Solution

For the Hackbooth presentation on the 14th of June, a working prototype called DARE, standing for Driver Alertness Recognition Equipment, was presented. This document details how it works and how it was built.

1. Algorithm

First of all, the NOOBS Operating System was downloaded on a 32GB SD card, following the Pi Foundation recommendations.Then, we created our own working environment in Python so that we could add all necessary libraries for the computer vision algorithm, as instructed by Adrian Rosebrock. [2] The system was programmed to boot up and run the main program immediately, using instructions found on the internet.[3]

Drowsiness Detection

The method used to detect drowsiness can be broken down in several steps.

1. OpenCV's Haar cascades is a pretrained face detector. [4] It works on black and white images (which is why we convert every colored picture to black and white). On a new picture, it will test different lighting features that it knows are recurrent in a human face. For each face detected, it will give an x and y-coordinate, as well as a height h and width w value. This is to locate a rectangle around the visage detected. The "cascade" comes from the fact that the algorithm tests groups of features one after another to save computing power. For example, a first group of 5 lighting features that are the most likely to identify a human face are tested on a new input. If this new image doesn't contain any of these features, then there's no need to test more less accurate features. The algorithm will decide there's no face and therefore stop the image processing. If it does contain these lighting patterns, then it tests a new block of, for example, 20 features, then 50 new ones, until all have been tested (a bit more than 6000 features). This detector is extremely fast, but pretty inaccurate: It will often detect faces where there are not. However, it will almost always correctly identify a visage, which is what matters to us since normally the device will always be facing the user.
2. The (x,y,h,w) coordinates are then given to dlib's facial landmark predictor (see Figure 1). This predictor returns 68 key points in an image to identify the important parts of a visage, such as the eyes, nose, mouth and ears.

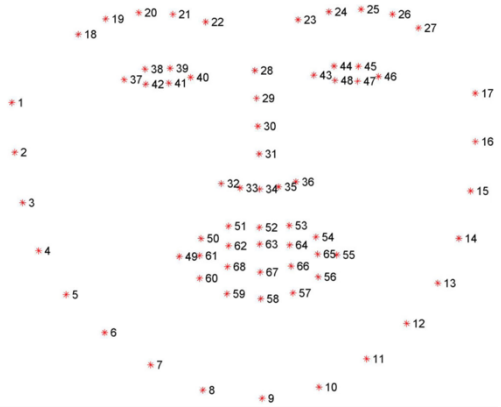


Figure 1: 68 facial landmarks

From there, we extract the points defining the eyes and mouth. We then use the Eye Aspect Ratio as defined below to determine if the distance between the upper eyelid and lower eyelid has fallen below a certain threshold value (see Figure 2). This method is used because it is more robust to the fact that users might stand closer or further away from the device from one car to another, as well as the fact that eyes will change from one customer to another. Taking the ratio of the vertical distance to the horizontal distance reduces the impact of these potential problems. Since we want to detect if the driver has fallen asleep but not if he is blinking, we look at the number of consecutive frames where the EAR is under the "eye closed" threshold. If this number is above a certain predefined value, then we declare the user is sleeping: we ring the alarm for a brief moment in order to wake him up.

$$EAR = 0.5 * (\frac{|p38 - p42| + |p39 - p41|}{2|p37 - p40|} + \frac{|p44 - p48| + |p45 - p47|}{2|p43 - p46|})$$

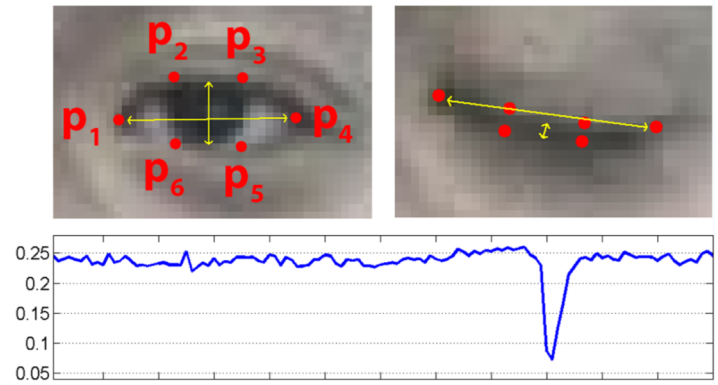


Figure 2: Top-left: A visualization of eye landmarks when then the eye is open. Top-right: Eye landmarks when the eye is closed. Bottom: Plotting the eye aspect ratio over time. The dip in the eye aspect ratio indicates a blink. [5]

For yawning, we look if the Mouth Aspect Ratio (MAR) as defined below goes above another threshold. We use again a ratio as mouth size can vary from one user to another and with different perspectives. If it does go above the threshold for a certain number of consecutive frames, we create a new thread (to enable the rest of the code to continue) that will turn on the LEDs depending on the drowsiness state. If it is the first time the driver has yawned, then the yellow LED will blink. If it is more than once, then the red LED will blink, indicating a dangerous sleepy state.

$$MAR = \frac{|p51 - p59| + |p53 - p57|}{2|p49 - p55|}$$

With sleeping and yawning detection, we are therefore able to detect and warn the user of his alertness deterioration.

2. Mobile Application and Bluetooth connection

The App created continuously receives data sent from the device which represents the driver's current status. Once drowsiness is detected, a different message is sent (see Bluetooth explanation below), leading to the App playing a music chosen initially by the user to keep him awake. If he wants to, he can press a "stop" button that will stop the music. To download the application, the "BLE App" folder should be dragged to XCode on a Mac, and then compile it and download it to the iPhone.

Defining the roles

Before the team actually designed the BLE app, it was important to define the role involved in the BLE protocol stack clearly. According to the specification of Bluetooth 4.0, different layers of the stack are in charge of different tasks. The **GAP** layer (Generic Access Profile) is responsible for finding and creating connections between different devices. There are four roles that needed to be addressed [6].

1. **Peripheral**: Sends out messages to inform central devices that it is ready for connection.
2. **Central:** Scans for messages from peripheral devices and initiates connections.
3. **Broadcaster**: Sends messages like a peripheral but we can't connect to it.
4. **Observer**: Scans for messages but it can't initiate connections.

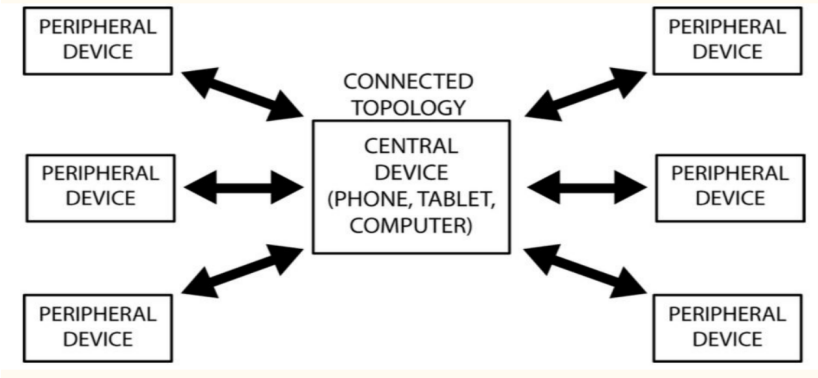


Figure 3: GAP layer

Following this concept, the Raspberry Pi was defined as a peripheral since it needs to send data to the phone at regular time intervals and it is generally resource constrained in term of power consumption, computing power and memory size. The phone would play the role of central.

Once the devices are connected together, the **GATT** (Generic Attribute Profile) layer of BLE protocol defines the level of application.

1. **Server**: the device that has the data to be read or written by the client.
2. **Client**: the device that sends out request to read or write data from the server.

The roles of server and client itself are independent of the role of peripheral and central. In our design, the phone will be the client receiving the data sent from the server, namely the Raspberry Pi. The next step was to program server and client separately in order to successfully transmit and receive data. The team started with the server side first.

Designing the server

In our implementation, the driver's status needs to be represented as one variable that has two states: sleeping and awake. By sending this variable from the server to the client, the phone can detect the user's drowsiness and thus play the music. The first task was to fit this variable into the GATT protocol.

Within the high-level design of GATT, there are nested objects called profile, service and characteristics, which hierarchical relationships are shown in Figure 4.

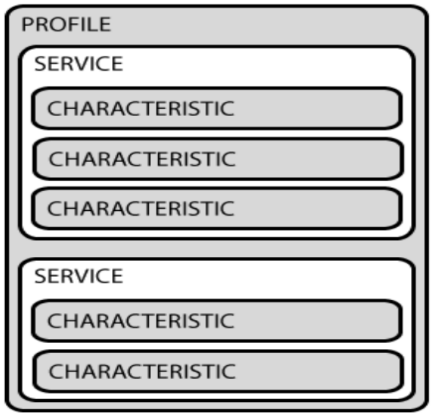


Figure 4: GATT design

According to the definition given on the official website [7], the profile is a pre-defined collection of services, and the service are collections of characteristics and relationships to other services that encapsulate the behaviour of part of a device. It indicates the possibility to wrap this variable into a characteristic and transmit it through BLE.

Instead of building it from scratch, several standard examples of servers in the official Linux Bluetooth protocol stack *bluez* were studied. The *bluez* library was therefore installed on the Raspberry Pi to enable it to communicate with other BLE devices. There are four ways to transfer data between server and client: **read**, **write**, **notify**, and **indicate**. For this product, the client does not send any data to server so only the **read** and **notify** operations are needed. The **notify** operation enables to send data from server to client as long as the data becomes available, so the client (iPhone), does not need to request it in advance. So, whenever a new driver's status is detected, the server will send the corresponding variable to the client. The next step was to configure a server to use the notify procedure.

An example found online was used for that purpose. Originally designed to report a simulated battery level to the client every two seconds, we modified it so that it would instead send the drowsiness status.

This variable can then be successfully received using a testing app called **Bluetooth Smart Scanner** once the program is run on the Raspberry Pi. Assuming there are two threads in the main code, thread A is responsible for drowsiness detection and thread B for BLE connections. A global variable "sig" is defined and shared between each thread to represent the driver's status.

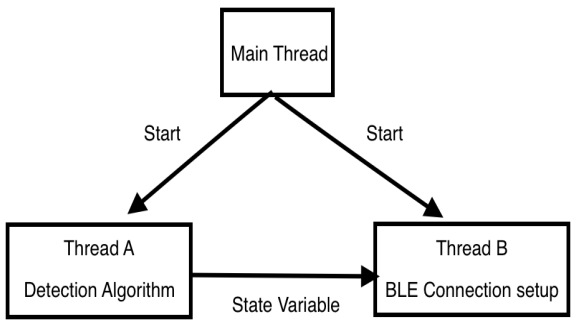


Figure 5: Algorithm threads

There is no possibility in Python to have two threads simultaneously reading or writing a global variable, so no specific action was taken to prevent such error. The "sig" variable is initially set to 80. If the driver has fallen asleep, the variable will be updated to

82 in thread A and then sent in thread B to the phone.

Designing the Client

Since the iPhone is defined as client in our design discussed above, the core Bluetooth framework was added to the Application code using the Swift environment. The GATT structure was then used to implement the Bluetooth connection. The whole was split into 4 "view controller" parts, a user-friendly way from the Swift environment to connect different code parts together. Here we used the following high-level organisation.

1. **FirstViewController**: in charge of the page layout and basic functions of the home page.
2. **SecondViewController**: in charge of the page layout and basic functions of the music playlist.
3. **BLECentralViewController**: in charge of the BLE connection, listing all connectable peripheral devices in a table.
4. **Uart Module ViewController**: checks the received data from the server and executes the appropriate action.

After the basic construction of the view controllers listed above, along with some interface design, the App was able to scan all the peripheral devices around the phone and put them into a table in **BLECentralViewController.** However, it seemed a bit redundant to list all BLE devices around, generating a very long list, as only the server actually matters so the code was changed to only find the server. The following features were then added.

1. A check mark to be placed behind the name of the song after it is chosen to inform the user which song would play if sleepiness was to be detected.
2. The app should keep the user's setting as default, so he doesn't need to re-enter it every time.
3. The song should only play for 1 minute then stop automatically.
4. Once the connection is lost between the phone and the device, it should be reported to the user.

