 9). The only issue is that the image will appear upside down to the driver. The fix is to use the Linux Operating System of the Raspberry Pi, Raspbian, to flip the screen upside down on the TFT. To do this, edit the `/boot/cmdline.txt` file. In the terminal, run

**sudo nano** `/boot/cmdline.txt`. At the very end of the line that is already there, add `video=HDMI-A-N:WxY@F,reflect_y` (Fig. 14), where N is the HDMI port number (1 is



Figure 11: How to find Screen Configuration on a Raspberry Pi to find the screen's resolution and frequency.

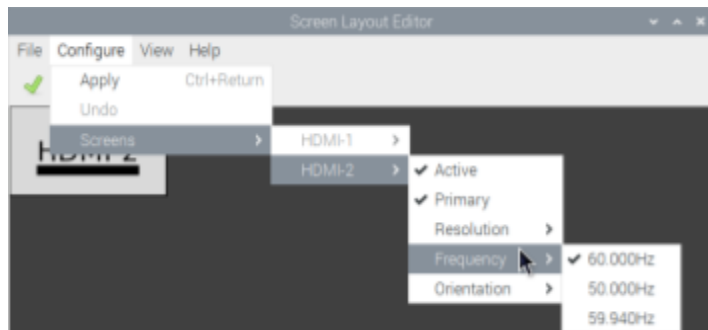


Figure 12: Finding the frequency of a screen connected to Raspberry Pi in Screen Configurations. Note how the checkmark is at 60.000Hz.

closer to the power port and 2 is the other mini-HDMI port), `WxY` is the screen resolution, and `F` is the frequency of the screen, see Figures 11, 12, and 13 to see how to find the screen resolution and the framerate.

Next, the text displayed on the GUI must be shifted down to put it in the right location on the Transparent Pane (Fig. 15). This is done very easily by adding a few lines of text to a new label widget and setting its color to the color of the background.



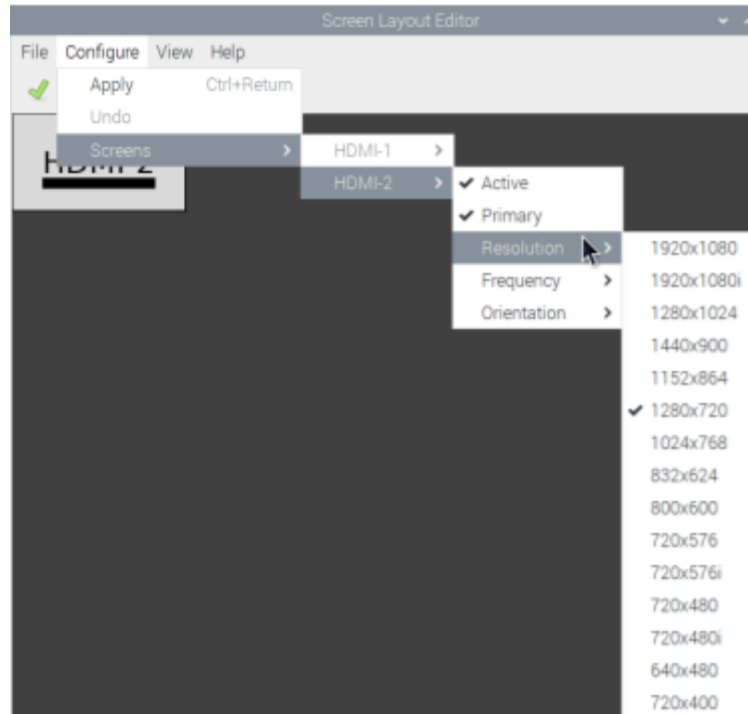


Figure 13: Finding the resolution of a screen connected to Raspberry Pi in Screen Configurations. Note how the checkmark is at 1280x720.

