Parking Availability System

Beta Build Report

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

Usability

Interface

Pi Configuration Script

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 an 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.

Mobile App (Alpha Build)

Normal 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 in a 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, cloud PostgreSQL database.

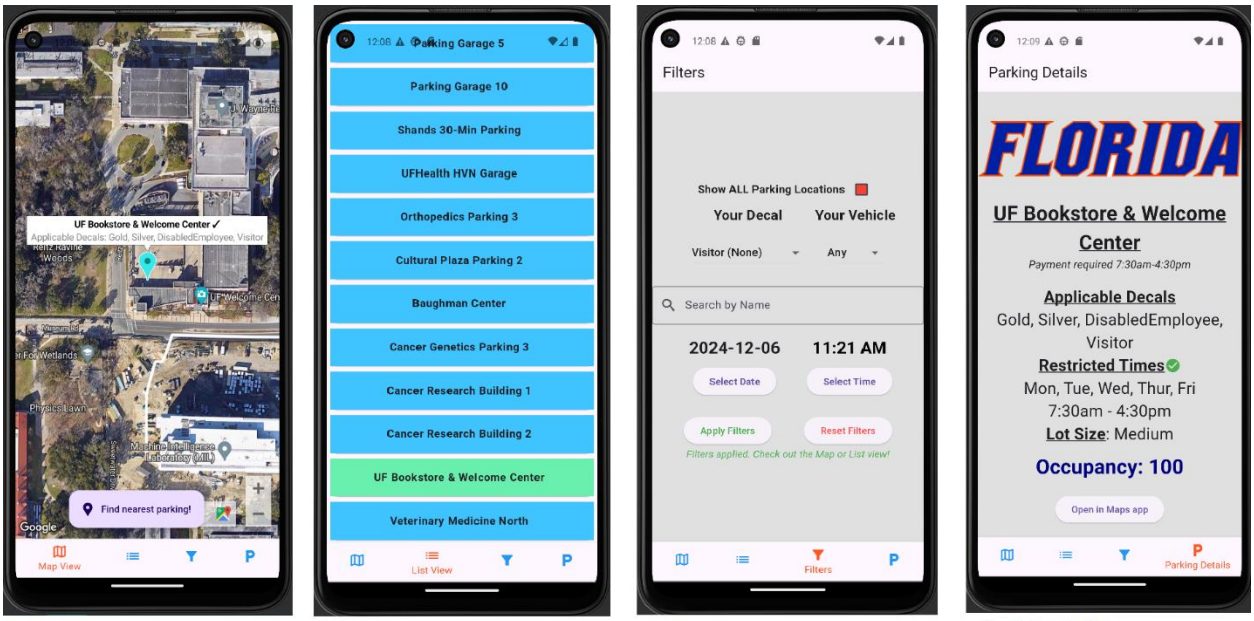


Figure 1. Parking App Frames

Navigation

Pi Configuration Script

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 should reconsider their aspirations if they are incapable of reading or typing. The user flow diagram in Figure 2 demonstrates the flow of script execution from the user's point of view.

