Parking Availability System

Beta Test Plan

Erik Meurrens, Benjamin Simonson, Ryan Jalloul, Evan Tobon, Samer Khatib

Alpha Test Results

Raspberry Pi License Plate Detection Pipeline

A goal of the project is to get as close as possible to an inference speed of 10 frames per second (100 milliseconds) without significantly sacrificing model performance scores. Utilizing processed image inputs into the model at a size of 640x384 pixels, we were able to achieve the resulting statistics in Table 1 on the Raspberry Pi 5, without any deliberate tuning or changes of the model to improve inference speed.

TABLE 1. License Plate Detection Inference Speed

Input Size	MinTime	MeanTime	MaxTime	MaxFPS	MeanFPS	MinFPS
640x384	203.4	271.14	373.4	4.916	3.688	2.678

Even under ideal conditions, the YOLOv11n model achieves just under half of the stated goal of 10 frames per second. To get closer to our stated goal in developing our Release Candidate, we aim to improve our performance by investigating model optimization methods, such as dimensionality reduction and pruning, hardware/software optimization methods, such as ensuring maximum core utilization and overclocking the processors, or alternative models to get closer to our 10 frame per second inference goal.

Raspberry Pi Persistent State

A goal of the RPi's persistent functionality was that it could perform its task of taking a picture, inferencing data from the picture, and manipulating the results accordingly without trouble. While the RPi demonstrated consistent success in recognizing license plates with high confidence, see Images 1 and 2 below, it failed to consistently operate from available mobile power sources such as being plugged into a laptop or battery supply. At the time of testing, there was no available power source that could drive the RPi without needing to be stationary. While this point is not a requirement in the project, it severely hinders the capacity of testing the RPi as it interferes with the mobility of the user testing it. Since we are bound to the location of a power outlet and the length of a substantial power cord. Driving the RPi from the batteries available and from various laptops did not produce the required voltage and amperage to drive the RPi without failure as seen in Image 3 it fails to reach the goal of 5V 3A. All this means though is that further testing will be required where a wall outlet is available, and the RPi can still maintain appropriate field of view on the garage setting.