Specifications 1 and 2 were achieved by combining the checkmark function and the UserDefaults.standard function in **Second View Controller**. Specification 3 was achieved by adding a timer to **Uart Module ViewController.** Specification 4 was realized by using a counter whose initial value is 5. The counter is decremented by 1 every second unless the client receives data from the server in which case the value is reset to 5. That is, once the client does not receive any data from the server for more than 5 seconds (once the counter reaches 0), the connection is considered as lost and this condition will be reported to the user.

3. 3D-printed case

The case should include heat sinks and be aesthetically appealing as well as durable, to efficiently protect the internal components. All 3D designs were completed using Autodesk Fusion 360. After the designs were finished, .stl files were generated for 3D printing. The equipment used is a 3D printer named Original Prusa i3 MK2 as shown below. The stl files used can be found in the "Raspberry Case" folder.

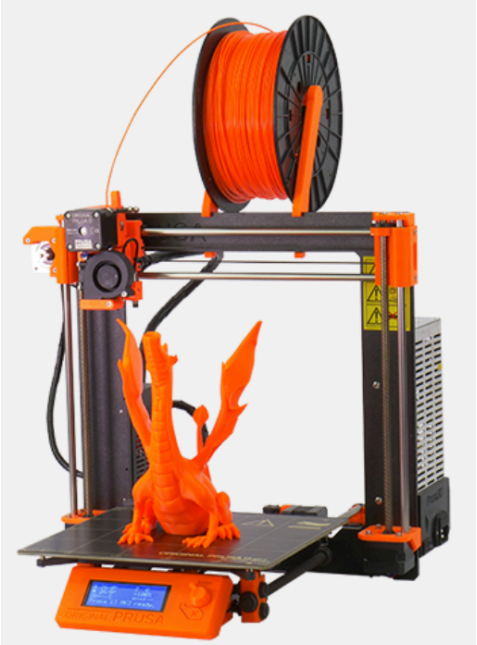


Figure 6: Original Prusa i3 MK2

The material used to construct the case was PLA (polylactic acid) which is a biodegradable and bioactive thermoplastic aliphatic polyester derived from renewable resources. The PLA colour used was grey.

Figure 7: polylactic acid

Main case

The main case was designed using the Raspberry Pi mechanical drawing (see below) provided on the internet.

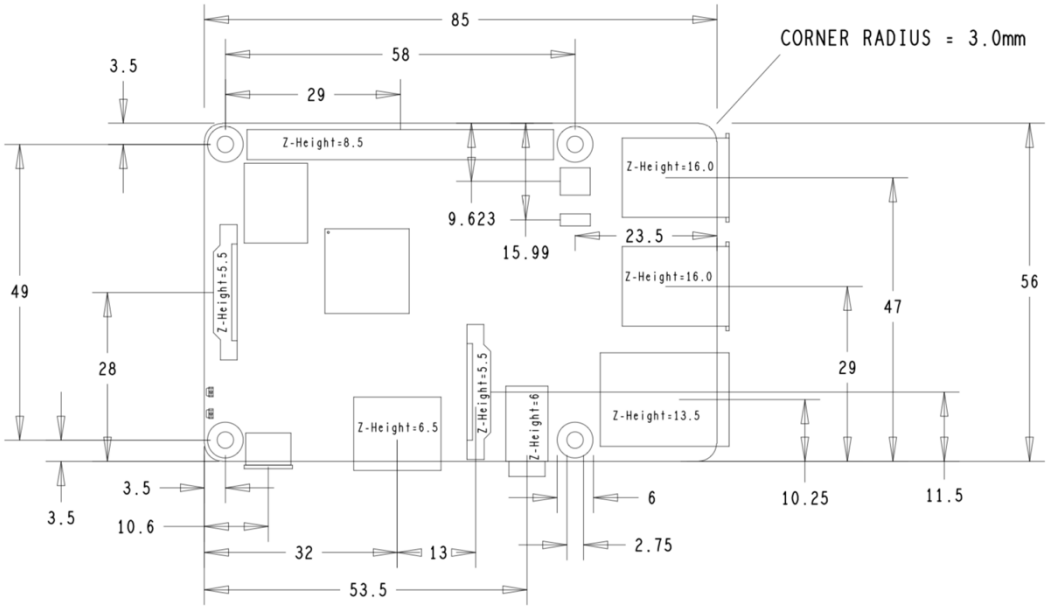
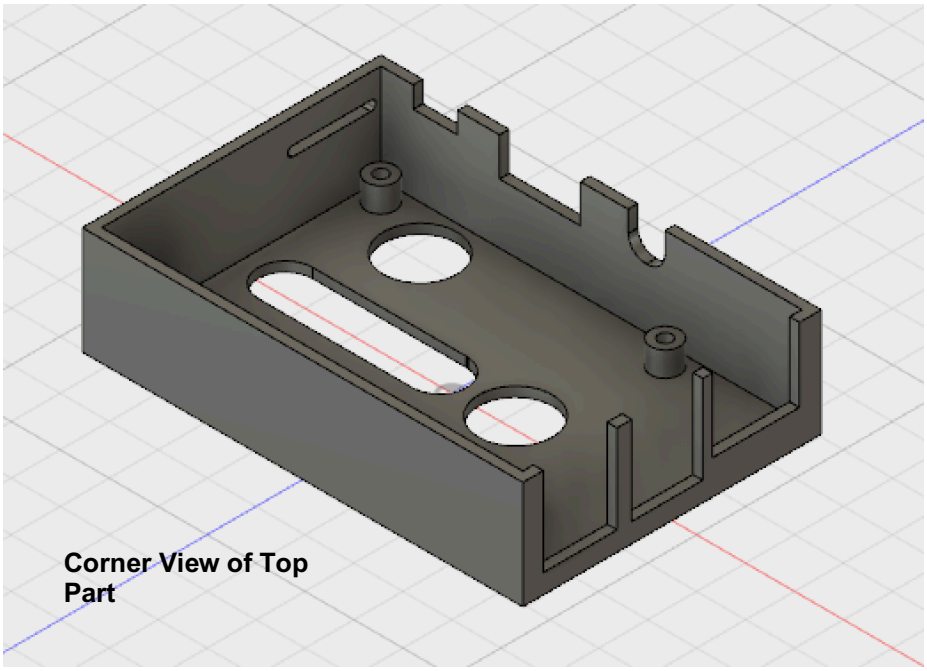


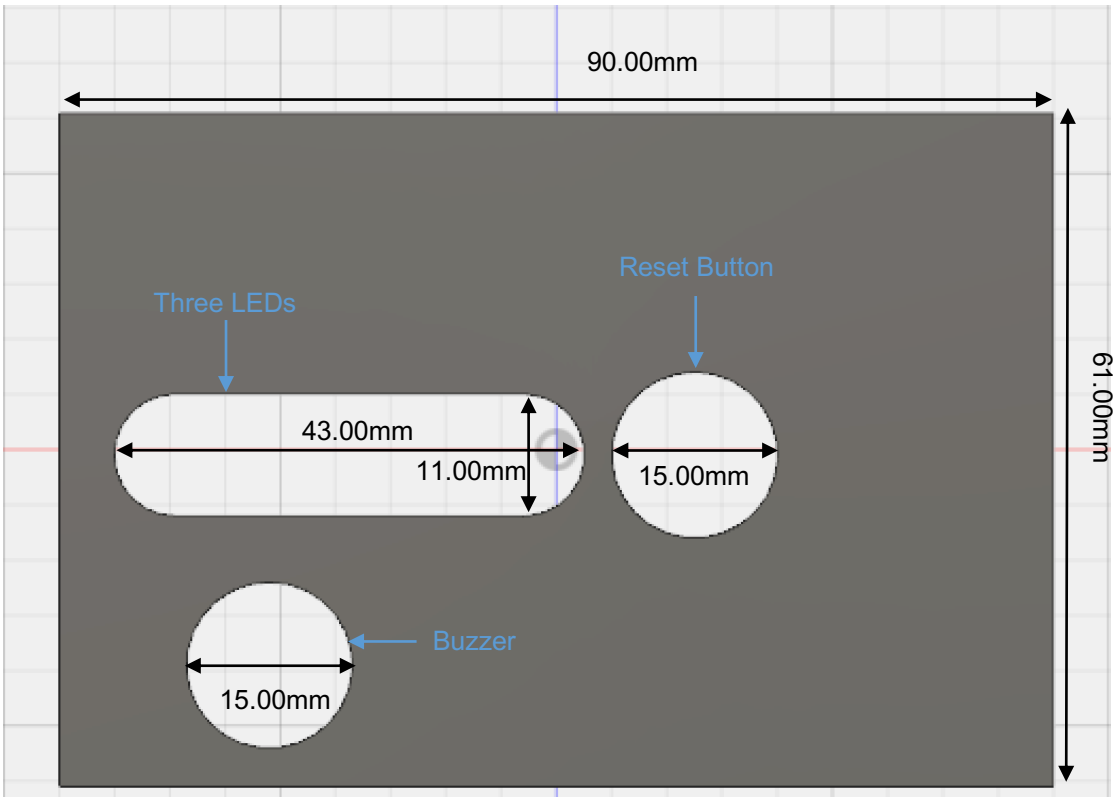
Figure 8: Raspberry Pi mechanical drawing

The case is divided into two parts, one top part and one bottom part. Two parts were joined together by screws passing through four holes on both boards.

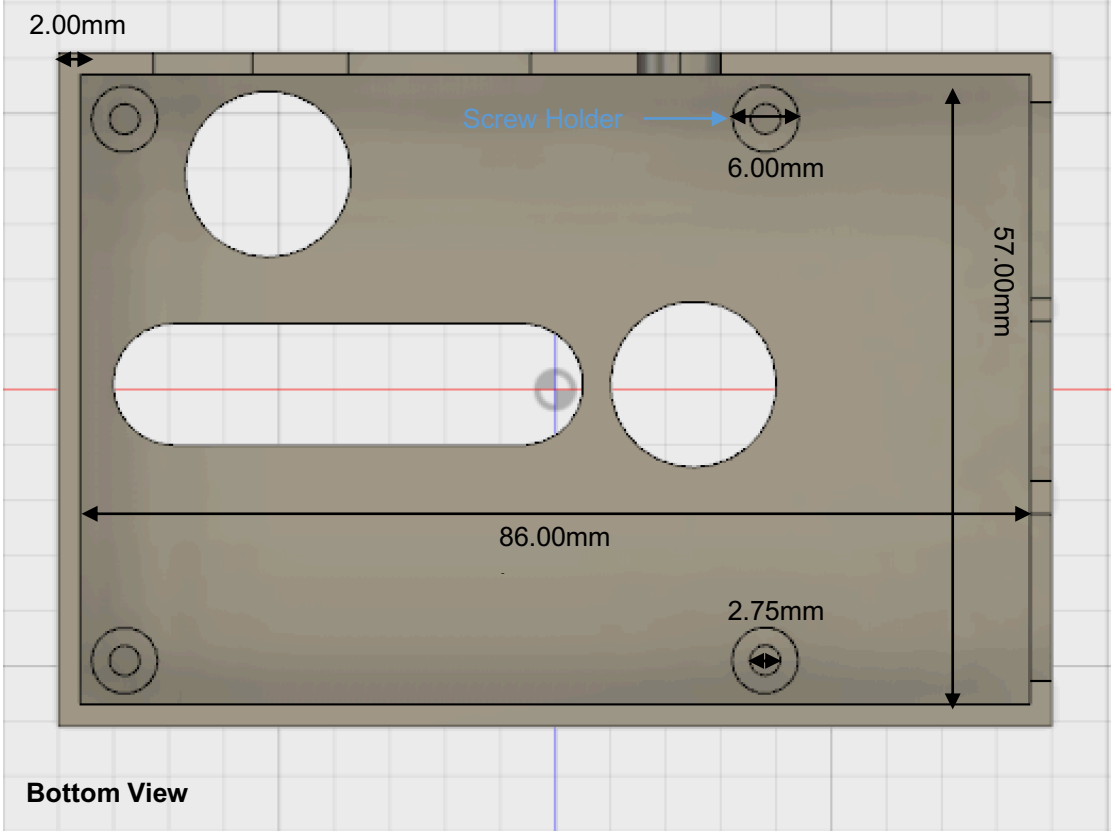
- Top Part

According to the mechanical drawing, the length of the Raspberry Pi is 85mm and the width is 49mm. For the case, both the length and width were increased by 1mm and the thickness by 2mm, to ensure the traffic HAT board could fit in.