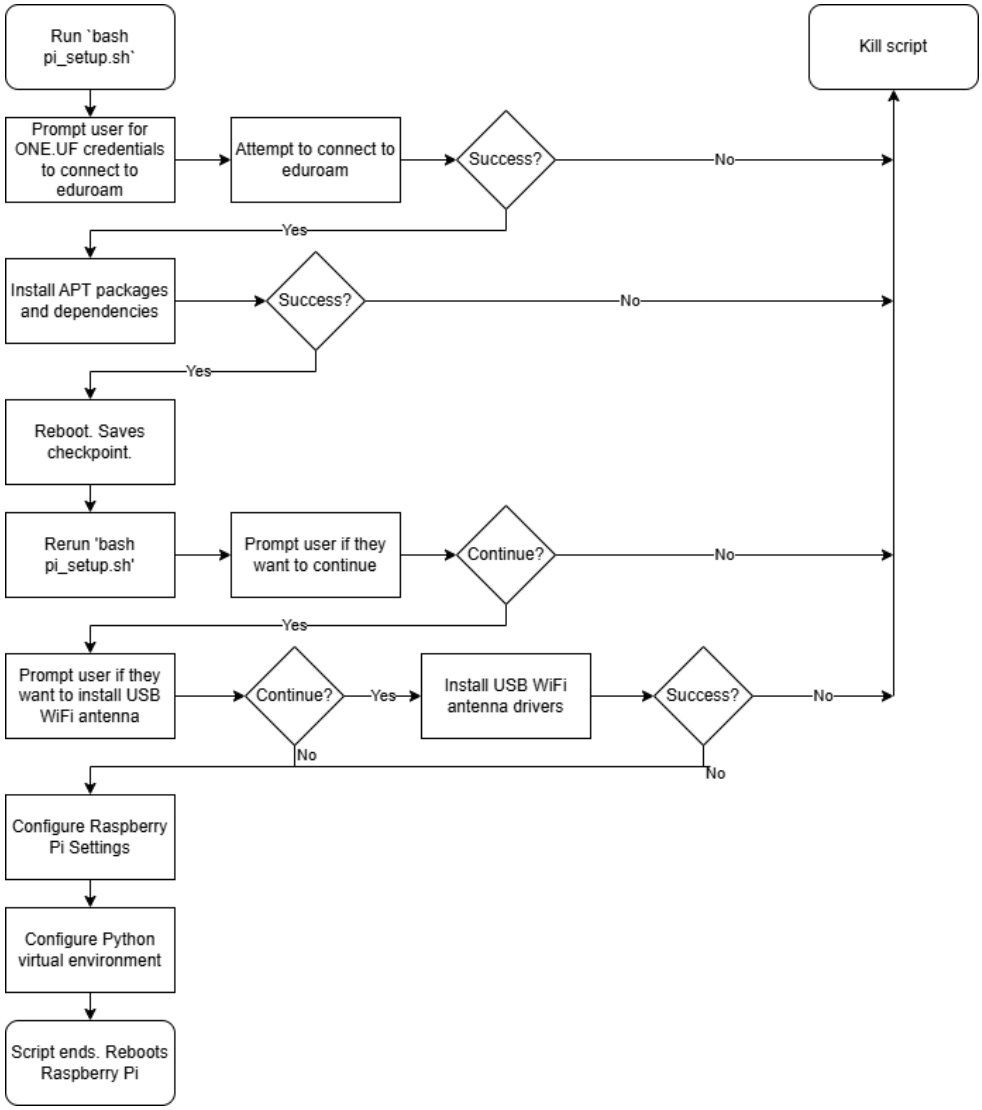


Figure 2. Pi Configuration Script User Flow Diagram

Mobile App (Alpha Build)

The mobile app currently implements all the features necessary to be able to view real-time parking facility updates by reading from 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.

Perception

Pi Configuration Script

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.

Mobile App (Alpha Build)

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.

Responsiveness

Pi Configuration Script

The configuration script notifies users with a prompt when they are needed 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.

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. If a license plate was determined to be detected, the database is updated as a car has either entered or left the parking facility. 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 3 and 4. Respectively, Figure 3 shows the first frame that includes a test car in a garage. Figure 4 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.

