Parking Availability System

Release Candidate

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

Usability

Interface

Mobile App

Users will be able to interact with the system and retrieve real-time parking facility data using a mobile app developed as part of the system. As seen in the figure below, users will be able to navigate a map of the campus and view the locations of parking facilities plotted on the map, colored according to their corresponding decals. Users can either select a location on the map page, or search for a location on the list view page. Additionally, users can filter locations based on decal, name, and restriction times, which affects the locations available on the map and list view. Once a user has selected a parking location, they will be able to view the facility's parking information in the parking details page on the bottom ribbon. This page's details are updated periodically by polling the LOT table tracking parking facilities in the backend PostgreSQL database.

For the Release Build, the user interface of the app has changed slightly. To better reflect the occupancy of a parking garage to the user in the lot info screen, the occupancy is now displayed using a circular percentage gauge to demonstrate lot "fullness", where the color of the gauge represents the occupancy state of the lot (i.e. red if at least 80% full, yellow if between 80% and 40% full, and green below 40% full). Additionally, instead of stating in text the number of cars parking in the lot (i.e. occupancy), as was the case with the Design Prototype version of the app, the app's new occupancy gauge states in text an estimate of the parking spaces still available in the lot instead. These changes are intended to make the lot info screen more appealing and clear to users.

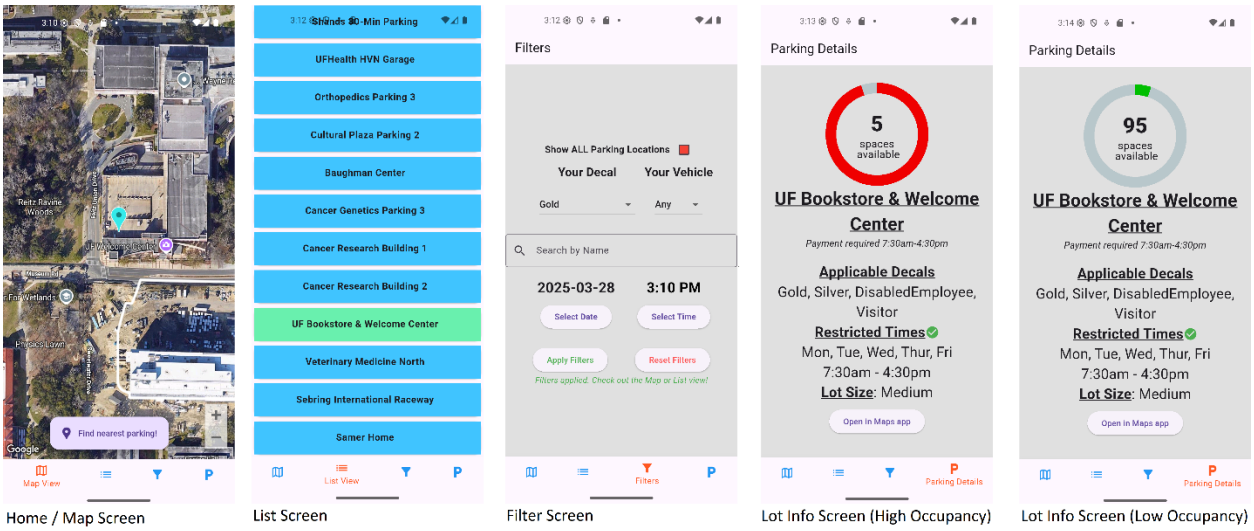


Figure 1. Parking App Frames

Pi Configuration Script (Last Updated: Beta Build)

The Raspberry Pi configuration script's target user is not the primary target user of the system, that is commuters and people looking for parking on campus, but system administrators and those looking to deploy the system in a parking garage. Additionally, it acts as a quality-of-life feature for us as developers of the system. The purpose of the script was to ensure that a fresh Raspberry Pi can be set up in a more automated, seamless way so that scaling the system to other garages could be feasible, configuration options and settings for the Pi can be documented, and that test devices could be reformatted easily if needed.
The Raspberry Pi configuration script is a bash script run via the command line. Before using the script, it must be downloaded from the public project repository using the command `git clone https://github.com/emeurrens/parking-availability-system.git -b rpi_files`. The user can then run the script by navigating to the clone repository directory `parking-availability-system/Desktop/setup` and running the command `bash pi_setup.sh`. Afterward, the user interacts with the script by following the script's instructions in the command line, providing keyboard input when requested by the script to control the execution and configuration settings.