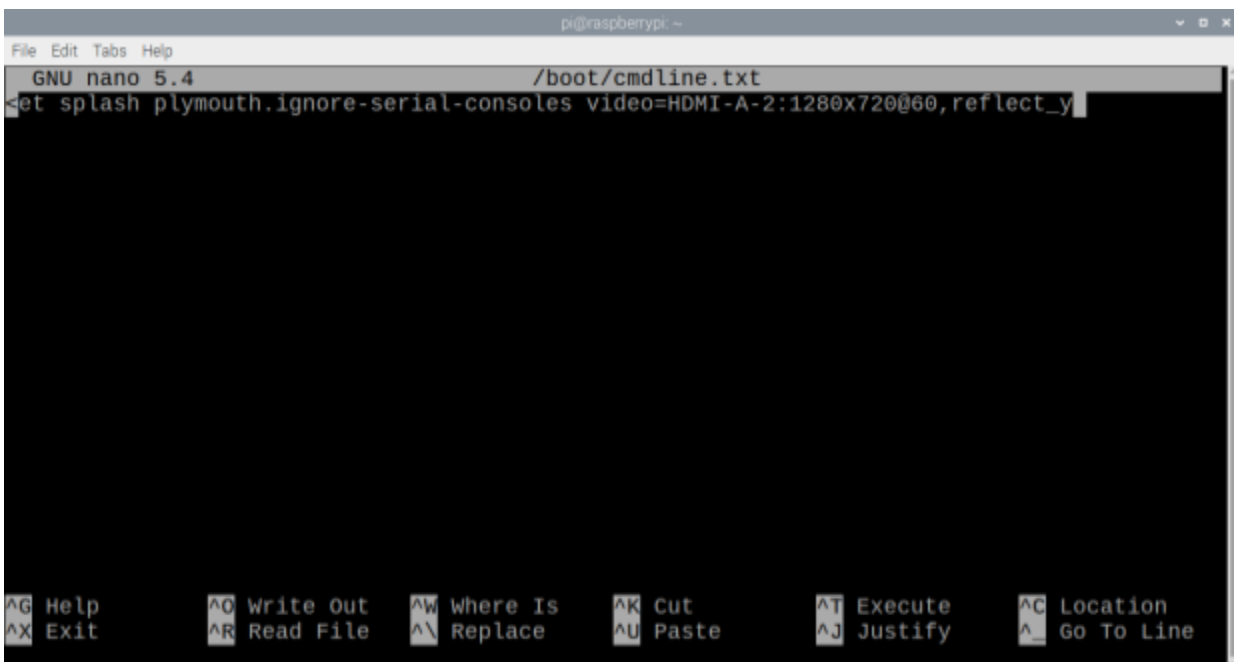


Figure 14: The /boot/cmdline.txt set to have the screen to flip the screen upside down (see how video=... is appended to the end of the line).

---

```
1  from numpy import ceil # Import the ceil() function from numpy
2  import threading as thread # Import the Python threading
   package and name it thread
3  import serial #Import Python Serial package
4  import obd # Import the python-OBD package
5  from tkinter import * # Import Tkinter
6
7  # Try all USB ports until the LiDAR connects
8  j = None
9  for i in range(5):
10     try:
11         ser = serial.Serial(
12             port='/dev/ttyUSB' + str(i),
13             baudrate = 115200,
14             parity=serial.PARITY_NONE,
15             stopbits=serial.STOPBITS_ONE,
16             bytesize=serial.EIGHTBITS,
17             timeout=1
18         )
19         print(ser.read())
20         if ser.read() == b'Y':
21             j = i
22             break
23     except:
24         print("Wrong Port")
25
26  # Try all USB ports until the OBD-II connects
27  for i in range(5):
28     if i == j:
29         continue
30
31     connection = obd.OBD("/dev/ttyUSB" + str(i))
32     print(connection.is_connected())
33     if connection.is_connected():
34         break
35
36  if not connection.is_connected():
```

```

37     velocity = 0.0
38 # Set up Tkinter
39 root = Tk()
40
41 guiFactor = 1.6 # Used to scale the entire gui by 1.6 times to
    fit the screen
42
43 # Set the size of the window to fullscreen
44 width = root.winfo_screenwidth() # Get width of screen
45 height = root.winfo_screenheight() # Get height of screen
46
47 root.geometry("%dx%d" % (width, height)) # Set size of window
    to fullscreen
48
49 # Padding
50 padding = Label(root, text="|\n|\n|\n|\n|\n|", font=("Arial",
    30), fg='gainsboro')
51 padding.grid(row=0, column=0)
52
53 # Set up Distance text
54 distance = Label(root, text="Starting Up", font=("Arial",
    int(ceil(25 * guiFactor))), padx=5, pady=5)
55 distance.grid(row=1, column=0)
56
57 # Set up speed header
58 speed = Label(root, text="Starting Up", font=("Arial",
    int(ceil(25 * guiFactor))))
59 speed.grid(row=1, column=1, padx=5, pady=5)
60
61 # Set up Braking Distance Text
62 braking_distance = Label(root, text=" ", font=("Arial",
    int(ceil(25 * guiFactor))), padx=5, pady=5)
63 braking_distance.grid(row=2, column=1, padx=5, pady=5)
64
65 # Braking Distance indicator
66 BD_indicator = Canvas(root, height= int(ceil(76 * guiFactor)),
    width= int(ceil(228 * guiFactor)), bg="gray")
67 BD_indicator.grid(row=3, column=1, padx=5, pady=5)
68