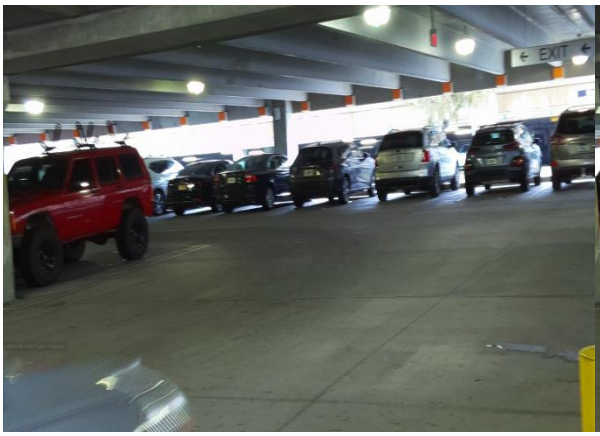


Figure 3. Bottom left side shows the test vehicle entering the frame

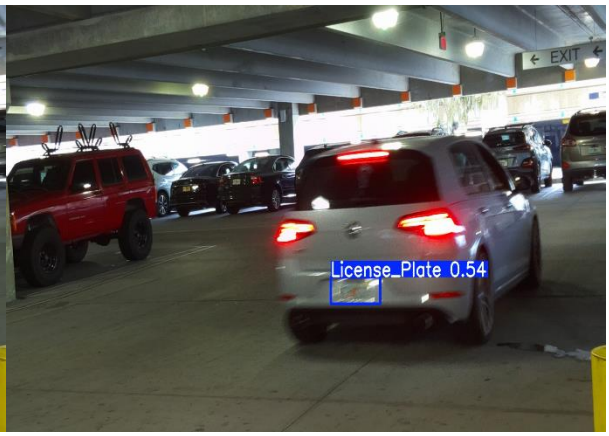


Figure 4. Shows the next frame captured by the RPi of the test vehicle moving 15 MPH past the RPi

Mobile App (Alpha Build)

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 for a specific location when that location is selected. Under conditions where the table could not be connected to, parking location information is also hard-coded into objects in the codebase, which allows users to always have access to some form of parking information, even if it is not up-to-date.

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 the RPi was able to detect 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 3 above versus Figure 4 for comparison. Figure 3 shows the vehicle at 15 MPH while Figure 4 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 4, 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 4 or 5.

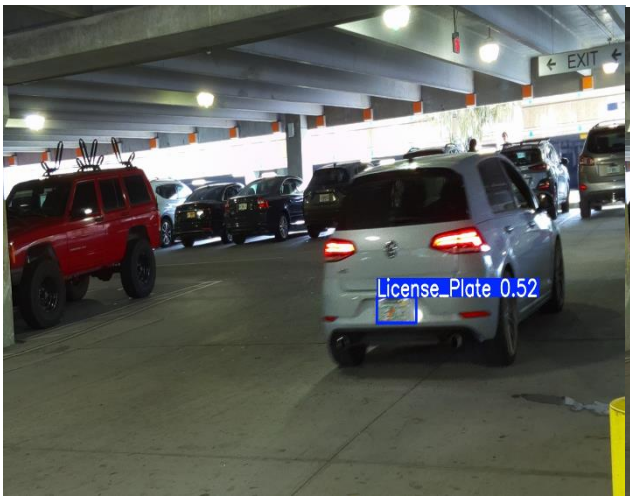


Figure 5. Car slowing down into turn from 15 MPH

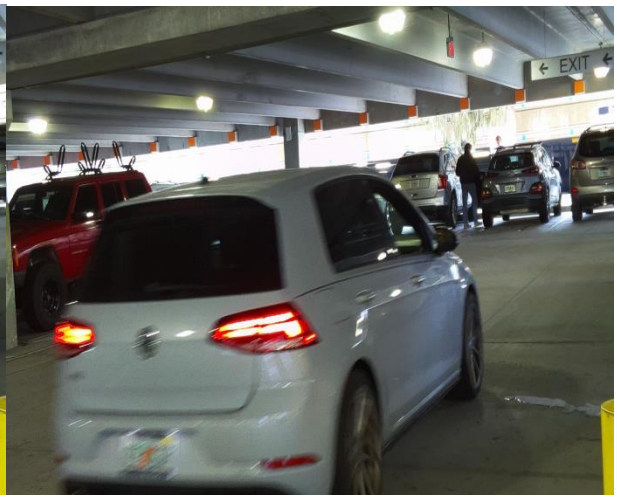


Figure 6. Failure to detect plate until next frame

Pi Configuration Script

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.