Navigation

Mobile App

The mobile app currently implements all the features necessary to be able to view real-time parking facility updates by reading entries from the LOT table in the backend database. The UI is intuitive, and users can quickly navigate to the parking information screen for a specific parking facility in no more than three taps. Navigation of the UI takes inspiration from other navigation apps, so controls should feel intuitive, especially for frequent users of such apps. Additionally, it implements Google Maps API functionality to provide users with an accurate and familiar top-down view of the campus, and allow them to navigate to pinned lots using their preferred navigation app. Finally, with the resources provided by a previous endeavor to improve parking observability systems, the app can also filter facilities on campus by the parking pass clearance you have, for example: red, green, orange, etc, allowing users to drive to lots available to their decal.

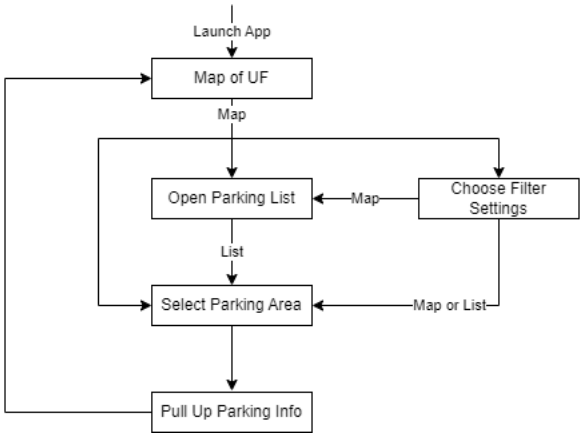


Figure 2. App User Navigation Flow Chart

Pi Configuration Script (Last Updated: Beta Build)

The user is guided through the script by prompted instructions in the command line; as long as they are capable of reading and typing, the user can easily navigate the script. The user flow diagram in Figure 2 demonstrates the flow of script execution from the user's point of view.