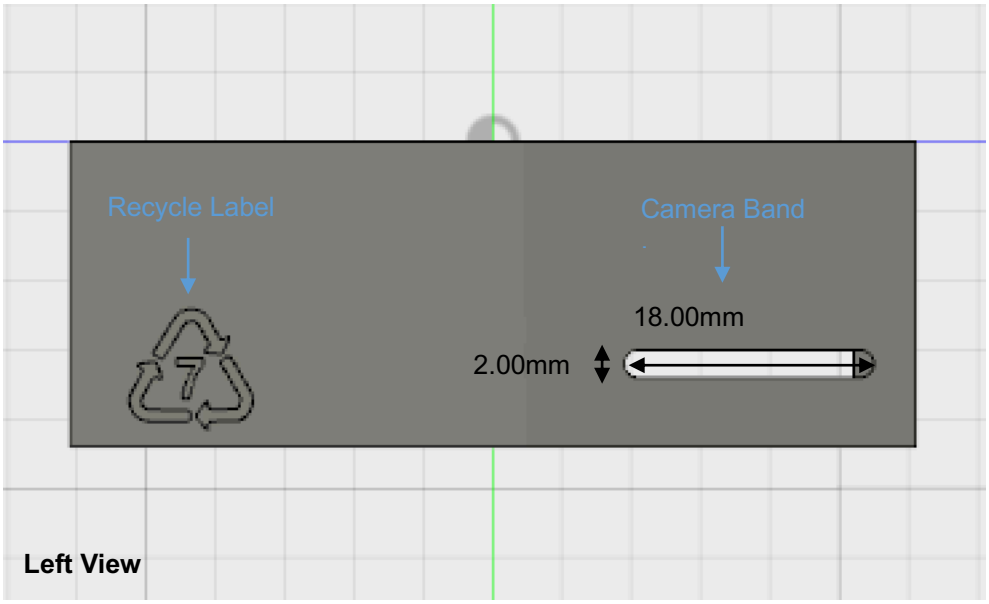




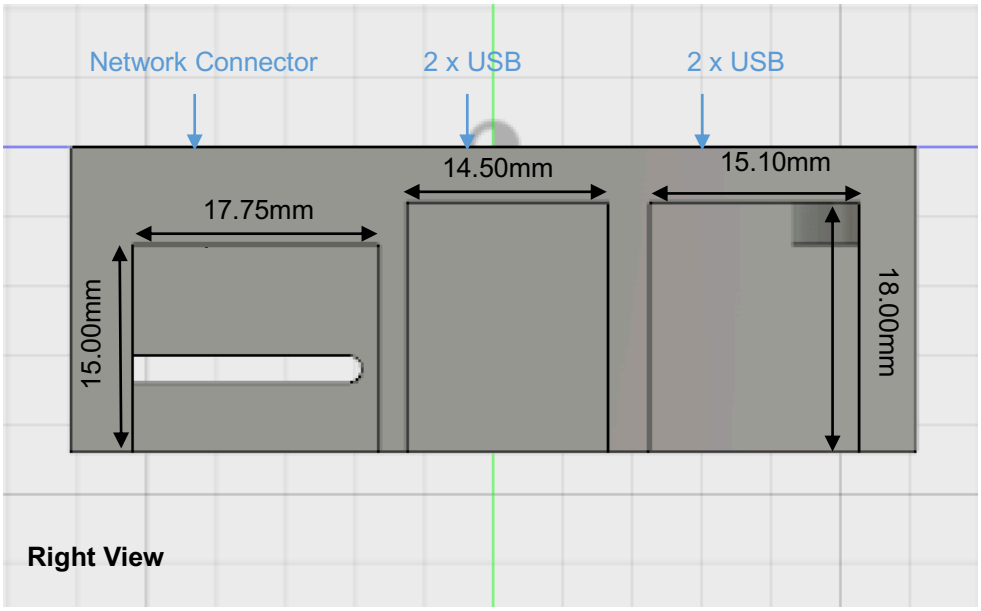
Top View



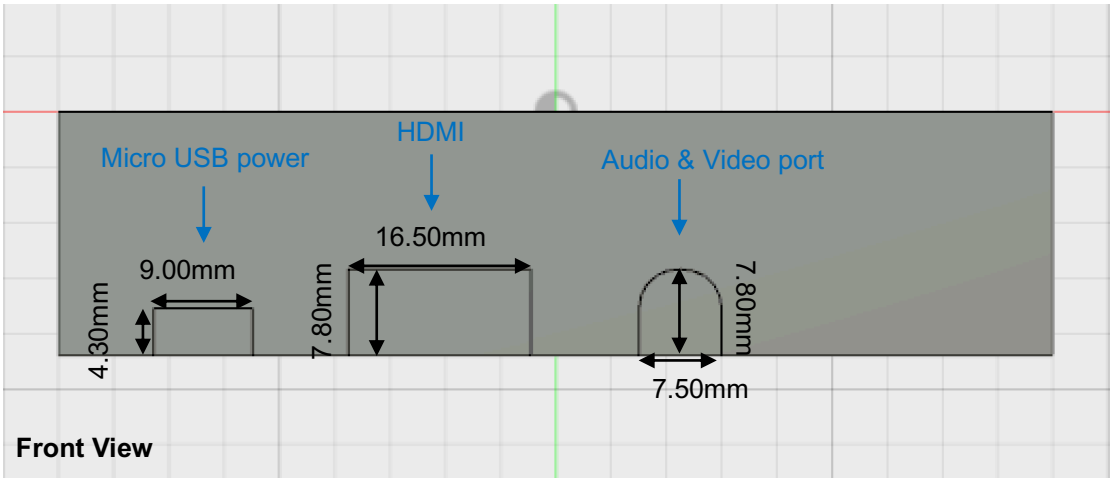
Bottom View

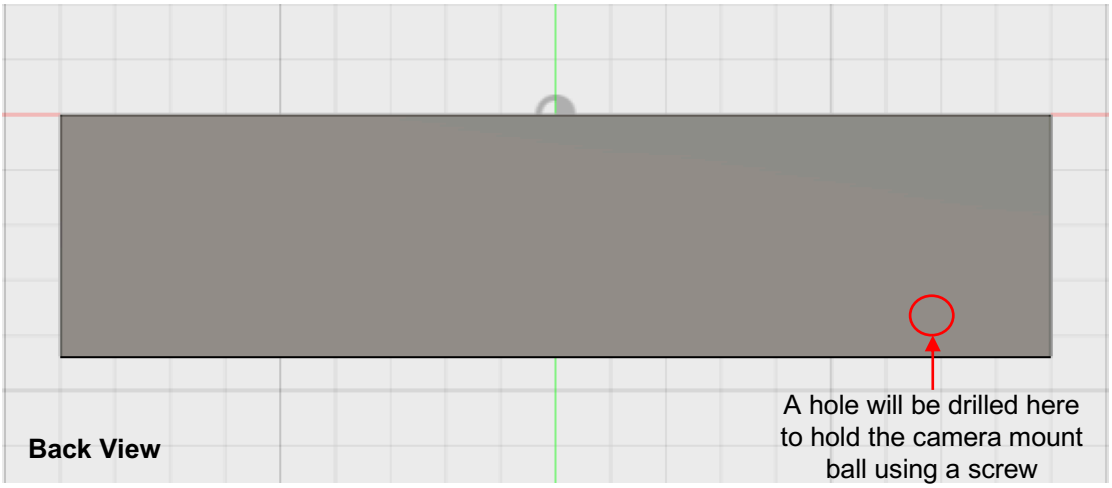


Left View



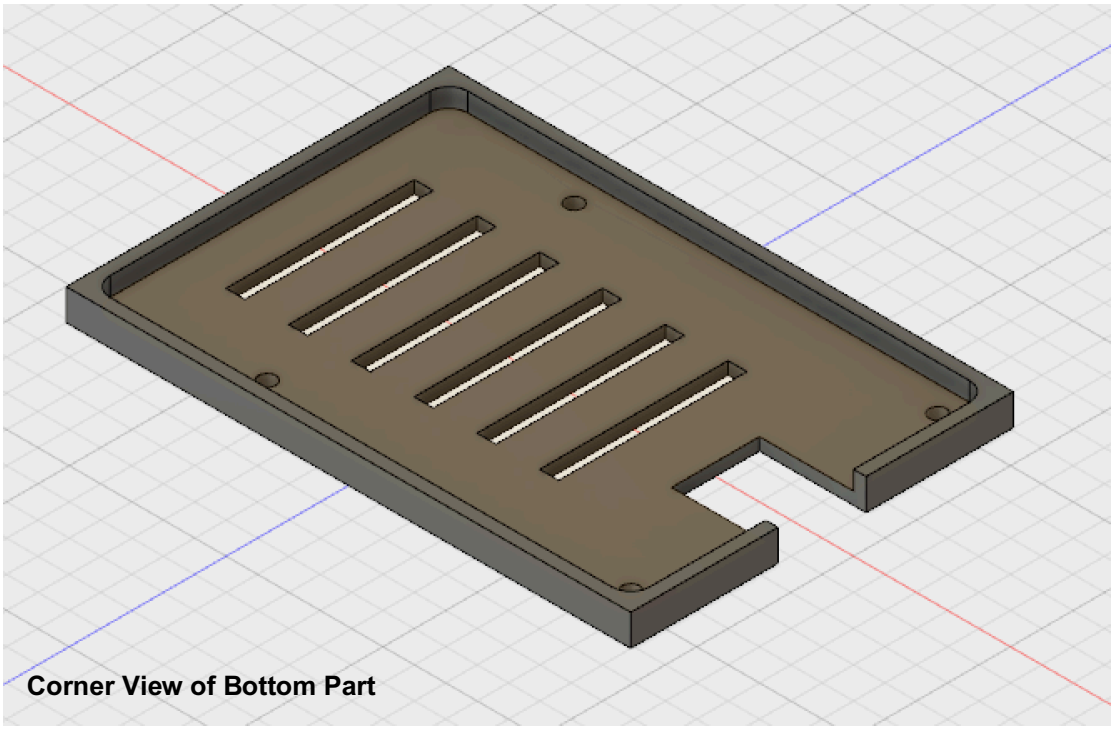
Right View


Micro USB power
HDMI
Audio & Video port
9.00mm
16.50mm
4.30mm
7.80mm
7.80mm
7.50mm
Front View


A hole will be drilled here to hold the camera mount ball using a screw
Back View

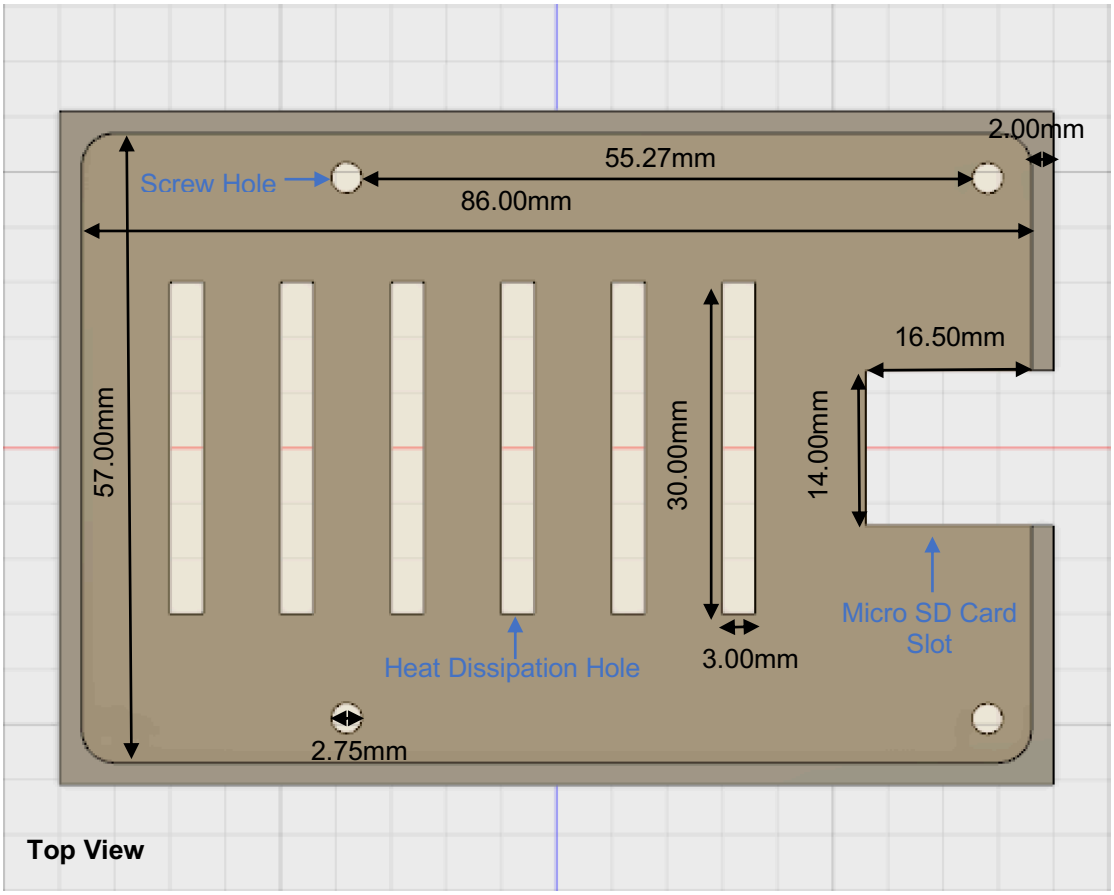
- Bottom Part

The figure below is a corner view of the bottom part. Six heat dissipation holes are carved to ensure the board doesn't get too hot, which could damage the components.