```

```

69 # Create the BD stop light
70 BDlight = BD_indicator.create_oval(int(ceil(8 * guiFactor)),
    int(ceil(8 * guiFactor)), int(ceil(68 * guiFactor)),
    int(ceil(68 * guiFactor)), outline="gray", fill="gray")
71 BDtext = BD_indicator.create_text(int(ceil(114 * guiFactor)),
    int(ceil(38 * guiFactor)), font=("Arial", int(ceil(13 *
    guiFactor)))), text="Error, Proceed with caution")
72
73 # 3 second rule header
74 threeSHeader = Label(root, text="Three second rule distance:",
    font=("Arial", int(ceil(25 * guiFactor))))
75 threeSHeader.grid(row=2, column=0, padx=5, pady=5)
76
77 # 3 second rule indicator
78 TSR_indicator = Canvas(root, height= int(ceil(76 * guiFactor)),
    width= int(ceil(228 * guiFactor)), bg="gray")
79 TSR_indicator.grid(row=3, column=0, padx=5, pady=5)
80
81 # Create the TSR stop light
82 TSRlight = TSR_indicator.create_oval(int(ceil(8 * guiFactor)),
    int(ceil(8 * guiFactor)), int(ceil(68 * guiFactor)),
    int(ceil(68 * guiFactor)), outline="gray", fill="gray")
83 TSRtext = TSR_indicator.create_text(int(ceil(114 * guiFactor)),
    int(ceil(38 * guiFactor)), font=("Arial", int(ceil(13 *
    guiFactor)))), text="Error, Proceed with caution")
84
85 # print to GUI as printg(widget, new_text)
86 def printg(widget, string):
87     widget.config(text=string)
88
89 # Change the options of the canvas oval item
90 # red = "red", yellow = "yellow", green = "#00E518"
91 def changeTSRLight(color):
92     if color.lower() == "red":
93         TSR_indicator.itemconfigure(TSRtext, fill="gray")
94         TSR_indicator.coords(TSRtext, 0, 0)
95         TSR_indicator.itemconfigure(TSRlight,
    outline=color.lower(), fill=color.lower())
96         TSR_indicator.coords(TSRlight, int(ceil(8 *

```

```

guiFactor)), int(ceil(8 * guiFactor)), int(ceil(68 *
guiFactor)), int(ceil(68 * guiFactor)))
97
98     elif color.lower() == "yellow":
99         TSR_indicator.itemconfigure(TSRtext, fill="gray")
100        TSR_indicator.coords(TSRtext, 0, 0)
101        TSR_indicator.itemconfigure(TSRlight,
outline=color.lower(), fill=color.lower())
102        TSR_indicator.coords(TSRlight, int(ceil((8 * guiFactor)
+ (76 * guiFactor))), int(ceil(8 * guiFactor)), int(ceil((68 *
guiFactor) + (76 * guiFactor))), int(ceil(68 * guiFactor)))
103
104     elif color.lower() == "#00E518" or color.lower() ==
"green":
105         TSR_indicator.itemconfigure(TSRtext, fill="gray")
106         TSR_indicator.coords(TSRtext, 0, 0)
107         TSR_indicator.itemconfigure(TSRlight,
outline="#00E518", fill="#00E518")
108         TSR_indicator.coords(TSRlight, int(ceil((8 * guiFactor)
+ (76 * guiFactor) + (76 * guiFactor))), int(ceil(8 *
guiFactor)), int(ceil((68 * guiFactor) + (76 * guiFactor) + (76
* guiFactor))), int(ceil(68 * guiFactor)))
109
110     elif color.lower() == "err":
111         TSR_indicator.itemconfigure(TSRlight, outline="gray",
fill="gray")
112         TSR_indicator.itemconfigure(TSRtext, fill="black")
113         TSR_indicator.coords(TSRtext, int(ceil(114 *
guiFactor)), int(ceil(38 * guiFactor)))
114
115     else:
116         raise ValueError("Incorrect input: color must be
\"red\", \"yellow\", \"#00E518\", \"green\", or \"err\"")
117
118 def changeBDLight(color):
119     if color.lower() == "red":
120         BD_indicator.itemconfigure(BDtext, fill="gray")
121         BD_indicator.coords(BDtext, 0, 0)
122         BD_indicator.itemconfigure(BDlight,