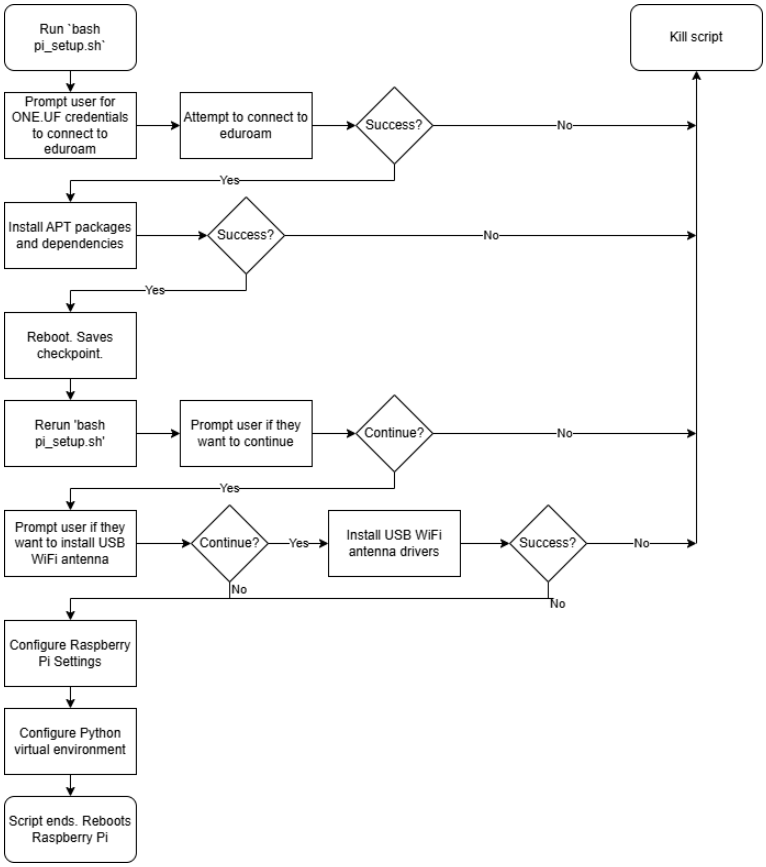


Figure 3. Pi Configuration Script User Flow Diagram

Perception

Mobile App

Within the mobile app, users are presented with a ribbon bar to navigate between four page states. The associated page visible on the screen is highlighted within the ribbon bar in orange, whereas the rest are blue, indicating the pages can be selected or moved through. Selecting another page is associated with a change in the page state. Additionally, within the mobile app, users can navigate the map intuitively with touch controls akin to many similar navigation apps. Parking locations can be selected by clicking on the location within the map view or list view, which centers the map in the map view onto that location. Additionally, selected parking locations are highlighted in the map view with a cyan pin and with an information pop-up on top, or in the list view with a green background on the parking location's card in the list.

As of the Release Candidate, when a user accesses a lot info screen, they are given feedback of updates in parking lot occupancy by changes of the occupancy gauge's appearance, whether that is by the number of available spaces for that lot actively changing on the screen, the color of the gauge changing on the screen, or by the animation that occurs to signify a change from the previous occupancy number to the new occupancy number.

Pi Configuration Script (Last Updated: Beta Build)

Users interact with the Pi configuration script through the command line, as mentioned above in the "Interface" section. The script guides users through its execution with prompts when user interaction is needed. Additionally, the script prints verbose output from the commands it runs, and any important updates, e.g. successful execution or failures.

Responsiveness

Mobile App

In the mobile application, users should always have access to the most up to date information according to what is stored in the LOT table. The table is polled at the start of the app, and periodically (currently at 5 seconds intervals) for a specific location while that location's info screen is open. Under conditions where there's a failure to connect to the table, parking location information is also hard-coded into the codebase, which allows users to always have access to some form of parking information, even if it is not up-to-date.

Pi Configuration Script (Last Updated: Beta Build)

The configuration script notifies users with a prompt when they need to interact with the script. Since the configuration script is executed on the command line, responsiveness is

near immediate. Execution time, however, is not immediate, and accounting for user-required input, full execution of the script, the longest path in the user flow diagram, takes 15-20 minutes to execute to completion. However, once configured, the user will never have to interact with the device or script again unless they change the Wi-Fi SSID or need to update something on the RPi itself.

Raspberry Pi

The Raspberry Pi has ample functionality of performing its task with relative speed. It executes the same order of tasks repeatedly, first taking and saving a .JPG file, then inferencing the .JPG file, extracting and manipulating results which includes checking whether a license plate was detected and saving the resulting image with a bounding box. If a license plate was determined to be detected, the database is updated as a car has either entered or left the parking facility depending on the RPi configuration as an “exit” or an “entrance” RPi. Through testing, we can observe the performance metrics in Table 1.

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

Using a more observable metric, observe Figure 4 and 5. Respectively, Figure 4 shows the first frame that includes a test car in a garage. Figure 5 shows the next available frame taken by the RPi. For reference, the test vehicle was moving at about 15 MPH which is an above average speed for garage driving operations, yet it still managed to capture the vehicle as it passed the RPi. This suggests that the RPi component and onboard model is responsive and effective at encapsulating its environment at a near “real-time” speed.