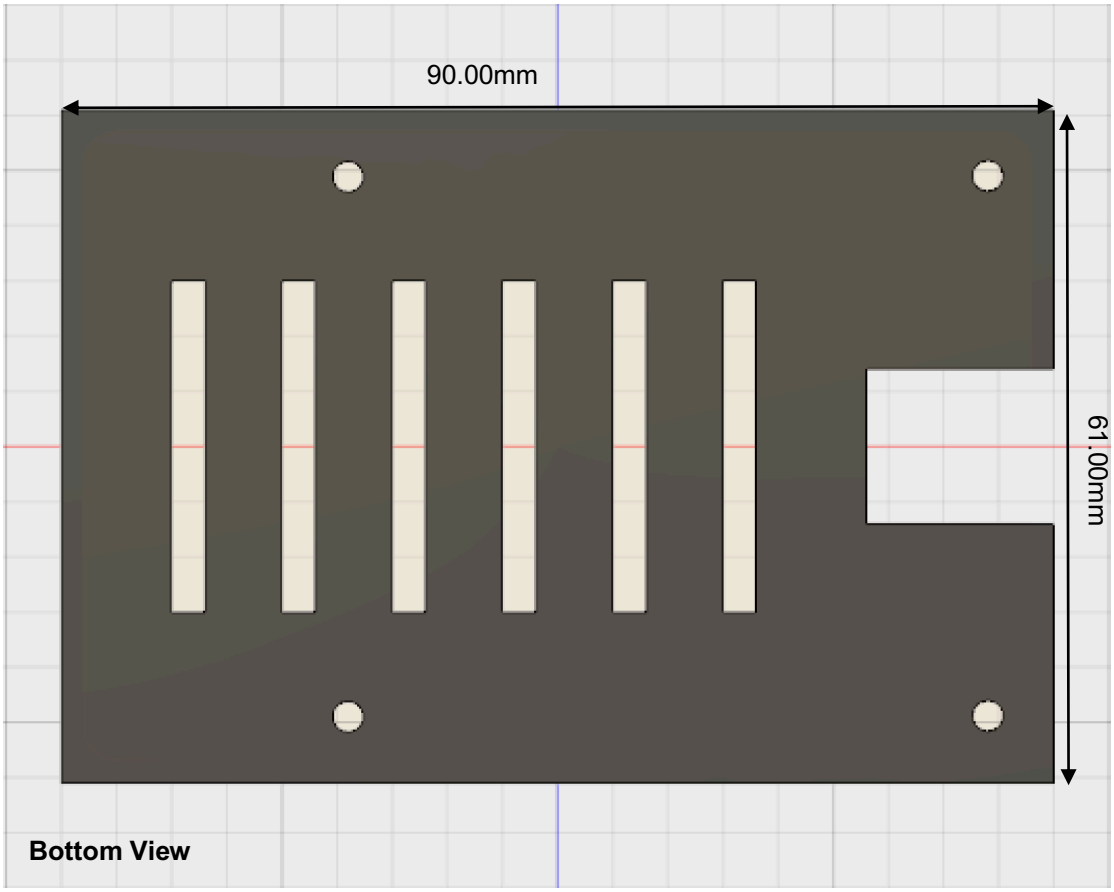


Corner View of Bottom Part

The sketches below are the top and bottom views of the bottom part of the case.



Top View



Camera Case

A camera case was designed to hold the Raspberry Pi camera. There are two parts, a front case and a back plate which will be joined together using screws.

- Front Part

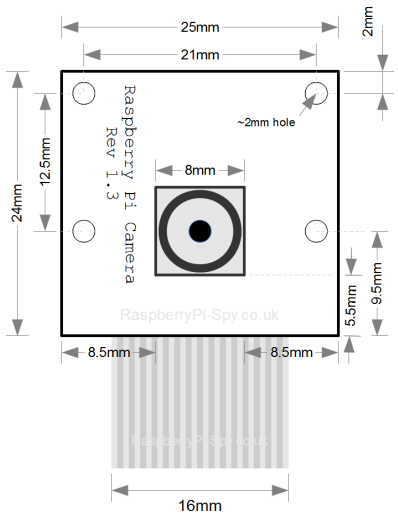
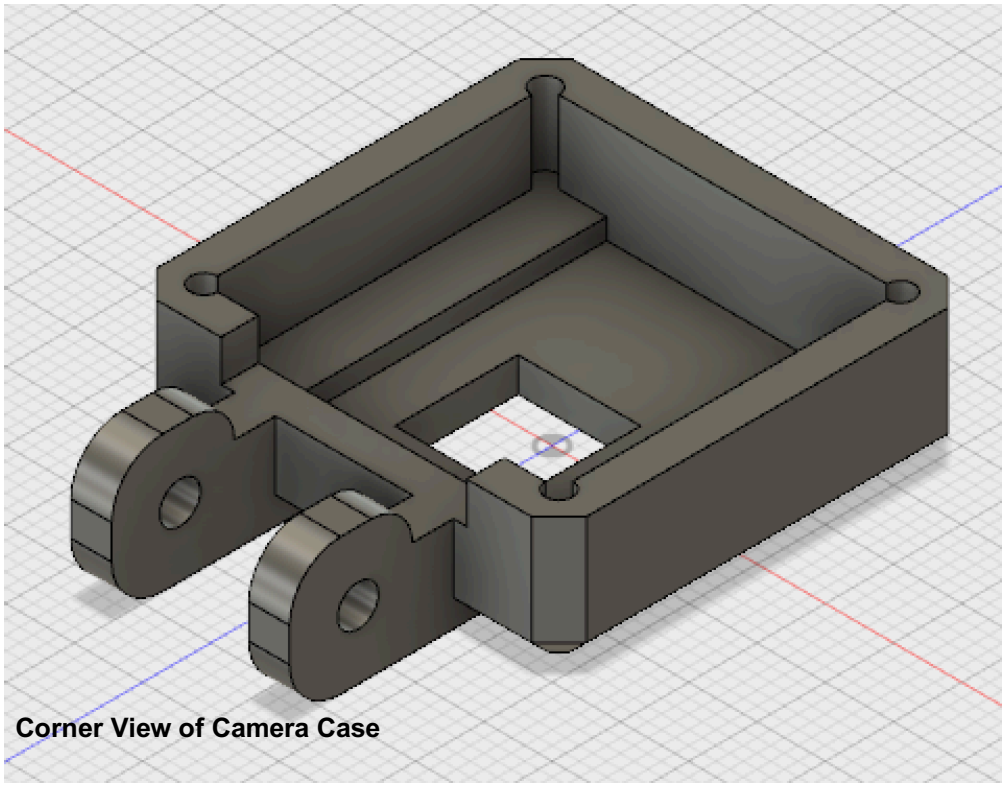
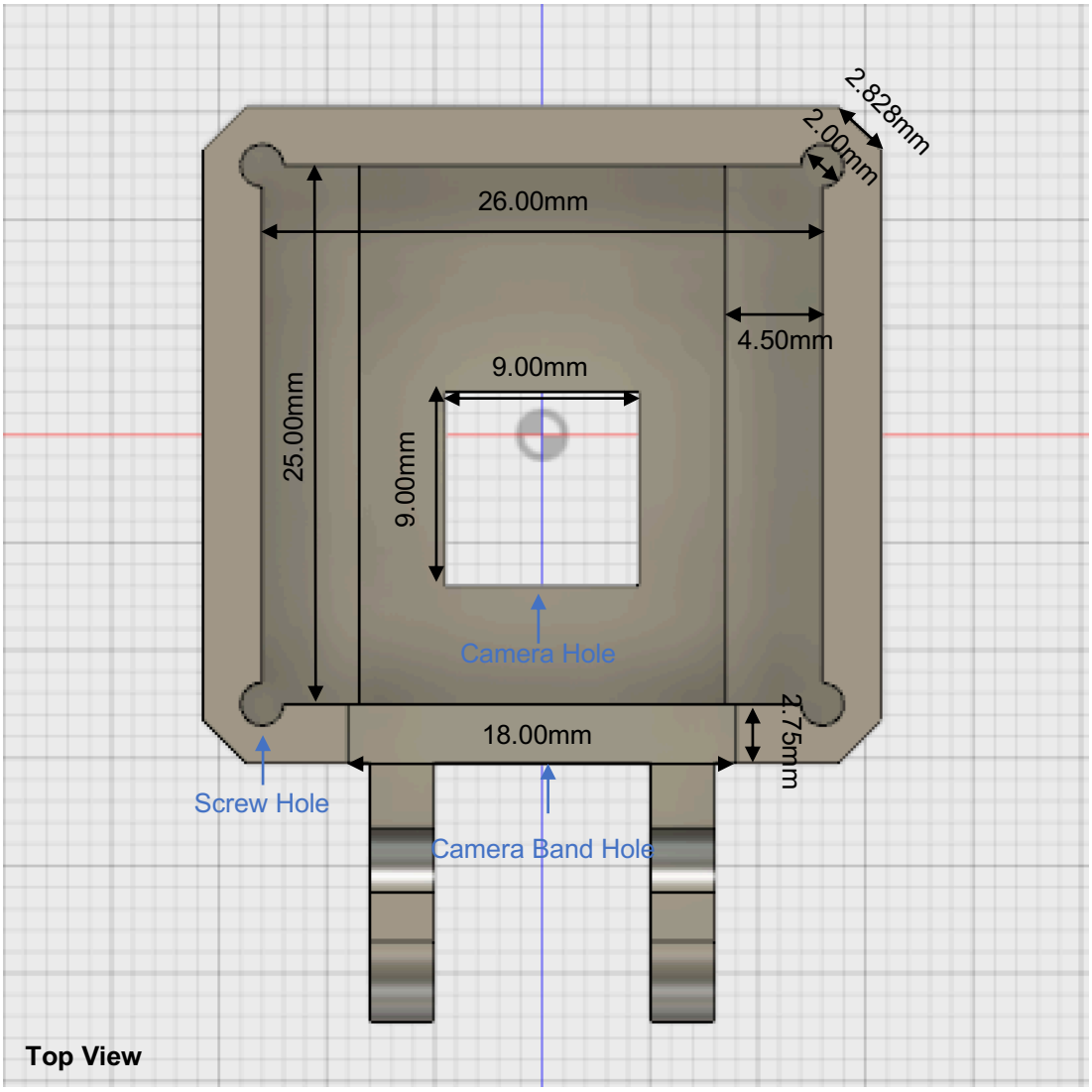