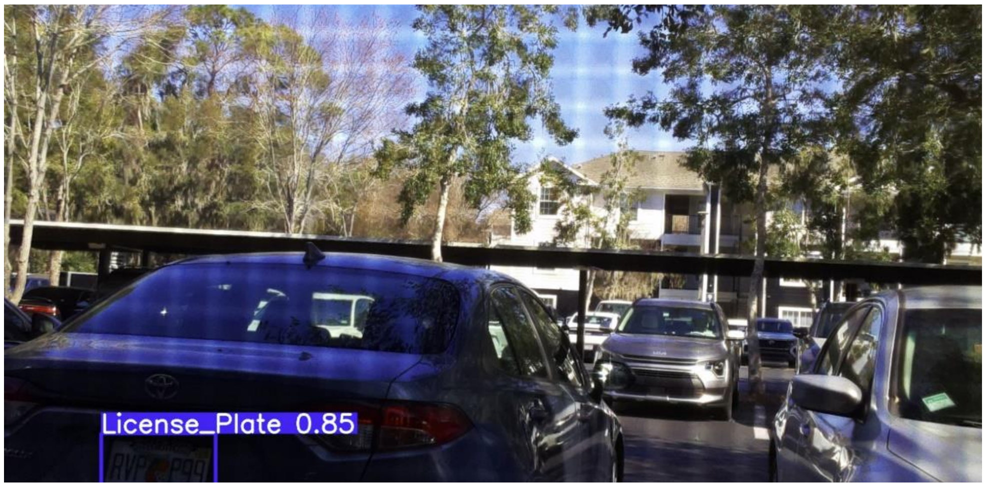
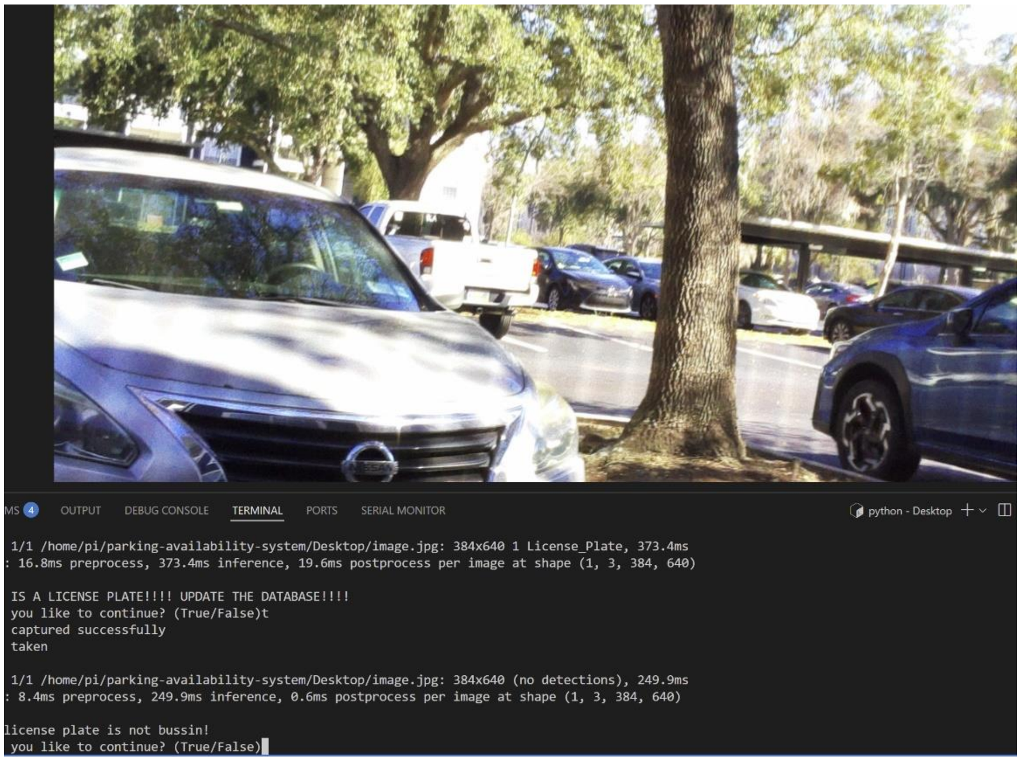


Image 1 shows the RPi working from the window of an apartment facing a parking lot



```
 1/1 /home/pi/parking-availability-system/Desktop/image.jpg: 384x640 1 License_Plate, 373.4ms
: 16.8ms preprocess, 373.4ms inference, 19.6ms postprocess per image at shape (1, 3, 384, 640)

 IS A LICENSE PLATE!!!! UPDATE THE DATABASE!!!!
 you like to continue? (True/False)t
 captured successfully
 taken

 1/1 /home/pi/parking-availability-system/Desktop/image.jpg: 384x640 (no detections), 249.9ms
: 8.4ms preprocess, 249.9ms inference, 0.6ms postprocess per image at shape (1, 3, 384, 640)

license plate is not bussin!
 you like to continue? (True/False)
```

Image 2 shows that when no license plate is detected, the RPi outputs the correct statement indicating that all that is left is inserting the appropriate HTTP request to handle this output

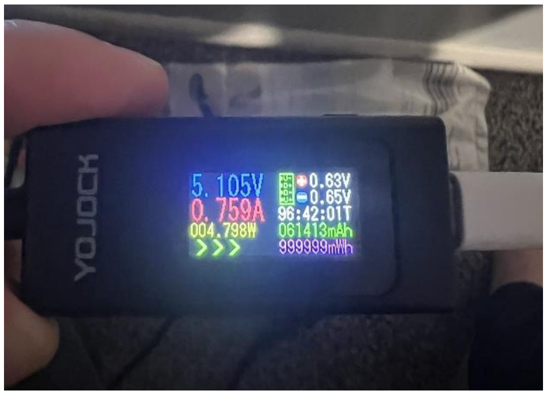


Image 3 shows the RPi failing to be supplied with the minimum of 3 amps for appropriate functionality