```

```

outline=color.lower(), fill=color.lower())
123     BD_indicator.coords(BDlight, int(ceil(8 * guiFactor)),
int(ceil(8 * guiFactor)), int(ceil(68 * guiFactor)),
int(ceil(68 * guiFactor)))
124
125     elif color.lower() == "yellow":
126         BD_indicator.itemconfigure(BDtext, fill="gray")
127         BD_indicator.coords(BDtext, 0, 0)
128         BD_indicator.itemconfigure(BDlight,
outline=color.lower(), fill=color.lower())
129         BD_indicator.coords(BDlight, int(ceil((8 * guiFactor) +
(76 * guiFactor))), int(ceil(8 * guiFactor)), int(ceil((68 *
guiFactor) + (76 * guiFactor))), int(ceil(68 * guiFactor)))
130
131     elif color.lower() == "#00E518" or color.lower() ==
"green":
132         BD_indicator.itemconfigure(BDtext, fill="gray")
133         BD_indicator.coords(BDtext, 0, 0)
134         BD_indicator.itemconfigure(BDlight, outline="#00E518",
fill="#00E518")
135         BD_indicator.coords(BDlight, int(ceil((8 * guiFactor) +
(76 * guiFactor) + (76 * guiFactor))), int(ceil(8 *
guiFactor)), int(ceil((68 * guiFactor) + (76 * guiFactor) + (76
* guiFactor))), int(ceil(68 * guiFactor)))
136
137     elif color.lower() == "err":
138         BD_indicator.itemconfigure(BDlight, outline="gray",
fill="gray")
139         BD_indicator.itemconfigure(BDtext, fill="black")
140         BD_indicator.coords(BDtext, int(ceil(114 *
guiFactor)), int(ceil(38 * guiFactor)))
141
142     else:
143         raise ValueError("Incorrect input: color must be
\"red\", \"yellow\", \"#00E518\", \"green\", or \"err\"")
144
145 mu_tire_road = 0.7 # Coefficient of Friction of a dry road
146
147 # Send OBD command and parse the response

```

```

148 velocity = connection.query(obd.commands.SPEED).value.to("m/s")
149 velocity = float(str(velocity).split(" ")[0])
150 printg(speed, "Current speed: " + format(velocity, ".2f") +
    "m/s") # Print to gui
151
152 # Function to parse the data from the lidar, calculate the
153 values and print the data
154 def send():
155     global velocity
156
157     #Get the distance from the lidar
158     dist = 0
159     TFbuff = [0,0,0,0,0,0,0,0,0]
160     TFbuff[0] = ser.read() # Read Laser data
161     if TFbuff[0] == b'Y':
162         TFbuff[1] = ser.read()
163         if TFbuff[1] == b'Y':
164             for i in range(2,8):
165                 TFbuff[i] = ser.read()
166                 TFbuff[8] = ser.read()
167                 dist = int.from_bytes(TFbuff[2],"big") +
int.from_bytes(TFbuff[3],"big")*256 - 7
168                 dist /= 100
169
170         # If there was an error
171         if dist > 170.00:
172             # Parse data, ep. 1: print the error | can be changed
to suit other lasers
173
174             # Print error
175             printg(distance, "Distance: Error") # to GUI
176
177             # Print braking distance
178             printg(braking_distance, "Braking Distance:\n" +
format(((velocity ** 2)) / (2 * mu_tire_road * 9.81), ".2f") +
    "m") # to GUI
179
180             # Print three second rule distance
181             printg(threeSHeader, "Three second rule distance:\n" +