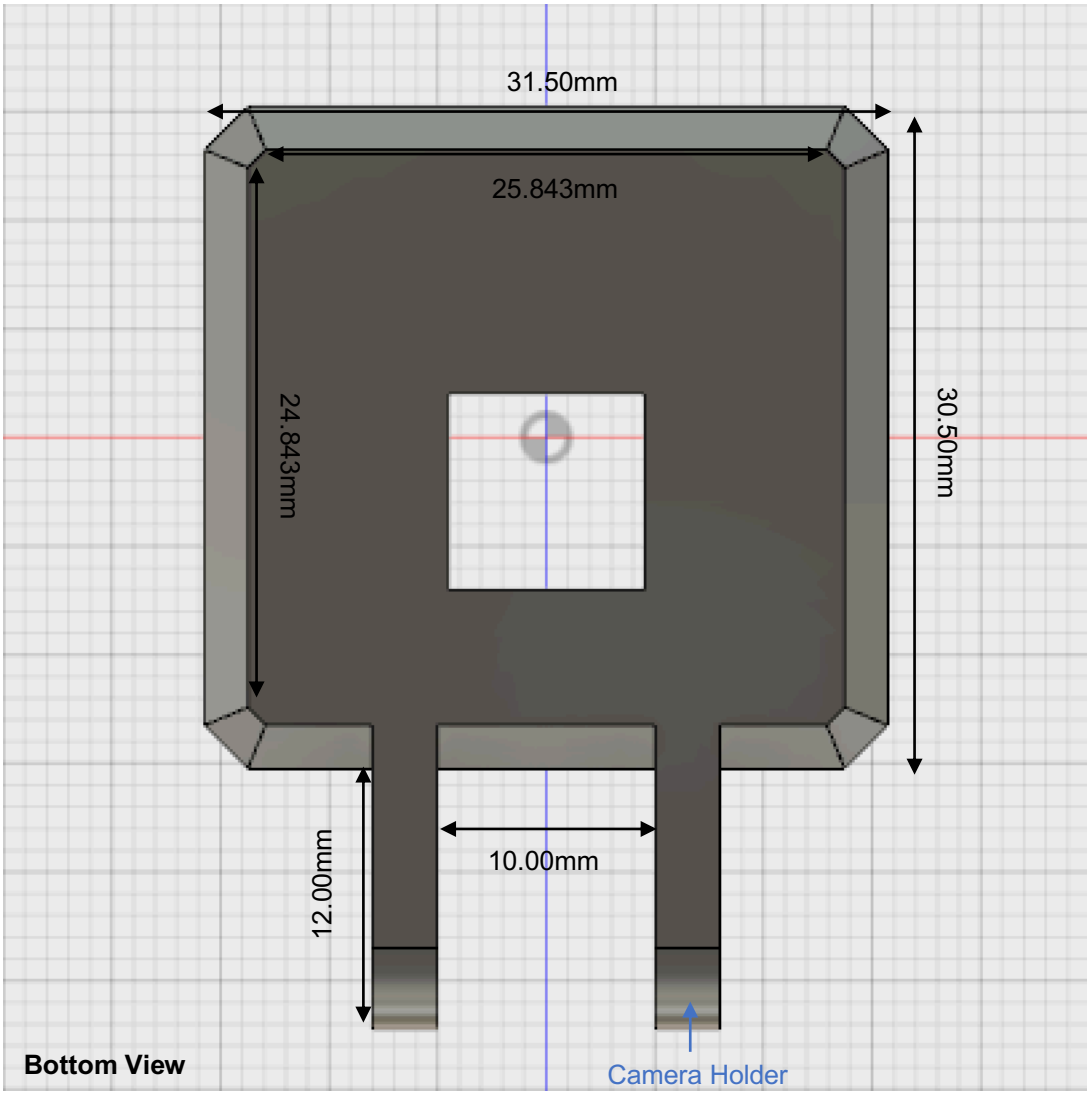
Figure 9: Front part of camera case

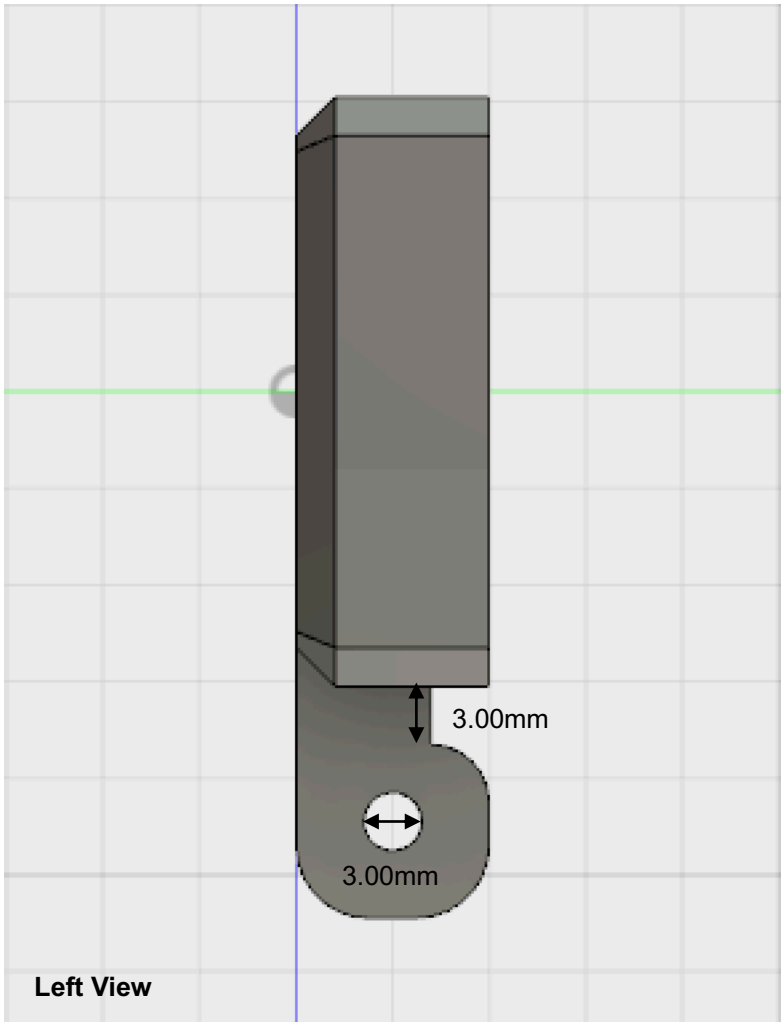


Corner View of Camera Case


2.828mm
2.00mm
26.00mm
4.50mm
9.00mm
25.00mm
9.00mm
Camera Hole
18.00mm
2.75mm
Screw Hole
Camera Band Hole