ESP32

The network connection to the ESP32 access point has proven to be stable through various tests. The ESP32 has undergone testing to ensure that only the Raspberry Pis can connect to the device, as well as ensuring

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.

Pi Configuration Script

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.

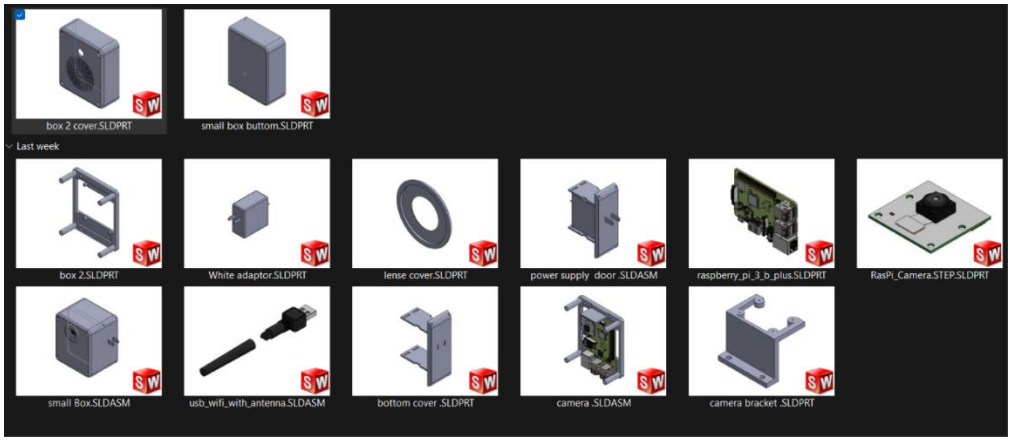
ESP32

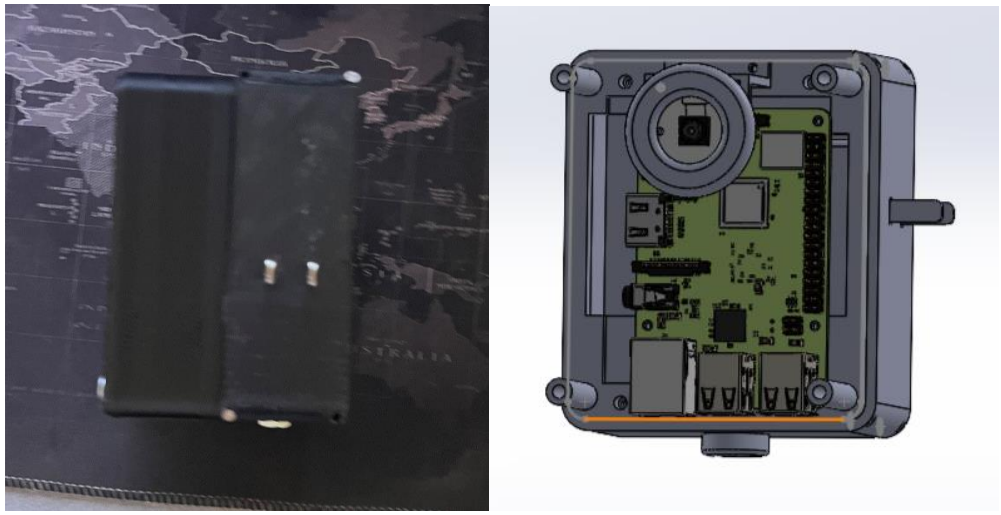
Aesthetic Rigor

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 WiFi capabilities, helping to refine the overall design and functionality.


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



Features

External Interface

Raspberry Pi

The RPi is configured to a camera to continuously capture images of its environment. These pictures are then fed into the object detection model to determine whether a license plate is present in one of the images. If it is, the RPi queries a PostgreSQL database to retrieve the parking facility occupancy count. Upon retrieval, the count is incremented or decremented depending on if the RPi is facing the exit or entrance of the parking facility. The license plate values are then recorded and added to or subtracted from the database. The update method functionality can be seen in Figure 8, where the RPi is able to query a parking facility, like UF's Parking Garage 14. From this, there is an occupancy integer attribute that can be adjusted accordingly. Once this is all done, the cycle repeats all over again. See the video for further details and code review.