```

```

format(velocity * 3.0, ".2f") + "m") # to GUI
182
183     changeTSRLight("err") # Make TSR_indicator show an
error message
184     changeBDLight("err") # Make BD_indicator show an error
message
185
185     # otherwise
186     else:
187         # Parse data, ep. 2: There was no error | can be
changed to suite other
188
188         # Print distance
189         printg(distance, "Distance: " + format(dist, ".2f") +
"m") # to GUI
191
191         # Print braking distance
192         brakingDistance = ((velocity ** 2)) / (2 * mu_tire_road
* 9.81) # Calculate
193         printg(braking_distance, "Braking Distance:\n" +
format(brakingDistance, ".2f") + "m") # to GUI
195
195         # Print three second rule distance
196         threeSDist = velocity * 3.0 # Calculate
197         printg(threeSHeader, "Three second rule distance:\n" +
format(threeSDist, ".2f") + "m") # to GUI
199
199         # Change TSR_indicator depending on how close you are
to the next car
200         if dist > threeSDist + 10:
201             changeTSRLight("green")
202
203         elif dist <= threeSDist + 10.0 and dist > threeSDist:
204             changeTSRLight("yellow")
205
206         else:
207             changeTSRLight("red")
208
209         # Change BD_indicator depending on how close you are to
210

```



```

the next car
211         if dist > brakingDistance + 10.0:
212             changeBDLight("green")
213
214         elif dist <= brakingDistance + 10.0 and dist >
brakingDistance:
215             changeBDLight("yellow")
216
217         else:
218             changeBDLight("red")
219
220     #Get the distance form the lidar
221     #dist = 0
222     #TFbuff = [0,0,0,0,0,0,0,0,0,0]
223     TFbuff[0] = ser.read() # Read Laser data
224     if TFbuff[0] == b'Y':
225         TFbuff[1] = ser.read()
226         if TFbuff[1] == b'Y':
227             for i in range(2,8):
228                 TFbuff[i] = ser.read()
229                 TFbuff[8] = ser.read()
230                 dist = int.from_bytes(TFbuff[2],"big") +
int.from_bytes(TFbuff[3],"big")*256 - 7
231                 dist /= 100
232                 dist += 1.15
233
234     # Send OBD command and parse the response
235     velocity =
connection.query(obd.commands.SPEED).value.to("m/s")
237     velocity = float(str(velocity).split(" ")[0])
238     printg(speed, "Current speed: " + format(velocity, ".2f") +
"m/s") # Print to gui
239     root.after(1, send)
241
242 # Constantly get the measured distance, calculate the data, and
update the GUI
243 root.after(0, send)
244 root.mainloop() # Start event loop