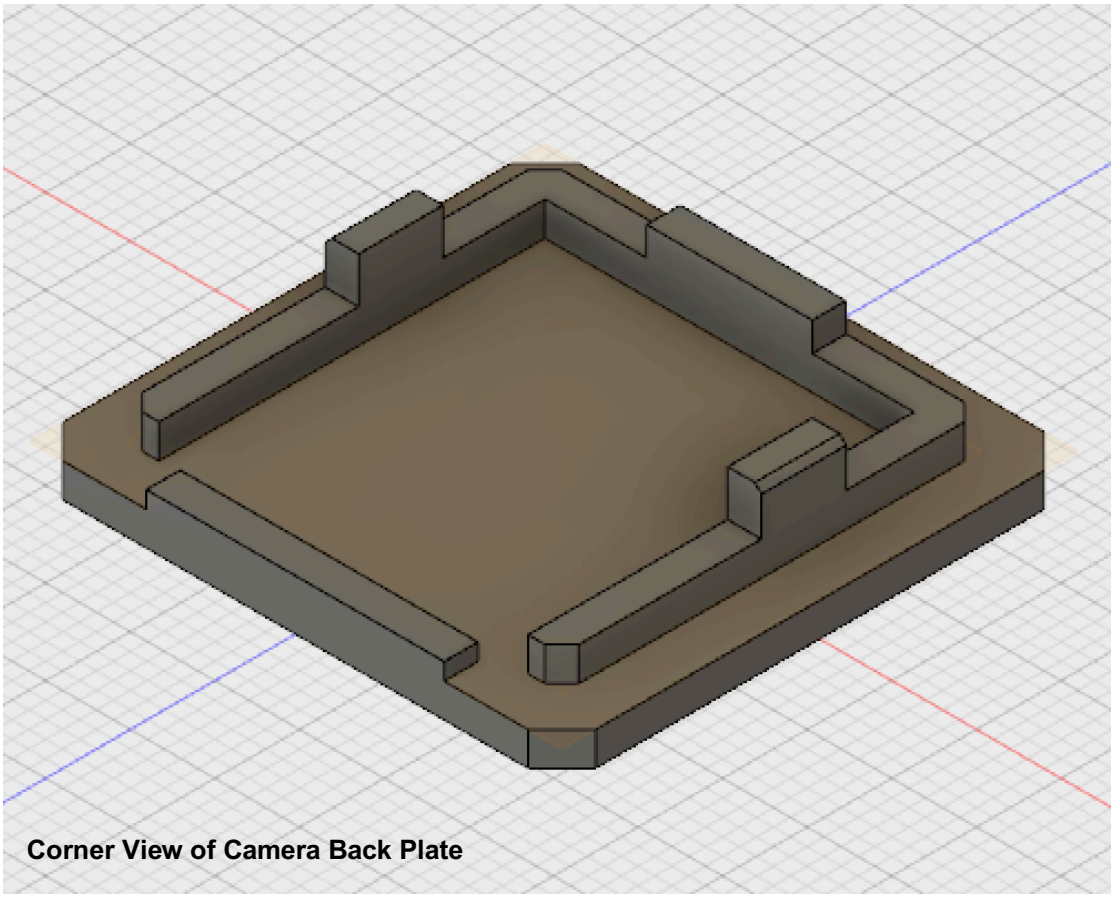
Top View


31.50mm
25.843mm
24.843mm
30.50mm
12.00mm
10.00mm
Bottom View
Camera Holder

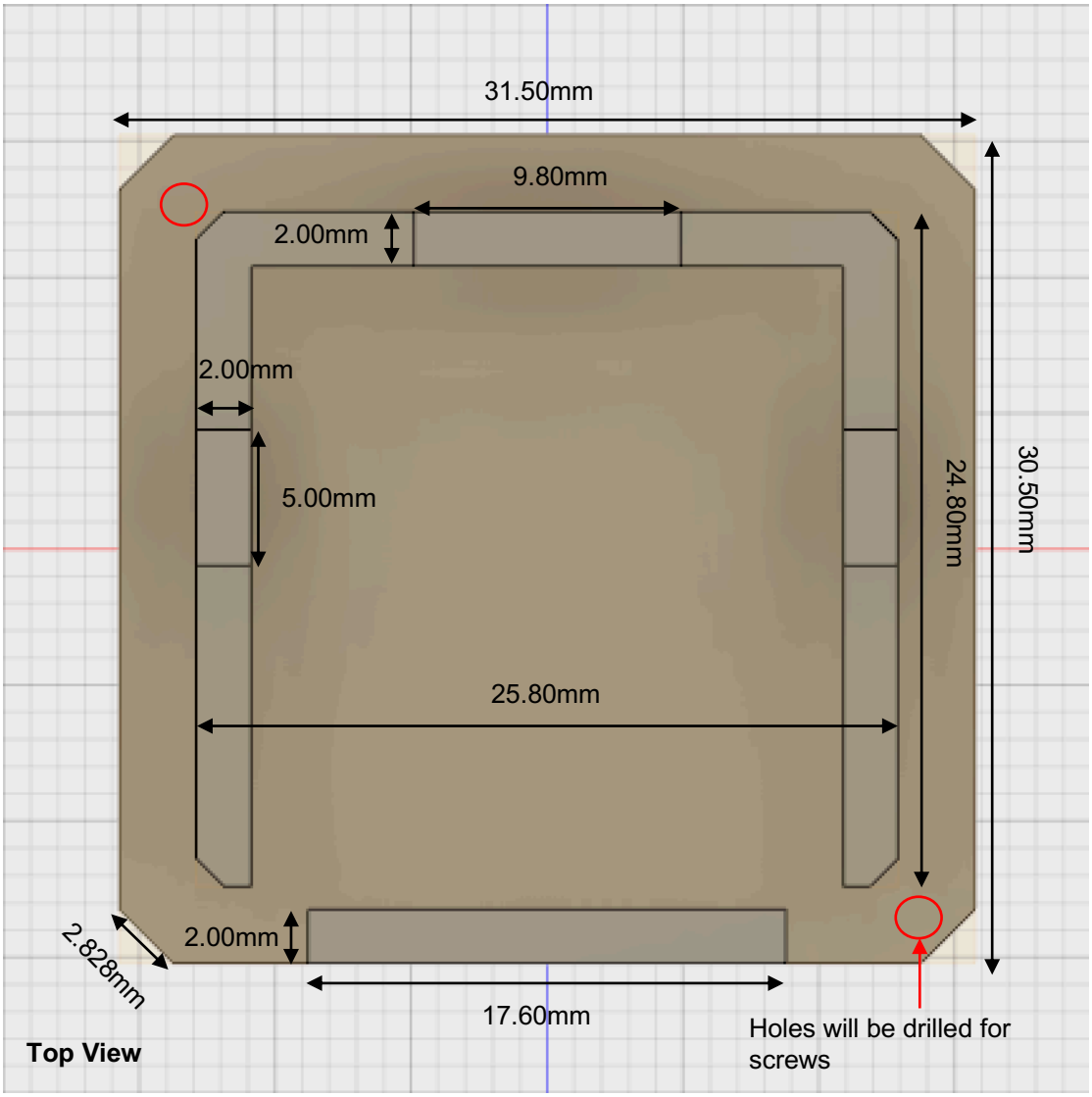

3.00mm
3.00mm
Left View

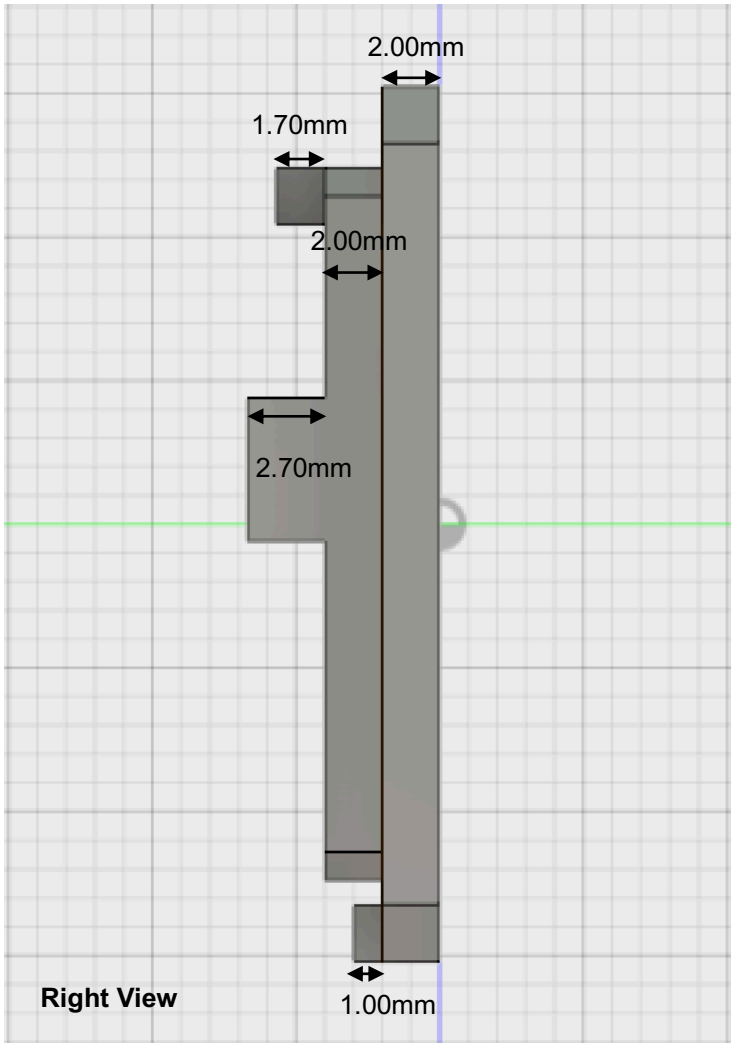
- Back Plate

The figure below is the back plate of the camera case, two holes will be drilled on the diagonal of the plate for screws.



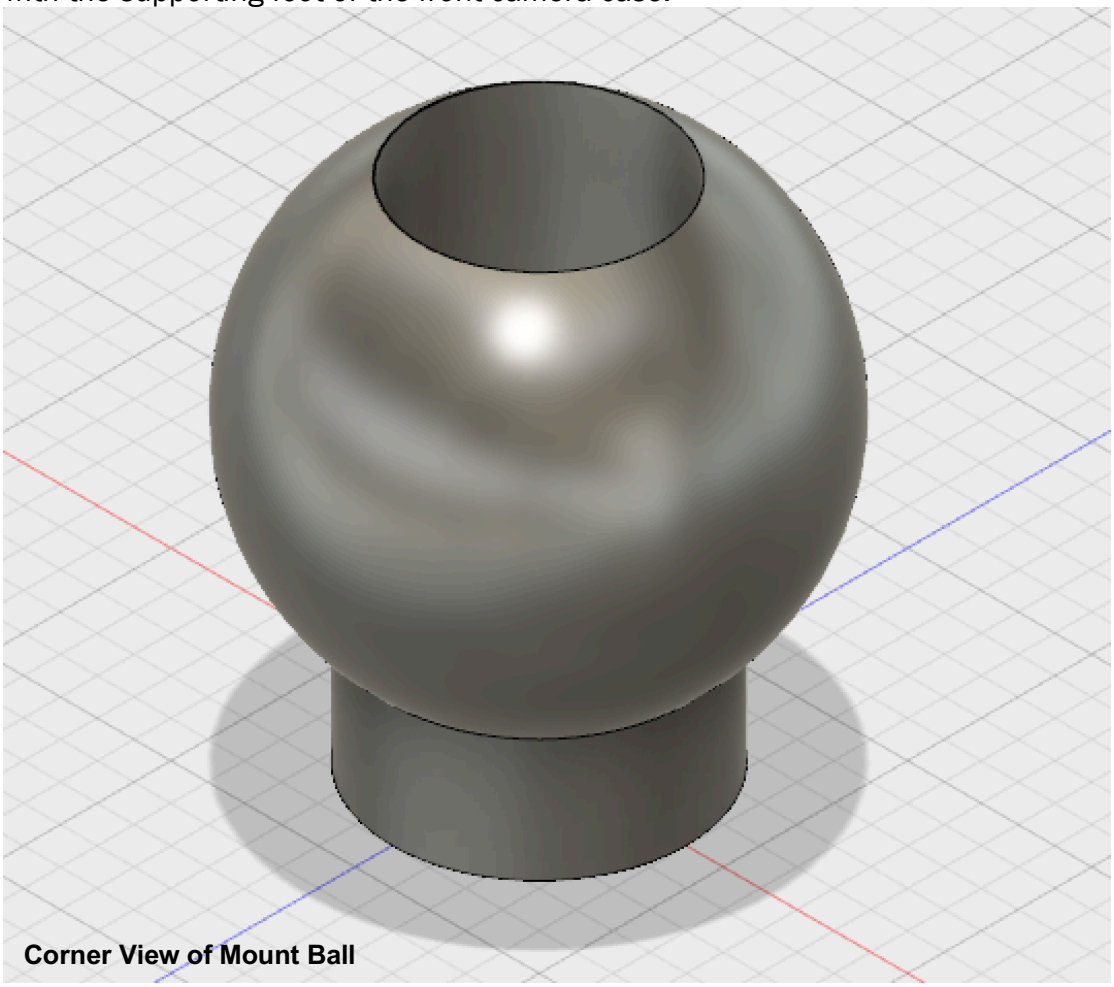
Corner View of Camera Back Plate


31.50mm
9.80mm
2.00mm
2.00mm
5.00mm
24.80mm
30.50mm
25.80mm
2.828mm
2.00mm
17.60mm
Holes will be drilled for screws
Top View