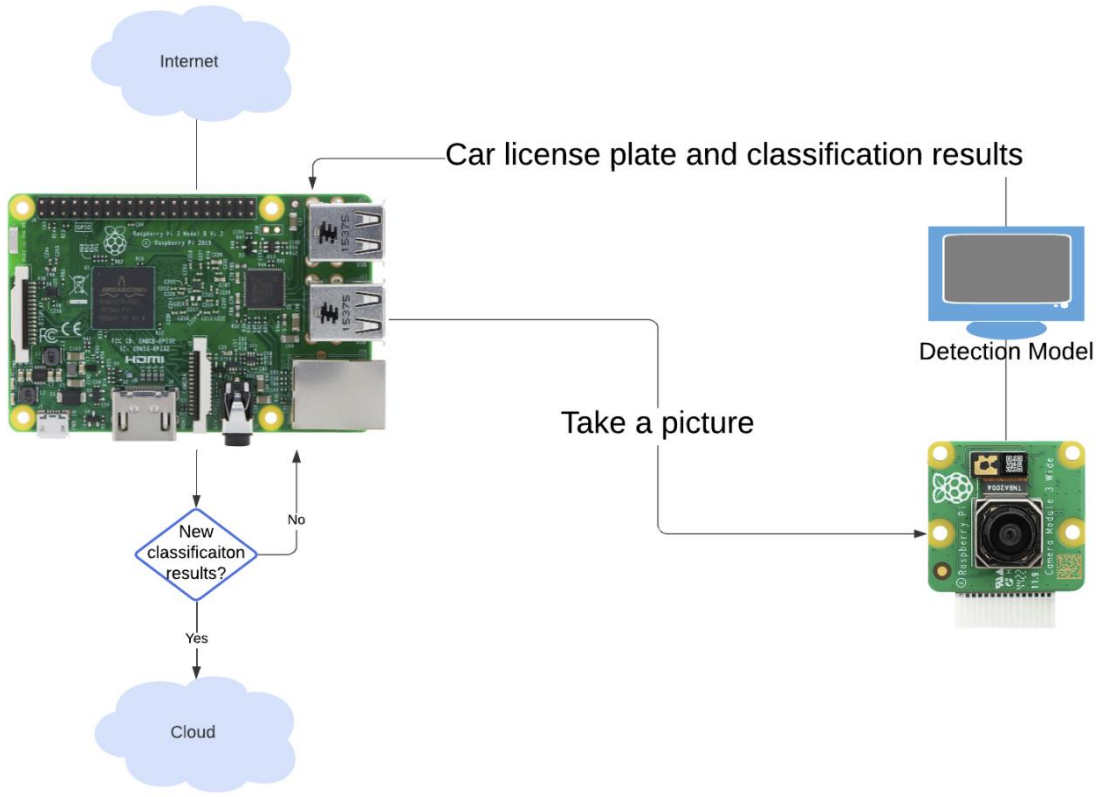


Figure 7. RPi Flow Diagram

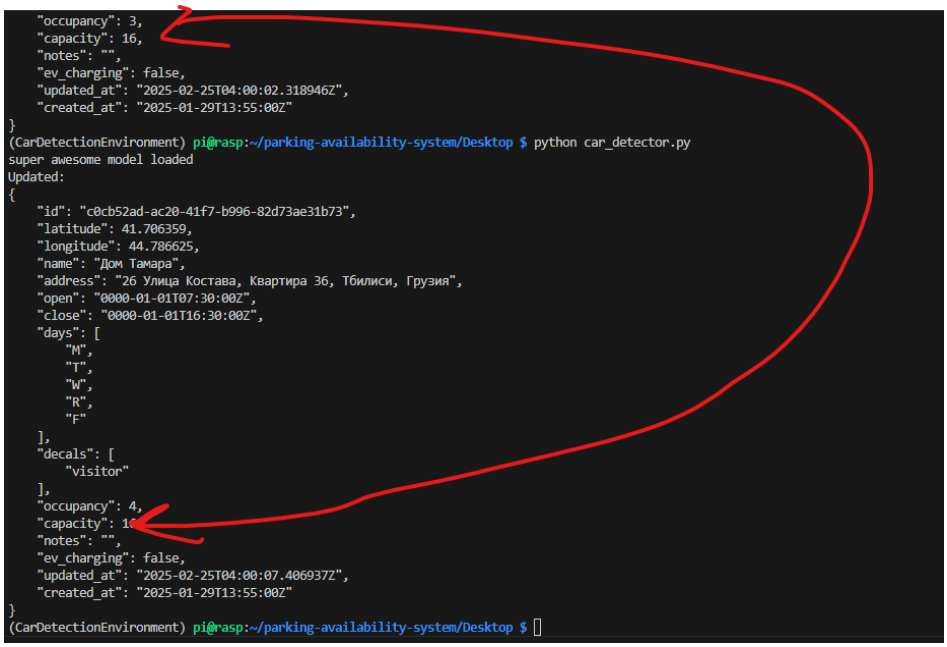


Figure 8. RPi querying and updating the occupancy values of a lot upon detecting a car

The external interface relies on the camera taking pictures fast enough to constantly provide the model with an accurate and timely representation of the environment. When a vehicle enters the photo frame, the model is fed the images and observes the frames provided by the photo feed from the camera. Our group makes the assumption that all motor vehicles will have a license plate, if an entity does not have a license plate, then it is not a motor vehicle.

Mobile App (Alpha Build)

The app communicates with the PostgreSQL database API endpoints via HTTP requests to perform CRUD operations on the data tables. Despite wrapper functions being developed for all supported operations on the LOT table, realistically the app only calls two of these functions —`getAllLots` and `getLot` — in order to retrieve the most recent lot data.

Persistent State

Raspberry Pi

While the RPi's caching mechanism is premature, it is able to continue to function completely without Wi-Fi. However, it will only store the latest result of the latest frame captured, meaning that if it were to lose Wi-Fi it would be unable to communicate with the database. Once the connection is restored, it would not be able to communicate the results of its finding during the downtime, rather, only the latest result acquired. This can be elegantly

resolved by writing the results during a downtime to a text file and then reading through the text file once the connection is restored and updating the database accordingly.