Raspberry Pi Configuration Script Testing

Despite not being listed in the Alpha Test plan, it was necessary to develop and test a configuration script to ensure that it can set up a fresh Raspberry Pi (yummy) so that the system can be scaled with additional devices to ensure accurate inflow and outflow counts from a parking garage, configuration settings are documented, and that test devices that would need to be reformatted can be reconfigured quickly. The configuration script was tested by reinstalling Raspberry Pi OS and running the script repeatedly to search for bugs or errors in the user control flow, testing all user flow branches for expected output. The maximum execution time for the configuration script on a fresh Raspberry Pi 5 is within the range of 15-20 minutes.

Raspberry Pi WiFi Antenna Testing

To ensure a reliable and stable connection to the University of Florida network, many possible solutions were investigated during the Alpha Build sprint to remedy a WiFi dead zone within our pilot garage for testing the Release Candidate. In addition to developing an ESP32 repeater during the Beta Build, it was deemed useful to investigate utilizing a USB WiFi modem with an antenna on the Raspberry Pi devices, as opposed to the on-board Wi-Fi modem, to maximize signal strength and stability. Wireless connection performance metrics were measured for both the on-board WiFi modem of the Raspberry Pi, as well as a USB WiFi modem with an antenna, at various locations using the following Linux command sequence,

```
iwconfig wlan0; iw dev wlan0 link; iw dev wlan0 info
```

and the resulting statistics were summarized in Table 2.

TABLE 2. Modem Performance Statistics

Modem Type	**AvgBitRate (Mb/s)**	**AvgLinkQuality**	**AvgReceivedSignalStrength (dBm)**

On-board (5 GHz)	54.53333	79%	-55
Antenna (5 GHz)	115.175	91%	-46.3333

Overall, the results demonstrate a better signal by demonstrating a higher average link quality and higher received signal strength using the USB WiFi modem, which is what we were hoping to achieve. However, a more robust test plan with more data is needed to support these results, which will likely be employed in tests with the ESP32 repeater. Regardless, the USB WiFi modem allows our team more flexibility in designing the housing, as the on-board WiFi chip may encounter signal attenuation due to the enclosure around the Pi or due to noise by other hardware components, whereas an antenna can be positioned outside of the housing.

Mobile Application

Unfortunately, the configuration script and WiFi testing distracted from mobile application testing, so mobile application user and data control flow testing will become technical debt due in the next sprint.

ESP32

While the connection to the ESP32 access point is secure, the download and upload speeds of the access point have proven to be inefficient. The security of the ESP32 access point was tested, and this was successful, as the ESP32 limited the number of clients on the network. During this test, the ESP32 remained stable although there were multiple requests being made to join the network. As a result of this testing, the plans for the ESP32 have shifted. Originally, the design of the ESP32 access point utilized the built in Wi-Fi capabilities of the ESP32. This would have solved the issue of the lack of a Wi-Fi network in the location where we are implementing the Raspberry Pi without having to extend the UF network. However, the new plan is to extend UF's eduroam network using the ESP32. This would mean that rather than placing the ESP32 directly with the Raspberry Pis, we would need to place the ESP32 about 100 feet away and extend the network. This change creates new problems as the ESP32 would now be further, and the speeds must support sending photos to the object detection model and uploading data to the database.

Expected Behavior

ESP32

The expected behavior of the ESP32 is to extend UF's eduroam network to the front of the parking garage. UF's eduroam network does not reach the entrance of the parking garage, but there is a connection about 100 feet away in the corner of the garage. The ESP32 is expected to bridge this gap between the UF network and the entrance of the garage where the Raspberry Pis will be, providing a stable connection for the Raspberry Pis.