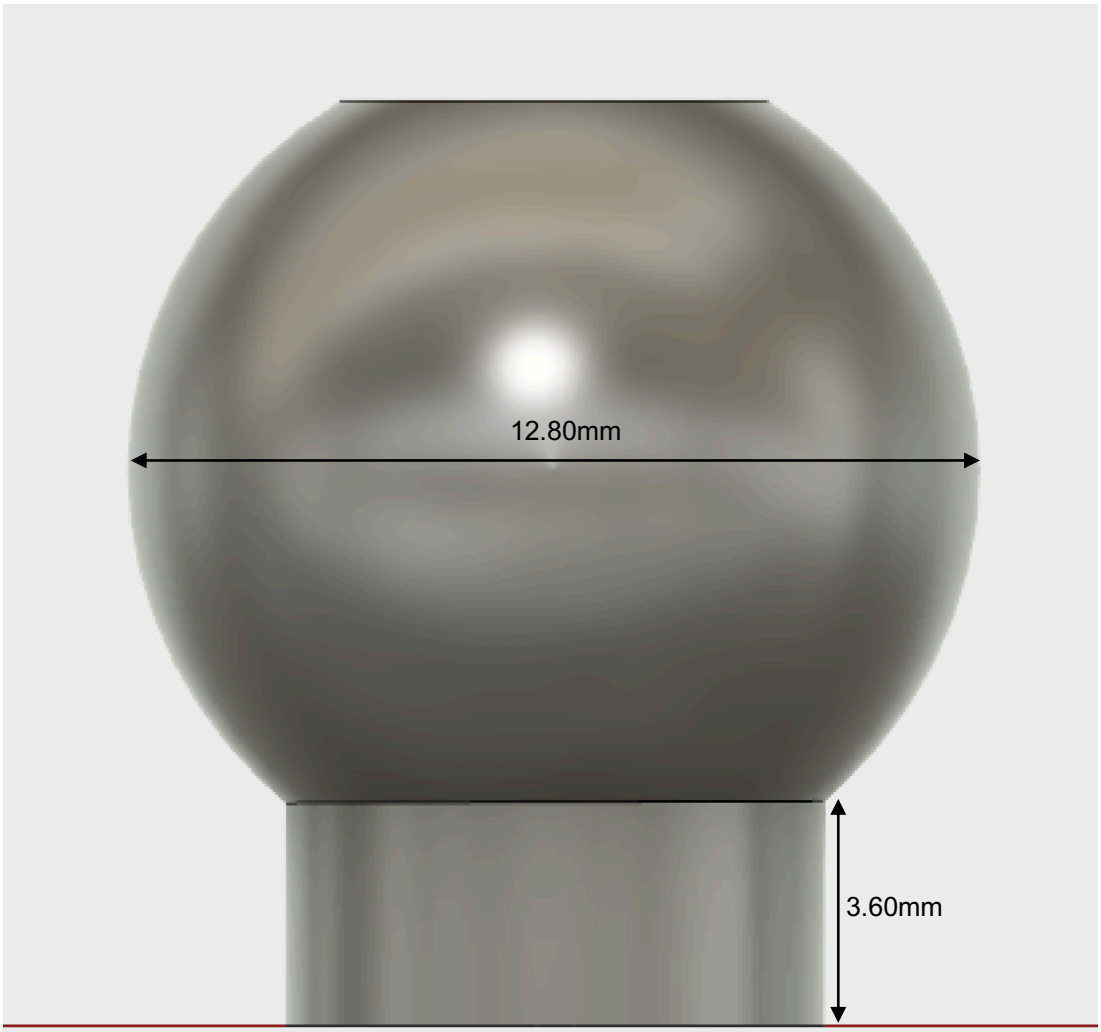

2.00mm
1.70mm
2.00mm
2.70mm
1.00mm
Right View

Mount Ball

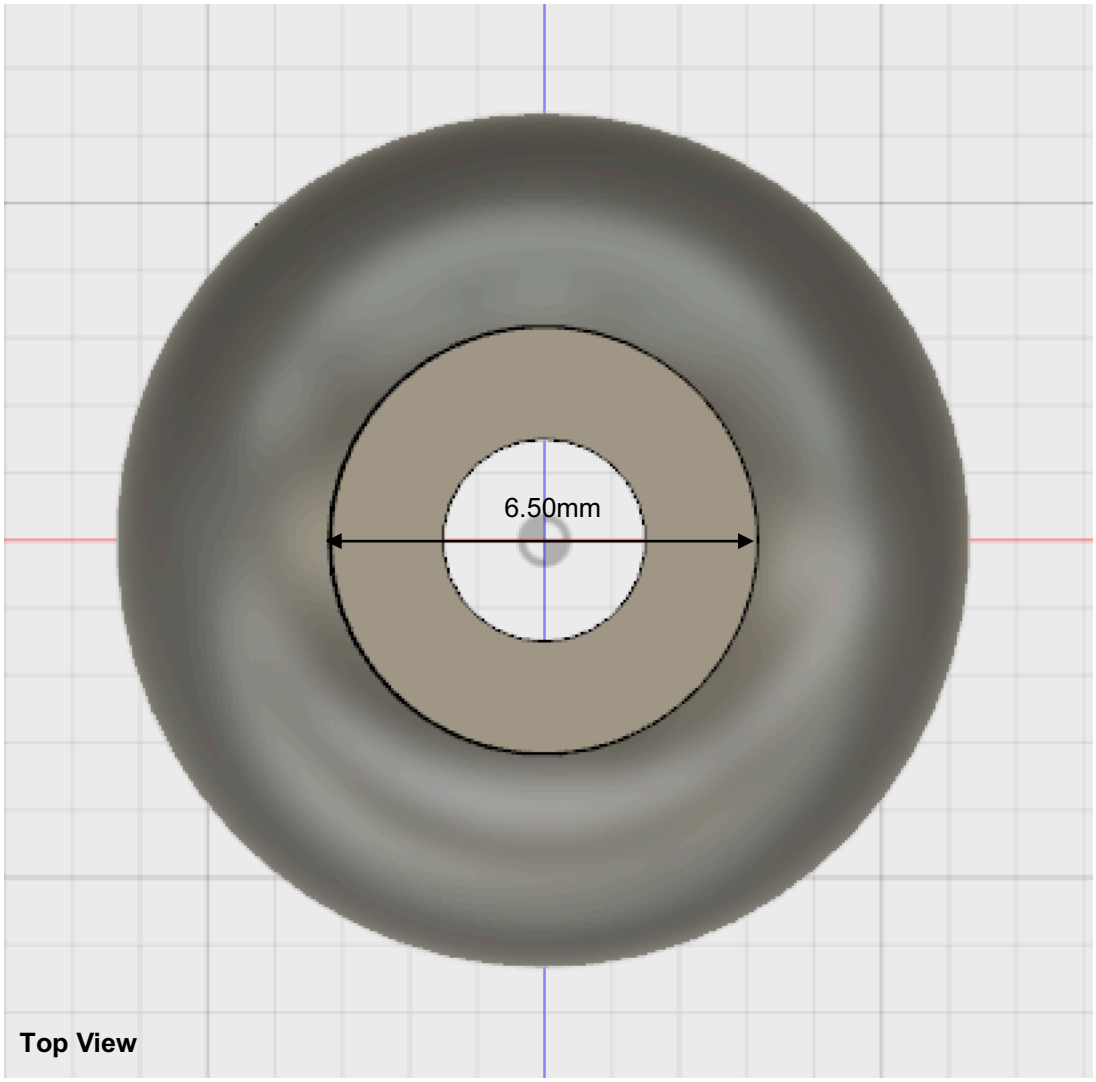
A mount ball was designed to hold the camera case and was placed in order to intersect with the supporting foot of the front camera case.

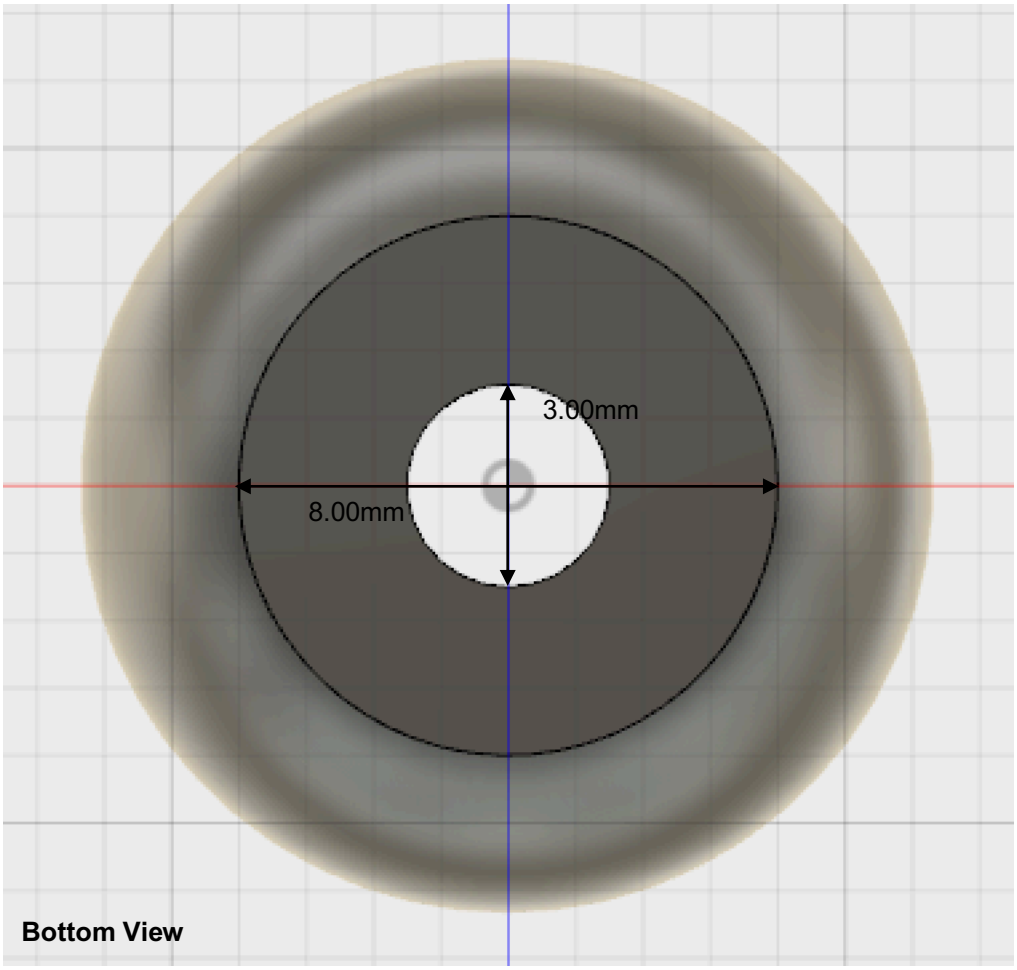


Corner View of Mount Ball



Front View

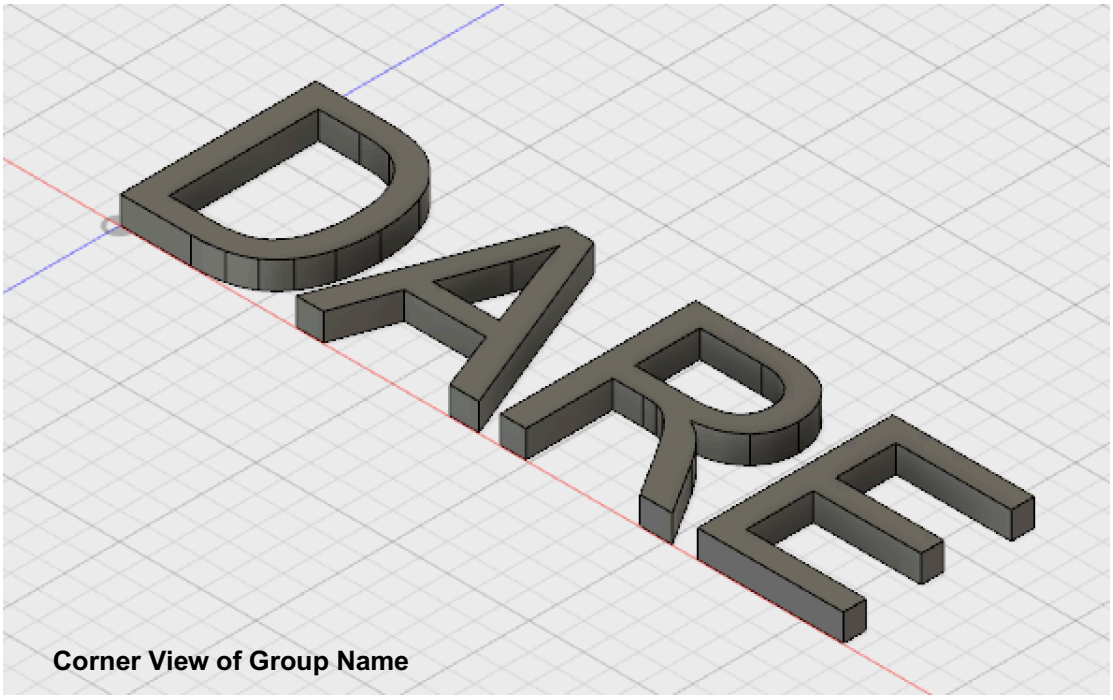

6.50mm
Top View



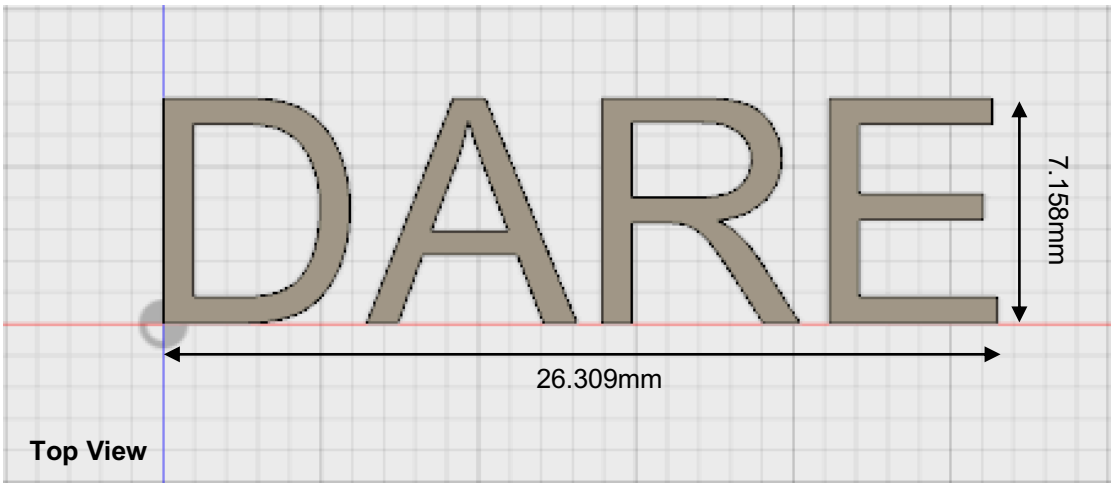
Bottom View

<u>Product Name</u>

The product name DARE was also 3D printed, and glued on the case.