```

---

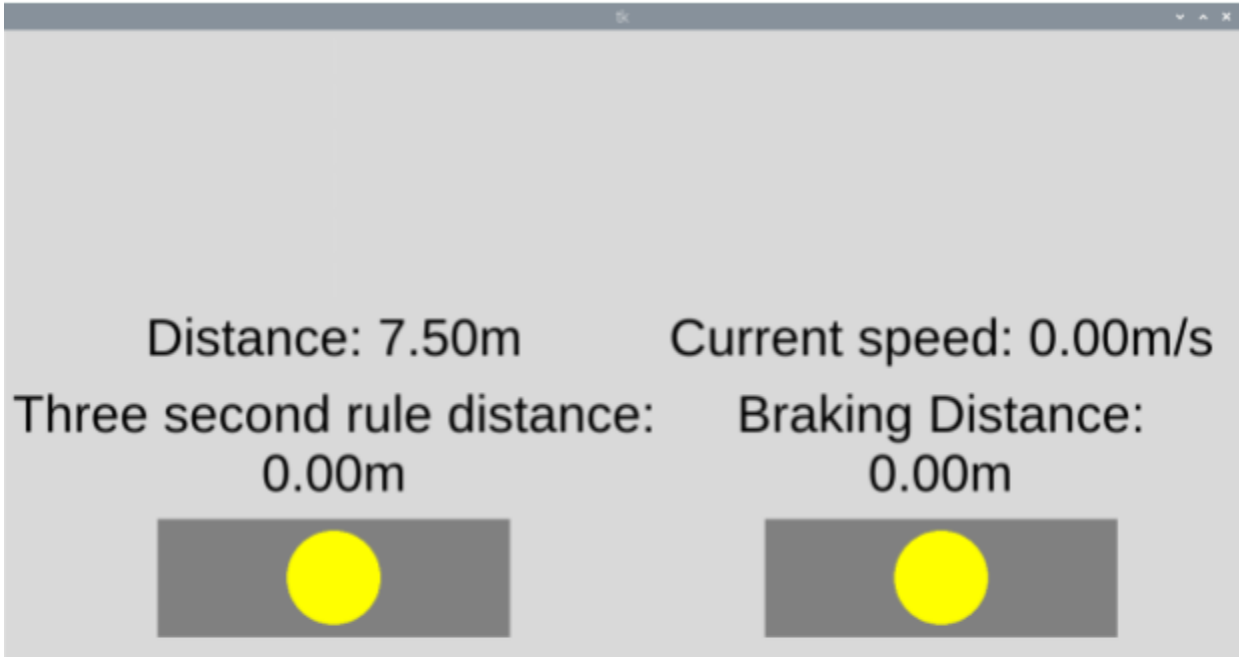


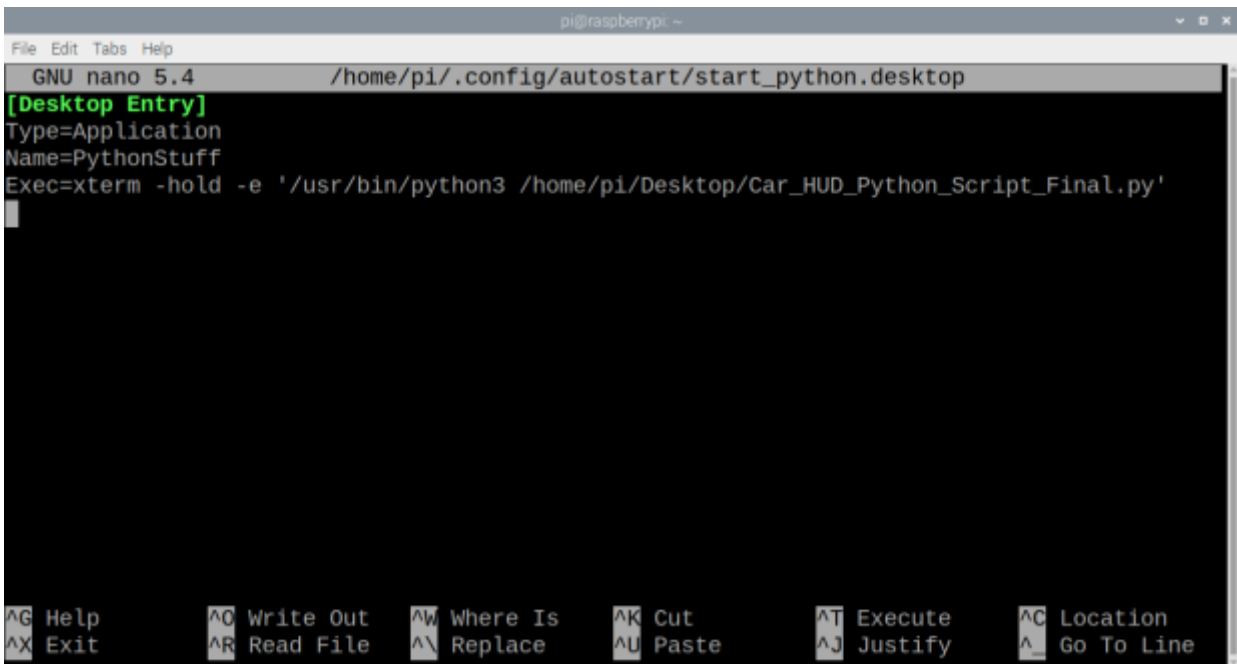
Figure 15: Final iteration of the GUI. Notice how all of the Tkinter widgets are at the bottom.

The last but arguably the most important is ensuring that the screen flips and the Python program executes when the Raspberry Pi boots. This also turned out to be the most difficult part of the design. To give some context, on a computer, many different files are run when the computer boots. These files start critical processes that allow the computer to function. On a Windows PC, the process of making software run on start-up is simple. The user must navigate to the start-up page in the settings, find the required software, and click the enable button. The computer automatically sets the software to run on start-up behind the scenes. On Raspberry Pi, which runs a distribution of Linux, this has to be done manually. Finding the right file to place the run command turned out to be very difficult. This is because many online resources are outdated and only work on older Raspberry Pis. The solution is to use the xterm package and the autostart directory (Hymel, n.d.). First, run `ls /home/pi/.config/autostart` to check if the directory exists (note the absence of `sudo`). If it does not, run `mkdir`