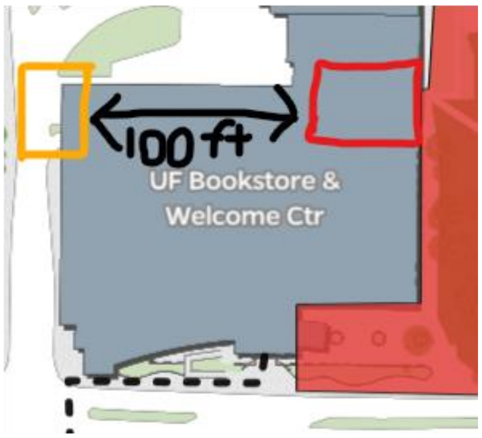
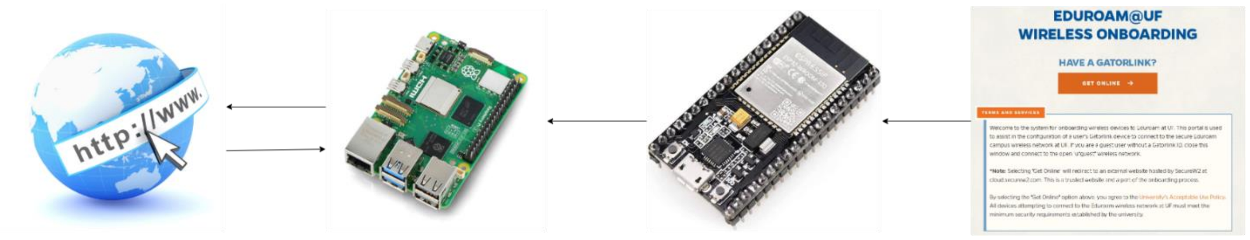


Figure 1 shows the distance between the Raspberry Pis (orange) and the UF eduroam network (red), which is approximately 100 feet.



Welcome to the system for onboarding wireless devices to Eduroam at UF. This portal is used to assist in the configuration of a user's Gatorlink device to connect to the secure Eduroam campus wireless network at UF. If you are a guest user without a Gatorlink ID close this window and connect to the open "ufguest" wireless network.

*Note: Selecting "Get Online" will redirect to an external website hosted by SecureW2 at cloud.securew2.com. This is a trusted website and a part of the onboarding process.

By selecting the "Get Online" option above you agree to the University's Acceptable Use Policy. All devices attempting to connect to the Eduroam wireless network at UF must meet the minimum security requirements established by the university.

Figure 2 shows the interaction between the eduroam network, the ESP32, the Raspberry Pi, and the internet

Raspberry Pi

The Raspberry Pi was expected to be able to take pictures, save the pictures on its drive, and then inference the using the license plate recognition model. This is the first step of the Raspberry Pi workflow that was intended to be tested before communication via the HTTP requests was to begin. The RPi is expected to be fully functional via a persistent power source providing 5V, 3A power supply, such as a wall outlet, and establish an internet connection autonomously. After this setup and power source, the RPi should take pictures and inference them in an infinite loop, all the while checking the model results and handling the outputs accordingly. This cycle can be seen in Figure 3.