Corner View of Group Name



5. Final Product

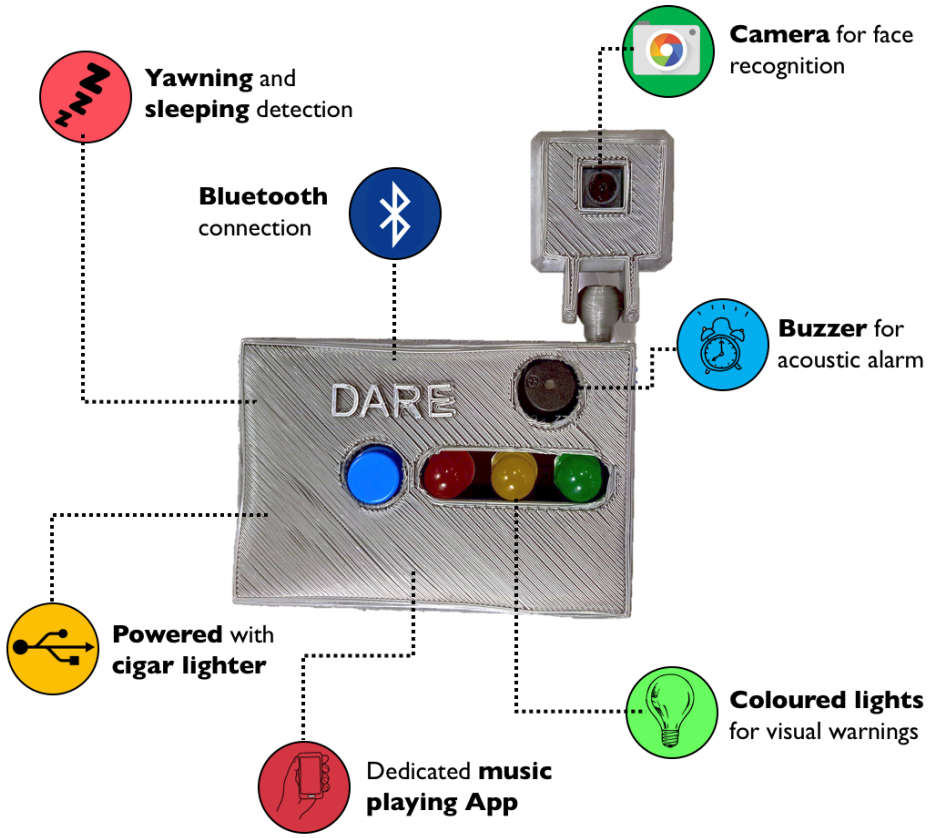


Figure 10: Final prototype with most important features

Material Used

1. 3D printed case made of Polylactic Acid
2. Raspberry Pi 3 Model B+
3. 32GB SD card with operating system NOOBS
4. Raspberry Pi HD Pi NoIR Camera Board V2

Specifications

Size and Weight
- Height: 3.50 inches (90.0 mm)
- Width: 2.40 inches (61.0 mm)
- Depth: 1.06 inch (27.0 mm)

Processor
- 1.4GHz
  Broadcom BCM2837B0, ARM Cortex-A53
  64-bit SoC @ 1.4GHz

Storage
- 32GB
  32GB Raspberry Pi SD Card

Memory
- 1GB LPDDR2 SDRAM

Connectivity
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE

Multimedia
- H.264, MPEG-4 decode (1080p30)
- H.264 encode (1080p30)
- OpenGL ES 1.1, 2.0 graphics

Input Power
- 5V/2.5A DC via micro USB connector

Environment
- Frequency: 50Hz to 60Hz
- Operating temperature: 0° to 50° C

Display
- Traffic HAT for Raspberry Pi with 3 giant LEDs, a button and a buzzer

Camera
- Sony IMX 219 PQ CMOS image sensor in a fixed-focus module
- Resolution: 8-megapixel
- Max image transfer rate
- 1080p: 30fps (encode and decode)
- 720p: 60fps
- Still picture resolution: 3280 x 2464
- Max image transfer rate:
  1080p: 30fps (encode and decode)
  720p: 60fps
- Automatic exposure control
- Automatic white balance
- Automatic band filter
- Automatic 50/60 Hz luminance detection
- Automatic black level calibration

Algorithm
- Drowsiness detection Algorithm

Start-up time
- 1 min for facial recognition by camera

App
- Music playing when drowsiness is detected
- Connect with Apple Music
- Bluetooth 4.0 connection

Features
- Drowsiness and distraction detector
- Intelligent face monitoring algorithm
- Intuitively smart recognition system
- Blinking and yawning detection
- Infrared camera for dark environment
- Audible chime
- 3 levels of LED alert for early stage warning
- Comprehensive interactive app for IOS
- Detection/ distinction of glasses and sunglasses
- Bluetooth connection to smartphone
- Easy installation

Safety
- International standard ISO26262

Languages
- **Language support**
  English (Australia, UK, U.S.)

In the Box
- DARE driver alert and recognition equipment
- micro USB connector
- Documentation

6. Functional Safety and Testing

Advanced Driver-Assistance Systems are created to enhance the driving experience but can also include potential risks, so functional safety was a major concern to the group. As outlined in the ISO 26262 standard, "Functional safety is the absence of unreasonable risk due to hazards caused by malfunctioning behaviour of electrical / electronics systems." [8] Therefore, several tests regarding functional safety were conducted to ensure the device is safe to use even in the case of malfunction or abnormal situations.

1. Camera

In order to test the scenario where the camera malfunctions, the team cut off the connection between the camera module and the Raspberry Pi before and while the device is running. In both cases, the green light turned off successfully which would warn the user of a malfunctioning behaviour.

The team also tried changing the ambient light from very bright to very dim in order to simulate the changing daylight of a typical day. Thanks to the infrared camera, it performed well in all light conditions.

Also, as the position of the device would probably change from a car to another, 6 dashboard positions were tested to determine how it affected the performance of drowsiness detection.

The camera was placed at different angles with respect to the face, changing of 15 degrees between each position. The conclusion was that as the camera was moved more and more sideways the results became increasingly worse, especially when more than one faces were detected, which could happen for example during family trips. The algorithm was calculating EAR values of every face, leading to false results. The solution to such issue is simply to make the device face the user, as only the headrest would be behind him.

2. Reaction time

To ensure the device is able to warn the driver at the right time, the reaction times of the buzzer and LEDs were measured (see Figure 11 below). The reaction time is defined as the time between user exhibiting drowsiness and the buzzer ringing/LED blinking.

Reaction Time(s)	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10	Avg.
Buzzer	0.7	0.65	0.63	0.67	0.71	0.63	0.64	0.66	0.62	0.63	0.654
Yellow LED	1.65	1.68	1.59	1.73	1.71	1.78	1.67	1.54	1.67	1.70	1.672
Red LED	1.21	1.03	1.11	0.98	1.24	1.15	1.06	1.14	0.92	1.08	1.092

Figure 11: Table of reaction time tests

It can be seen that the reaction times were relatively fast, as expected. An average of 0.654 seconds for the buzzer is explained by the fact that if we reduced the reaction time (by reducing the number of consecutive frames where the eyes need to be detected as closed), eye blinking would be detected instead, which is undesirable. However, if the driver closes his eyes for more than half a second, there must be something wrong, and the alarm should be triggered. On the other hand, as immediate yawning detection isn't crucial, we can tolerate higher reaction time averages.

3. Bluetooth connection

Our prototype can still function well even if the Bluetooth connection is lost because the detection algorithm and BLE connection algorithm are written in two independent threads. During the simulation, when the connection was lost, the string "connection lost" was printed in the middle of the iPhone's screen but the rest of the drowsiness detection algorithm functioned normally, as desired. The test was conducted 10 times and the success rate was 100%.

4. Robustness/Stability of the case

The material used for 3D printing is PLA (polylactic acid) whose physical properties are shown below. [9]

Property	Value
Technical Name	Polylactic Acid (PLA)
Chemical Formula	(C3H4O2)n
Melt Temperature	PLLA: 157 - 170 °C (315 - 338 °F)
Typical Injection Moulding Temperature	PLLA: 178 - 240 °C (353 - 464 °F)
Heat Deflection Temperature (HDT)	49 - 52 °C (121 - 126 °F) at 0.46 MPa (66 PSI)
Tensile Strength	PLLA: 61 - 66 MPa (8840 - 9500 PSI)
Flexural Strength	PLLA: 48 - 110 MPa (6,950 - 16,000 PSI)
Specific Gravity	PLLA: 1.24
Shrink Rate	PLLA: 0.37 - 0.41%

According to the table the material has good strength, but the only problem is its relatively low heat deflection temperature (49 - 52 °C). To check if such temperature could be reached on the processor, we let the full algorithm run for a whole day. At the end, its temperature had been maintained to 42 °C thanks to the heat dissipation holes on the bottom of the case. However, the temperature in a car parked in direct sunlight can reach more than 60 °C, so for product safety reasons we recommend keeping the device outside the car during summer.

5. Restart time

The button was programmed as a reset pin. That is, it will restart the whole operating system instantaneously if the button is pressed. The booting up of the whole system takes a long time, that needed to be measured for safety reasons. The restart time is defined as the time between the user pressing the restart button and green LED turning on (meaning the algorithm has reached the drowsiness detection code). Ten measurements were taken, and the average time was calculated.

	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10	Avg.
Time(s)	75	78	76	83	81	79	75	77	81	79	78.4

Figure 12: Booting up time measurements

The average was found to be 78.4 seconds, which was considered as reasonable. Normally the device is powered with a cigar lighter, which means that when the driver enters his vehicle, the device will boot up as he buckles up and starts the engine, so he won't waste any noticeable extra time using this device.

6. Accuracy

The drowsiness detection's accuracy may vary depending on the multiple factors. Indeed, eye size and shape will vary from person to person, but even more from continent to continent. This makes the predefined EAR threshold value difficult to choose and the device may not work as well for everyone depending on their origins. Additionally, wearing a hat or glasses may lead to difficulties for dlib's facial landmark predictor to correctly locate the user's eyes, which might affect the accuracy.
Therefore, four students from Imperial College were invited to test the prototype. Five sleeping and yawning tests were performed for each condition (see the Figure 13 below), the number of successes was then recorded.

Testers and facial features	Wearing Conditions (with or without glasses or hat)	Success times Eye Closing Test	Success times Yawning Test
Student 1: Asian Normal eye size	No glasses, no hat	5/5	5/5
	Only glasses	5/5	5/5
	Only hat	5/5	5/5
	Glasses and hat	4/5	5/5
Student 2: European Normal eye size	No glasses, no hat	5/5	5/5
	Only glasses	5/5	5/5
	Only hat	5/5	5/5
	Glasses and hat	4/5	4/5
Student 3: European Relatively big eyes	No glasses, no hat	5/5	5/5
	Only glasses	4/5	5/5
	Only hat	5/5	5/5
	Glasses and hat	4/5	4/5
Student 4: Asian Relatively small eyes	No glasses, no hat	3/5	4/5
	Only glasses	3/5	4/5
	Only hat	4/5	5/5
	Glasses and hat	2/5	4/5

Figure 13: User type and accessories tests

To conclude, for people with normal or big eye size (European and Asian), this device works almost perfectly. However, the accuracy did deteriorate for an Asian student with small eyes. The accuracy of the eye closing detection was found to be worse than the yawning detection overall. This inaccuracy should be improved, which we will develop in the “Future work” section in the “Conclusions” document.

References

[1] Raspberry Pi. Download Raspbian for Raspberry Pi. [ONLINE] Available at: https://www.raspberrypi.org/downloads/raspbian/. [Accessed 27 June 2018].

[2] PyImageSearch. Raspberry Pi: Facial landmarks + drowsiness detection with OpenCV and dlib - PyImageSearch. [ONLINE] Available at: https://www.pyimagesearch.com/2017/10/23/raspberry-pi-facial-landmarks-drowsiness-detection-with-opencv-and-dlib/. [Accessed 27 June 2018].

[3] PyImageSearch. Running a Python + OpenCV script on reboot - PyImageSearch. [ONLINE] Available at: https://www.pyimagesearch.com/2016/05/16/running-a-python-opencv-script-on-reboot/. [Accessed 27 June 2018].

[4] OpenCV: Face Detection using Haar Cascades. OpenCV: Face Detection using Haar Cascades. [ONLINE] Available at: https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html. [Accessed 27 June 2018].

[5] Tereza, Soukupova´.and Jan, C˘ech. 2016. Real-Time Eye Blink Detection using Facial Landmarks. Rimske Toplice, Slovenia: Center for Machine Perception, Department of Cybernetics Faculty of Electrical Engineering, Czech Technical University in Prague, p.1.

[6]Emma Yann Zhang. Understanding different roles of BLE devices - Emma Yann Zhang. [ONLINE] Available at: http://emmayannzhang.com/understanding-different-roles-of-ble-devices/. [Accessed 23 June 2018].

[7]GATT Services. GATT Services |Bluetooth Technology Website. [online] Available at: https://www.bluetooth.com/specifications/gatt/services [Accessed 23 Jun. 2018].

[8]ISO 26262 - Wikipedia. ISO 26262 - Wikipedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/ISO_26262. [Accessed 23 June 2018].

[9]Tony Rogers. Everything You Need To Know About Polylactic Acid (PLA). [ONLINE] Available at: https://www.creativemechanisms.com/blog/learn-about-polylactic-acid-pla-prototypes. [Accessed 23 June 2018].