ESP32

The Wi-Fi connection in the Reitz parking garage does not extend to the entrance of the garage. However, there is a corner of the garage which has a Wi-Fi connection. Therefore, we are utilizing an ESP32 as an access point. This has been done by using the Wi-Fi library built into the ESP32. The ESP32 has been configured with an SSID and a password, as well as a max connection which defines the maximum simultaneous connected clients to the access point. The ESP32 temporarily stores data packets temporarily and then sends them out to the clients. The Raspberry Pi can connect to this network rather than attempting to connect to the UF network directly.

AWS Backend (Alpha Build)

The AWS backend interacts with the other components of the system using HTTP requests from other components that interact with the RESTful API implemented in the EC2 instance. The API implements CRUD operations that allow access to the relational databases storing car and lot information.

Mobile App (Alpha Build)

When the mobile app is first launched, it calls the `getAllLots` function to retrieve the most up-to-date parking location information from the database. However, in the case that this call is unsuccessful or times out, to prevent the user from being unable to access any lot data, ParkingLocation objects have hard-coded data in the app**.**

Internal State

Raspberry Pi

The RPi data processing is fully implemented. See Figure 9 below for a flow chart. So long as the device is connected to an ample power source, it will be able to continuously and without fail take images of its environment, inference them, and update the database.