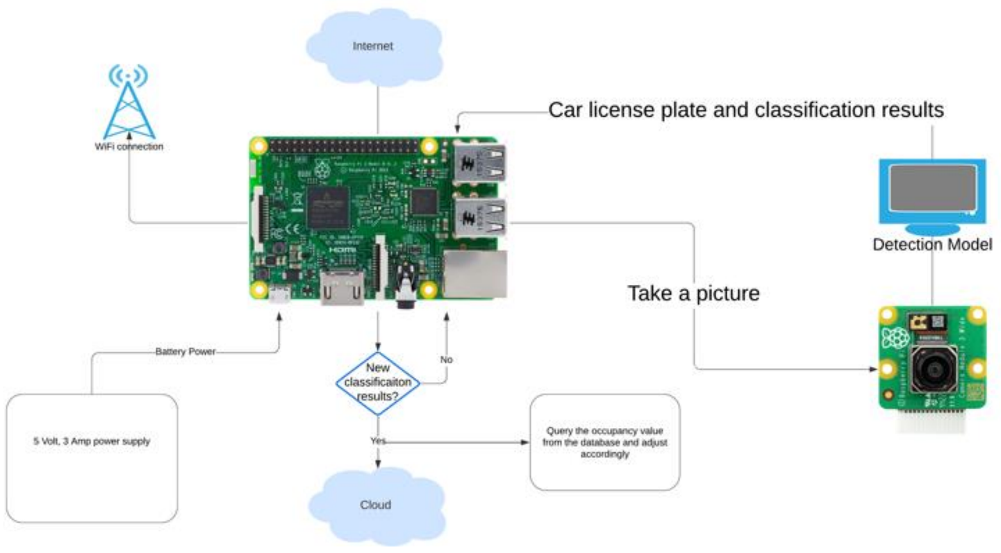

Internet
WiFi connection
Car license plate and classification results
Detection Model
Take a picture
Battery Power
New classificaiton results?
No
Yes
5 Volt, 3 Amp power supply
Query the occupancy value from the database and adjust accordingly
Cloud

Figure 3. shows the loop that should occur (on the right of the image) and the setup protocol that should be maintained persistently (on the left of the image).

Raspberry Pi Configuration Script

The purpose of the Raspberry Pi configuration script is to allow our team to configure multiple Raspberry Pi devices seamlessly to work within the system, while also documenting the steps needed to configure the Raspberry Pi devices. By running the script on a Raspberry Pi 3 Model B+ or higher, anyone with a fresh Raspberry Pi can configure their environment to run exactly as our team's devices do.

Upon cloning the "rpi_files" branch from our project's GitHub repository into a fresh Pi, a user can run the file located in the directory "parking-availability-system/Desktop/setup/pi_setup.sh". The configuration script automates the setup, though it requires user supervision as there are multiple user inputs and two restarts. The script performs the following tasks in under 20 minutes, including tasks requiring user input:

- Connection to the UF eduroam network
- Installing APT package manager packages and dependencies
- Install and configure USB WiFi modem drivers (if needed)
- Configures Raspberry Pi OS and boot settings
- Set up of the virtual environment for our program.

Mobile App (Alpha Plan, Technical Debt)

The mobile app follows a particular user flow and data flow, as seen in Figures 4. Users would open the app and be introduced to the map screen. As the app starts up, the app loads ParkingLocation objects and initializes them with hard-coded data. Following the instantiation of these objects, the app attempts to query the database for up-to-date data within the database and update the existing ParkingLocation objects. If the query failed, parking information will still be available in the app, albeit not up-to-date, and real-time availability will not be available. If successful, a ParkingLocation's corresponding information screen will have the most up-to-date information. If the initial query of the database timed out, a user can still access up-to-date information later by navigating to a particular ParkingLocation object's information screen, which queries the database periodically for updated information on that particular location. This ensures that users will still have access to the most up-to-date, real-time information, supposing nothing is wrong with the database connection.