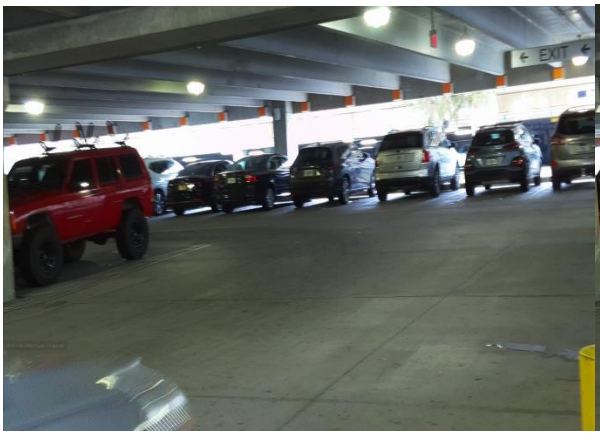


Figure 4. Bottom left side shows the test vehicle entering the frame

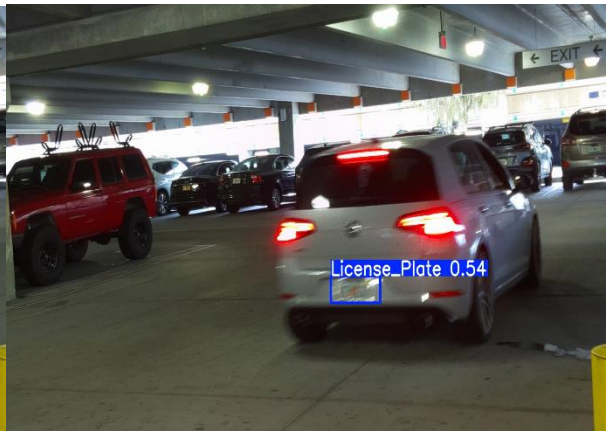


Figure 5. Shows the next frame captured by the RPi of the test vehicle moving 15 MPH past the RPi

ESP32

One glaring problem the RPi faced was establishing a stable connection to the internet in some settings. The Reitz Union garage proved to be an environment with an unstable internet connection causing RPi operations to hang as it was unable to transmit data to the database over Wi-Fi. Many solutions were considered, however, the ESP32 seemed the most capable and achievable the fastest. The ESP32 would serve as a Wi-Fi extension hotspot to expand the range of an SSID. For example, it would be set up at around the midpoint of the Reitz Union and the Reitz Union garage while being connected to the UF Wi-Fi. The RPi would then connect to that and have a much stronger and more stable connection compared to just trying to connect to the UF Wi-Fi itself. Testing proved that the ESP32 is more than capable of hosting a stable connection even through physical impediments and distances as seen in figure 5. The ESP32 was tested outside at approximately 45 feet of range from a Wi-Fi router. The router was kept inside an apartment building which would simulate the task the ESP32 would have to perform. Despite the range and building obstruction the laptop connected to the ESP32's Wi-Fi hotspot was able to establish the below results. Under these conditions, the RPi would have no problem in establishing a connection to the ESP32 and transmitting data from areas with a poor connection service.

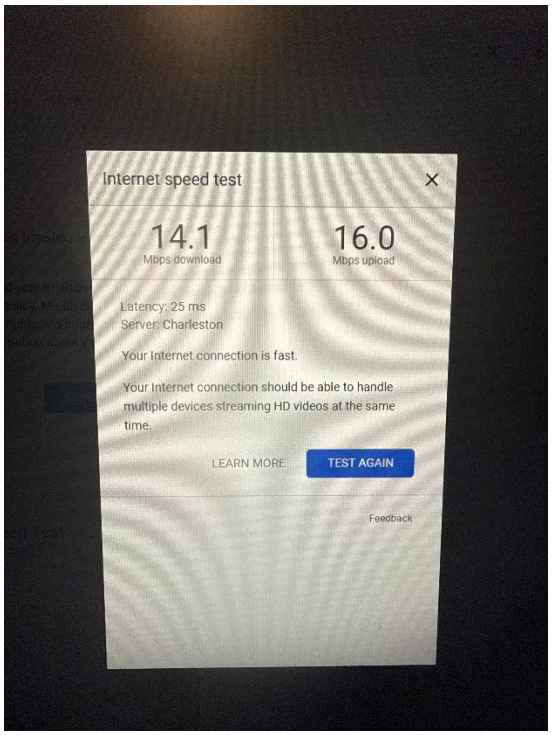


Figure 6. Wi-Fi speed performance from ESP32 host

Build Quality

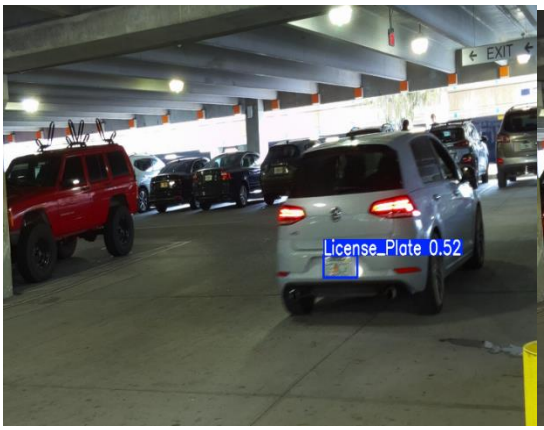
Robustness

Raspberry Pi

The RPi is resistant to bugs in optimal operation, meaning that when it is appropriately supplied with the power it requires (5V/3A minimum), it does not experience any crashes due to power insufficiency. There have been a few bugs observed depending on the distance between the computer that is SSH'ed into the RPi and the RPi itself. This is not considered a problem as the RPi will be flashed and function completely independent of the host computer. Additionally, there have been no problems connecting to the RPi while operating on the UF VPN network, thus, there is no real problem with the RPi operation.