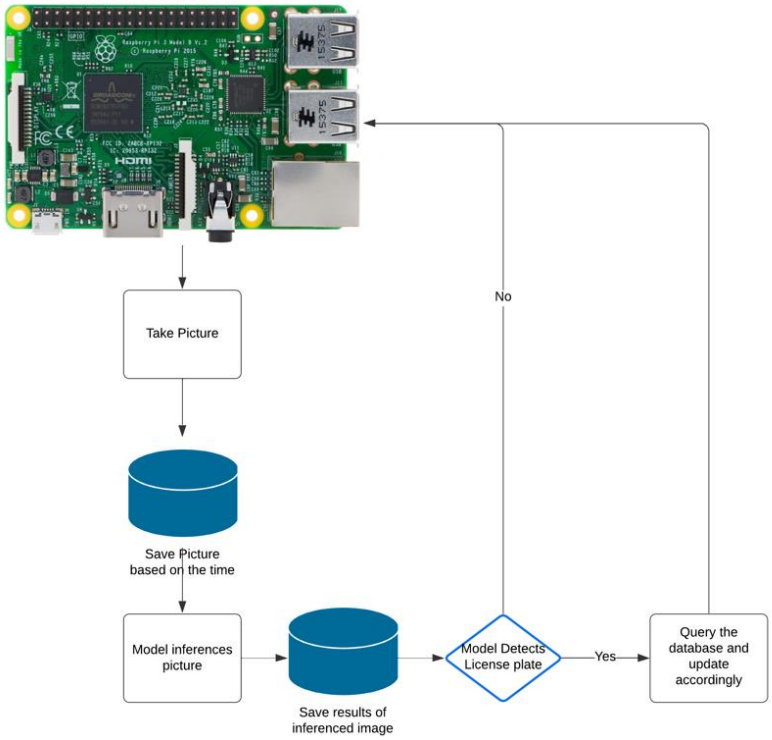


Figure 9. RPi Internal System Flow

Object Detection and License Plate Recognition (Alpha Build)

Object detection occurs both within the RPi and within the cloud. Object detection is implemented with Ultralytics's YOLOv11 nano model on the Pi to detect license plates within a scene and send frames with a detected license plate to the Optical Character Recognition (OCR) pipeline in the cloud. The OCR pipeline runs another YOLO model to verify the existence of the car and its license plate within the frame, sends that frame to be processed with OpenAI's GPT-4 to interpret the license plate, and saves the car information to the database using the Save Car API endpoint.

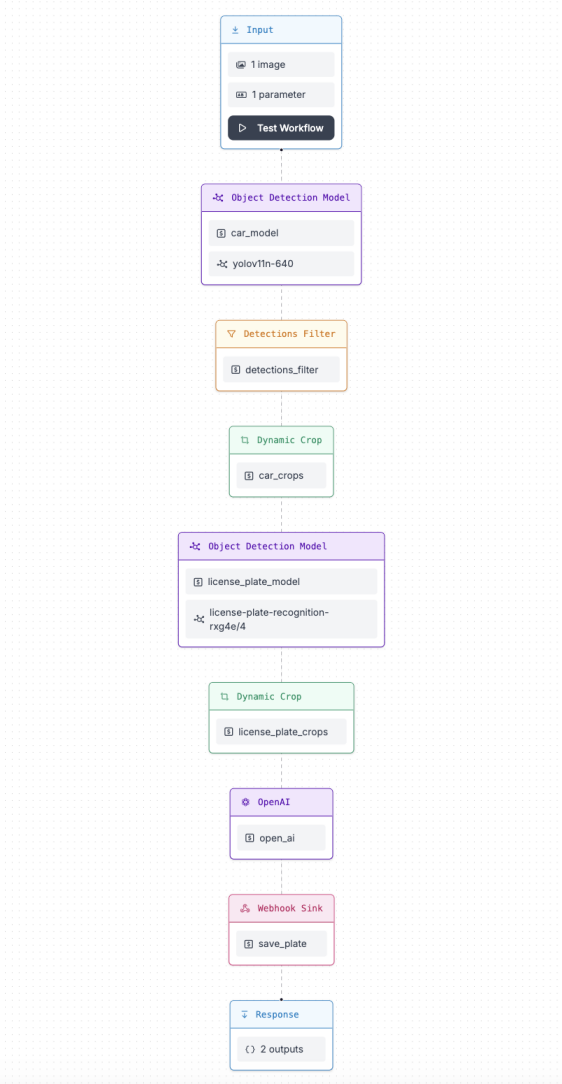


Figure 10. OCR Pipeline

AWS Backend (Alpha Build)