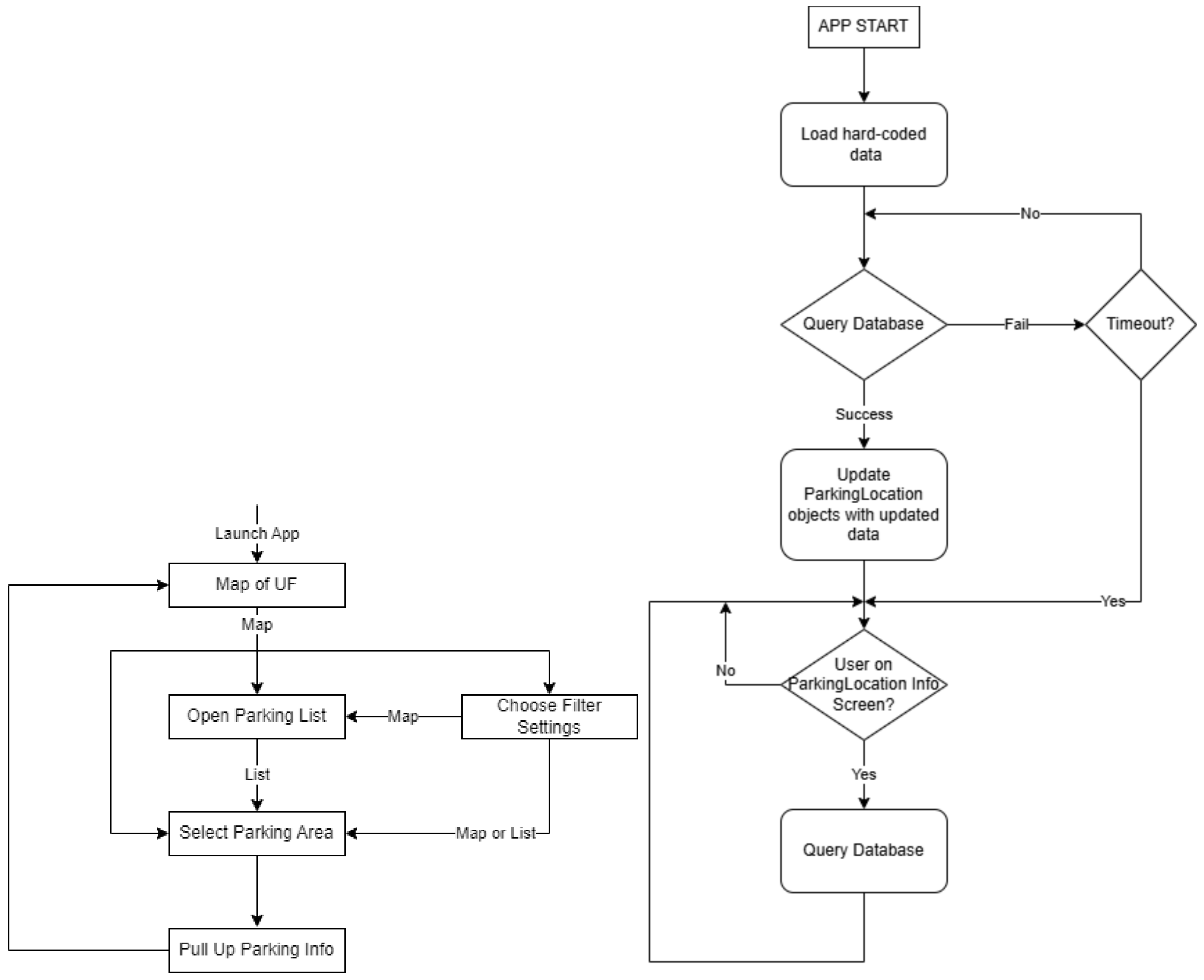


Figure 4. Mobile App User Flow Diagram (Left) and ParkingLocation Data Update Diagram (Right)

Test Procedures

ESP32 and Connection Metrics

The first step in testing the ESP32 with the new expected behavior will be ensuring that the connection between UF's network and the ESP32 is stable. The ESP32 will need to connect to UF's network and maintain a stable connection. This can be tested by monitoring the connectivity of the network over a long period of time to ensure that the connection is stable. Next, we will need to test the range of the ESP32 as a Wi-Fi extension. Previously, our design simply had the ESP32 hosting its own Wi-Fi without an external router connected. This meant that we could have the ESP32 directly next to the Raspberry Pis. However, with the new design, the ESP32 will need to be approximately 100 feet away from the Raspberry Pis. To test the speed and connectivity of the connection at different ranges, we will record certain parameters such as upload and download speeds at a given range and record these values. If these tests are successful, our next test would be to connect the Raspberry Pis to the ESP32 network and ensure that data can be sent and received. This can be done by sending an arbitrary photo to the Raspberry Pi and monitoring the database to ensure that the photo was processed and uploaded properly.