The code has also not presented any issues in function and frankly has never experienced any crashing. Additionally, its operating is not dependent on lighting of its environment and the speed of vehicle. As discussed above, the RPi was tested with vehicles at different operational speeds of 5, 9, and 15 MPH. All instances these tests resulted in the RPi detecting the test vehicles license plate making it robust to many edge cases. All garages tested in also provided more than ample lighting for the RPi to appropriately view the vehicle. The only consideration is the shutter speed of the RPi camera. While functional and the model demonstrates high efficacy of detecting license plates despite the blur, it is clear the image quality worsens as the speed of vehicle increases. See Figure 7 above versus Figure 8 for comparison. Figure 8 shows the vehicle at 15 MPH while Figure 7 shows the vehicle slowing down into the turn, approximately 7 MPH. Even the human eye cannot make out the license plate detected which could pose potential issues for the second LPR model hosted on AWS. Additionally, in Figure 7, we can observe that the RPi works a bit slower than anticipated as the model is unable to detect a license plate even though there is clearly one in frame. A frame was still detected on this testing iteration, but it was detected in a similar position as Figure 6. Considering this, we can conclude that the angle of operation for the RPi module is paramount. It is more than capable of detecting cars at operational speeds in parking facilities, however, the vector of capture is critical to ensure that camera has the best field of view and angle to see the plate.

Additionally, the raspberry pi is undergoing testing to improve the captured image quality. This is to reduce the "grainy" look of the image so that when the image is provided to a stronger model, that model has a realistic change of inferencing the image and extracting the license plate.



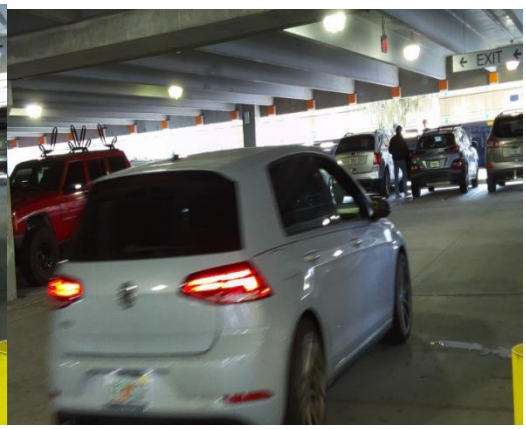


Figure 7. Car slowing down into turn from 15 MPH

Figure 8. Failure to detect plate until next frame

ESP32

The network connection to the ESP32 access point has proven to be stable through various tests. We have connected the ESP32 to multiple different networks, as well as multiple devices, and the connection has remained stable throughout all tests. The ESP32 has undergone testing to ensure that only the Raspberry Pis can connect to the device, as well as ensuring that the connection remains stable during these tests. This ensures that our Raspberry Pis will continually be connected to the device without any user input required. See figure 6 above to confirm.

Mobile App

User testing the app has led to no complete crashes of the app, as the Flutter framework is designed to allow the exceptions to occur without conflicting with other uninvolved parts of the code. Hence, failures in minor parts of the app's operation tend to go unnoticed unless the logs are viewed.

As of the Release Candidate, the app had been reformatted to improve its robustness. Functions were reformatted to match changes to the backend database's behavior and data output format. Additionally, when storing longitude and latitude from the database into a ParkingLocation object, the values from the database are verified to be cast into a double datatype, otherwise the functions would throw an exception and ParkingLocation objects would not update from the database. Deliberate behavior changes were made as well; the app now adds new lots from the database instead of only updating the pre-existing set that is hard-coded into the codebase, as the prior method would make it vulnerable to exceptions due to non-existing lots in the codebase.