/home/pi/.config/autostart to make the directory (note the absence of **sudo**). Next, run **sudo apt-get xterm -y** to install the xterm package, type **y**, and press enter when prompted. Then run **nano /home/pi/.config/autostart/start\_python.desktop** to create a desktop file (note the absence of **sudo**). This is the file that will be run when the Raspberry Pi boots. In the desktop file, write:

```
[Desktop Entry]
Type=Application
Name=Clock
Exec=xterm -hold -e '/usr/bin/pythonN path/name.py'
```

In this case, pythonN is python3, path is /home/pi/Desktop/, and name.py is Car\_HUD\_Python\_Script\_Final.py (Fig. 16).



```
pi@raspberrypi: ~
File Edit Tabs Help
GNU nano 5.4 /home/pi/.config/autostart/start_python.desktop
[Desktop Entry]
Type=Application
Name=PythonStuff
Exec=xterm -hold -e '/usr/bin/python3 /home/pi/Desktop/Car_HUD_Python_Script_Final.py'
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

Figure 16: Inside of the start\_python.desktop file. This executes the Python file when the Raspberry Pi boots up.

## Discussion

The completed assembly is placed on the dashboard with the translucent pane directly in front of the driver's field of view. It is then secured with duct tape (Fig. 17), while The LiDAR sensor is placed on the hood of the car and is also secured with duct tape (Fig. 19). The cable leading to the LiDAR sensor is routed through the driver-side window and into one of the 4 USB ports. The USB-C power cable is then plugged into a 12v power inverter, which is connected to one of the vehicle's 12v sockets. Finally, the OBD-II to USB adapter is plugged into one of the 4 USB ports on the Raspberry Pi, and its switch is flipped to the HS position (Fig. 20).



Figure 17: Photo of the completed assembly in a car.



Figure 18: Photo of the HUD at night. The Assembly is in the passenger side to enable safe filming.

The field test showed that the device is viable as a product. But as it is still in the prototype phase, several issues need to be fixed by future research and development before it can be put into the market. First, the reflection of the display on the transparent pane is barely visible in the daytime (Fig. 17 vs Fig. 18). Second, the Raspberry Pi eventually sleeps, and the software freezes and needs to be restarted. Third, the assembly is not very stable, so it does not

stay in place, and parts are liable to come out. Fourth, there is a 3 to 5-second delay between when the LiDAR reads the distance and when it is displayed. This delay increases over time due to unoptimized code. Fifth, LiDAR placement is very unforgiving to error. Not only does a cable hang out the driver-side window, but there is little room for error when finding the right angle for the sensor. There needs to be a place where the LiDAR is guaranteed not to miss the vehicle in front of it and where it is level. Lastly, on some models of cars, the Raspberry Pi still receives power even when the car is off, causing the car to run out of battery.

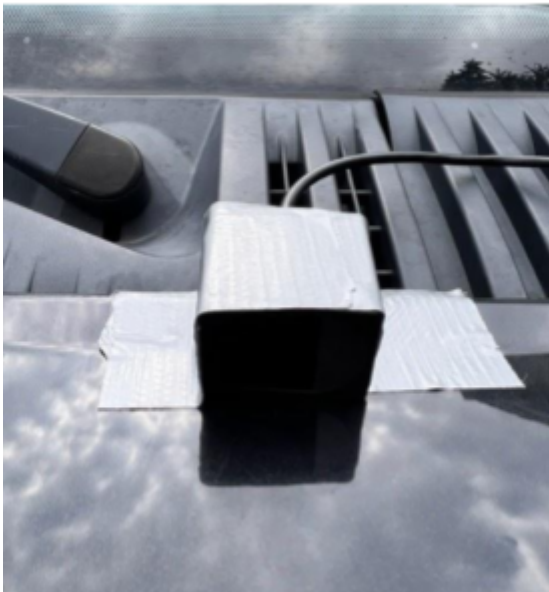


Figure 19: Photo of the LiDAR duct taped to the front of the car just in front of the windshield.



Figure 20: Photo of the OBD-II adapter with the switch flipped to the HS position.

## Conclusion

As a whole, the device works as intended. All of the Peripherals connect, and the software starts up automatically when the Raspberry Pi turns on, in addition to the GUI updating in response to the LiDAR and the OBD-II. Therefore, if the device could be made more compact, more efficient, and less expensive, it would be a viable product to increase driver safety, including for young drivers who need extra safety features when first learning. Finally, the device meets the set requirements. It gets the vehicle's speed and distance to the next vehicle, calculates the braking and three-second rule distance, and safely displays the information to the driver.