Raspberry Pi Database Communication

Since a power source capable of providing 5V 3A is not necessarily required as well will have persistent and capable power, this test is negligible. The RPi is now capable of performing the desired task and the next phase can commence. This test will be about ensuring that the HTTP requests are being made appropriately and are returning the correct results. While this is outdated, the code that will be tested can be seen here, and this will be tested by actively viewing the database and ensuring that the value the RPi receives is accurate, and the value the RPi sends is reflected in the database.

Raspberry Pi License Plate Recognition (Alpha Plan Continuation)

Investigate model optimization methods, such as dimensionality reduction and pruning, hardware/software optimization methods, such as ensuring maximum core utilization and overclocking the processors, or alternative models, to achieve 10 frame per second inference speed. After applying these methods individually, record their effect on the inference speed and model performance metrics as seen in the Alpha Plan.

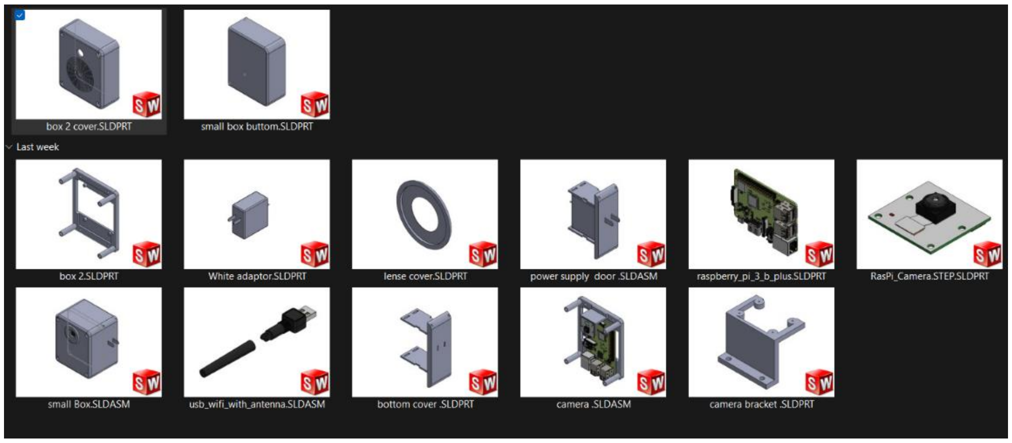
Mobile App (Alpha Plan, Technical Debt)

The mobile app will be tested by following the ParkingLocations' data states as the app runs and reaches points in the data flow diagram and user flow diagram that should result in changes in the data. This will be done by using the debugging tools within Android

Studio as the program executes. Front-end features will be tested by acting as the user and navigating the user flow diagram, looking for bugs. Additionally, database access functions can be unit tested.

Hardware/ 3D Prints

3D printing a custom enclosure for the Raspberry Pi is essential for protecting the hardware and ensuring a stable setup for photography and mounting applications. A well-designed case not only shields the Raspberry Pi from dust, impact, and environmental factors but also provides a secure way to mount it on a tripod for capturing clear and stable images. When working on this project, we had to wait a couple of weeks for key hardware components, such as the camera module and additional add-ons, to arrive, which were crucial in optimizing the enclosure's design and minimizing its overall size. This delay allowed us to carefully plan the case's dimensions to ensure a compact form factor while maintaining ventilation and cable management. The integration of a tripod mount was particularly important, as it allows for precise positioning, reducing vibrations and improving image quality. Unlike off-the-shelf cases, our 3D-printed design gives us full control over essential features like ventilation, access to ports, and structural reinforcements for durability. By leveraging 3D printing, we created a highly functional, space-efficient enclosure that enhances the Raspberry Pi's usability while providing a professional and reliable mounting solution for photography and other applications.


box 2 cover.SLDPRT
small box buttom.SLDPRT
Last week
box 2.SLDPRT
White adaptor.SLDPRT
lense cover.SLDPRT
power supply door .SLDASM
raspberry_pi_3_b_plus.SLDPRT
RasPi_Camera.STEP.SLDPRT
small Box.SLDASM
usb_wifi_with_antenna.SLDASM
bottom cover .SLDPRT
camera .SLDASM
camera bracket .SLDPRT