Pi Configuration Script (Last Updated: Beta Build)

Executing the script fully from a fresh Raspberry Pi, within range of an eduroam access point for the University of Florida, failure should not occur, especially since all branches of execution were tested for expected behavior and issues encountered were fixed. There is no expectation that if the prerequisites are not met (e.g. completely fresh Raspberry Pi, access to eduroam network, the user has a ONE.UF account, etc.) that the script will work as intended.

Consistency

Raspberry Pi

The RPi has not demonstrated any signs of inconsistency. In all operational fixtures it has been able to function flawlessly, capture images, inference the images, and decode their values. In addition to this, the RPi has not demonstrated any issues in detecting cars as they drive by, in all tests and outings the RPi has been able to take pictures fast enough and the model has been able to detect license plates in all tests.

Mobile App

App behavior is consistent according to deliberate decisions made on behalf of us, the developers. For example, inconsistency for the user may arise due to default filter settings causing some lots to disappear from the map depending on the time of day, as that was how the filter was intended to behave, or when test lots are added to the app through the backend database. The app now waits for data to be read from the database before loading the map view to ensure its data is consistent with that in the database, as well as to ensure loading time is predictable and consistent.

Pi Configuration Script (Last Updated: Beta Build)

The Raspberry Pi configuration script ensures that all Raspberry Pi devices needed for the system are configured the same way with the necessary dependencies to run the software we develop to run the car detection algorithm.

Aesthetic Rigor

Raspberry Pi Housing

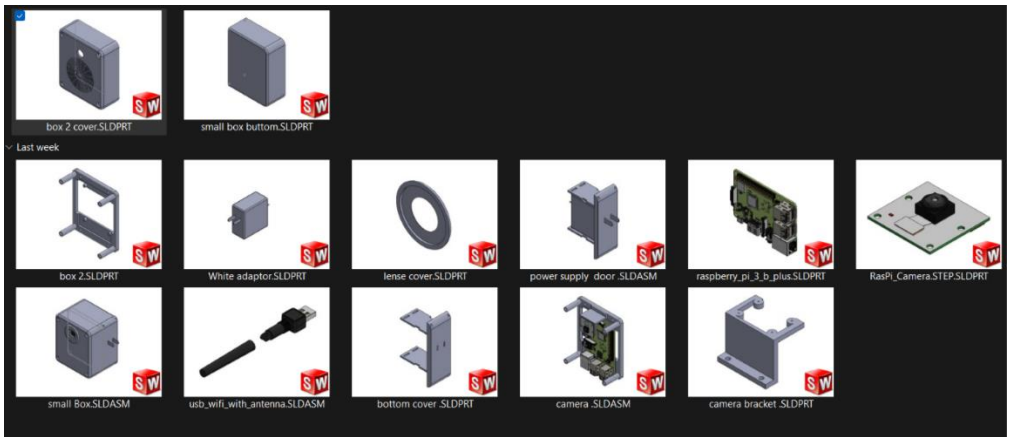
3D printing a custom enclosure for the Raspberry Pi is essential for protecting the hardware and ensuring a stable setup for photography and mounting applications. A well-designed case not only shields the Raspberry Pi from dust, impact, and environmental factors but also provides a secure way to mount 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.

Though many of the components have not been tested just yet, a few more 3D prints are currently in development to ensure proper ventilation and allow for future Wi-Fi capabilities, helping to refine the overall design and functionality.

As the project progressed, a few design changes were necessary to address practical challenges that emerged during assembly. Some of the internal wiring didn't fit as originally intended, prompting a few new cuts and adjustments to the enclosure layout to accommodate cable routing more effectively without compromising structural integrity. Additionally, due to growing concerns about temperature buildup inside the compact case, a dedicated slot for a small fan is being added to improve airflow and overall ventilation. This modification will help maintain optimal operating conditions for the Raspberry Pi and camera module, especially during extended use. Furthermore, a redesigned camera holder is in development to address inefficiencies in the original mount, aiming to provide better alignment, easier adjustments, and a more stable hold for high-quality image capture.

At one point, I considered designing an entirely new enclosure from scratch to address the various updates and challenges that had come up. However, as I started prototyping the new version, I realized that many of the changes I planned actually aligned well with the existing model. Rather than overhauling the entire design, it became clear that refining and building on the original enclosure was the more efficient route. This experience reinforced the idea that if something isn't broken, there's no need to fix it—sometimes small, thoughtful adjustments can be more effective than starting over completely.


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