## Bibliography

Acuity Laser. (2021, August 31). *Laser triangulation sensors - non-contact measurement*.

Retrieved June 13, 2022, from

<https://www.acuitylaser.com/sensor-resources/laser-triangulation-sensors/>

Acuity Laser. (2021, February 11). *Principles of measurement used by laser sensors and*

*scanners*. Retrieved June 13, 2022, from

<https://www.acuitylaser.com/sensor-resources/measurement-principles/>

California Air Resources Board. (2019, September 19). *On-Board Diagnostic II (OBD II)*

*Systems Fact Sheet* | California Air Resources Board. Retrieved June 9, 2022, from

<https://ww2.arb.ca.gov/resources/fact-sheets/board-diagnostic-ii-obd-ii-systems-fact-sheet>

Cesar-S. (2019, February 5). Cesar-S/DFRobot\_TFmini: Master library arduino for XOD

Library. GitHub. Retrieved July 31, 2022, from

[https://github.com/Cesar-S/DFRobot\\_TFmini](https://github.com/Cesar-S/DFRobot_TFmini)

Charissis, V., Falah, J., Lagoo, R., Alfalah, S. F., Khan, S., Wang, S., Altarteer, S., Larbi, K. B.,

& Drikakis, D. (2021). Employing emerging technologies to develop and evaluate

in-vehicle intelligent systems for Driver Support: Infotainment Ar Hud Case Study.

*Applied Sciences*, 11(Artificial Intelligence and Emerging Technologies), 1397.

<https://doi.org/10.3390/app11041397>

Hymel, S. (n.d.). *Method 2: autostart*. How to run a Raspberry Pi program on Startup.

Retrieved August 10, 2022, from

<https://learn.sparkfun.com/tutorials/how-to-run-a-raspberry-pi-program-on-startup/method-2-autostart>

Illinois Secretary of State. (2022). *Illinois rules of the road 2022*. Jesse White, Secretary of

State. Retrieved June 9, 2022, from

[https://www.ilsos.gov/publications/pdf\\_publications/dsd\\_a112.pdf](https://www.ilsos.gov/publications/pdf_publications/dsd_a112.pdf)

Jain, G. (2019, April 15). *Holographic car heads-up display (HUD)*. Instructables. Retrieved

July 12, 2022, from

<https://www.instructables.com/DIY-Holographic-Car-Heads-Up-Display-HUD/>

Mehendale, N., & Neoge, S. (2020). Review on lidar technology. *SSRN Electronic Journal*.

<https://doi.org/10.2139/ssrn.3604309>

National Association of City Transportation Officials (2013, September 17) Vehicle Stopping

Distance and Time | *University of Pennsylvania School of Engineering*. Retrieved June

9, 2022, from

[https://nacto.org/references/a-hrefdocsusdgvehicle\\_stopping\\_distance\\_and\\_time\\_upenn](https://nacto.org/references/a-hrefdocsusdgvehicle_stopping_distance_and_time_upenn)

National Oceanic and Atmospheric Administration (NOAA), US Department of Commerce,

(2012, October 1). *What is Lidar?* NOAA's National Ocean Service. Retrieved June 14,

2022, from <https://oceanservice.noaa.gov/facts/lidar.html>



Suchocki, C. (2020). Comparison of time-of-flight and phase-shift TLS intensity data for the diagnostics measurements of buildings. *Materials*, 13(2), 353.

<https://doi.org/10.3390/ma13020353>

Terabee. (2022, April 26). *Time-of-flight principle (TOF): Brief overview, technologies, and advantages*. Time-of-Flight principle. Retrieved June 13, 2022, from

<https://www.terabee.com/time-of-flight-principle/>

Whitfield, B. (2019, May 14). Brendan-W/Python-OBD: OBD-II serial module for reading Engine Data. GitHub. Retrieved July 31, 2022, from

<https://github.com/brendan-w/python-OBD/>

Youngworth, R. N., Tsao, C.-hsi, & Chan, C. (2021, April 21). Designing and analyzing automotive head-up displays. Laser Focus World. Retrieved June 9, 2022, from <https://www.laserfocusworld.com/optics/article/14199007/designing-and-analyzing-automotive-headup-display>

YouTube. (2019). Diy: Making a Car Hud display. YouTube. Retrieved July 11, 2022, from <https://www.youtube.com/watch?v=1XJNcdbZsGo>.