The cloud backend is hosted on AWS and implements two important components of the Parking Availability System, an object detection and license plate recognition model as well as the relational databases used to access lot and car information. As seen in Figures 11 and 12, a PostgreSQL database is hosted using AWS Relational Database Service, implementing two tables — CARS and LOTS — which tracks information, like license plate and color, for the cars parked in specific lots on campus detected by the system and information related to parking facilities on campus, respectively.

```
burrow=# TABLE cars;
                carid                 |  plate   |   color    |                lotid
--------------------------------------+----------+------------+--------------------------------------
 904f7f20-28f9-495e-afaa-f4d8bda1fd99 | JLF-1304 |            | f24dacb0-9513-45d9-90f8-b07e5dee5271
 1d8acb5e-1944-407f-ba24-0a3615feaf2e | T719257C |            | f24dacb0-9513-45d9-90f8-b07e5dee5271
 3a452865-6e30-46b0-83e9-c495556518c7 | 5K15     |            | f24dacb0-9513-45d9-90f8-b07e5dee5271
 5695e925-8527-4a22-964c-5adbe8aa8e44 | KNOJBL   |            | f24dacb0-9513-45d9-90f8-b07e5dee5271
 ae6f74b0-7647-4c89-b875-d40ef941fb68 | KN3 JBL  |            | f24dacb0-9513-45d9-90f8-b07e5dee5271
 6d62375a-be5c-4b05-ac75-4b31e4d8f9cc | "NMTE84" |            | f24dacb0-9513-45d9-90f8-b07e5dee5271
 24028284-8201-4c49-87b1-6b81d54b18c5 | TEST-CAR | TEST-COLOR | f24dacb0-9513-45d9-90f8-b07e5dee5271
 8e049944-94e5-4d87-8d70-6e8f84a2c768 | ADJ JBL  |            | f24dacb0-9513-45d9-90f8-b07e5dee5271
 07ba9230-09bc-48e1-933d-1f3156bdb901 | EVT R34  |            | f24dacb0-9513-45d9-90f8-b07e5dee5271
(9 rows)
```

Figure 11. PostgreSQL CARS Table

lotid [UUID]	latitude [NUMERIC]	longitude [NUMERIC]	name [VARCHAR]	address [VARCHAR]	open [TIME]	close [TIME]	days [_VARCHAR]	decals [_VARCHAR]
0210feeb-cf1d-42a7-95(	29.647502	-82.359249	Fraternity Row 2		08:30:00	15:30:00	{M,T,W,R,F}	{parkAndRide,red1,disabl
0226350f-5dc0-433e-af	0.407479	0.345672	test name	5678 Elm St	12:35:22	12:35:22	{M,T}	{Red,Green}
02bf2c6a-1810-4fe2-90	29.645655	-82.337330	Parking Garage 8		08:30:00	15:30:00	{M,T,W,R,F}	{orange,red1,red3,disable
0405bdde-778a-4de7-9	29.639044	-82.346679	Parking Garage 2		07:30:00	17:30:00	{M,T,W,R,F}	{gold,silver,visitor,disable
060dc221-bd70-408a-af	29.649438	-82.339431	Tigert Hall/13th Street M		07:30:00	16:30:00	{M,T,W,R,F}	{motorcycleScooter}
06fa4bf0-3ecb-4ef7-a0l	29.645198	-82.348377	UF Bookstore & Welcome		07:30:00	16:30:00	{M,T,W,R,F}	{gold,silver,disabledEmpl
09549251-14f3-44ab-9:	29.645754	-82.352848	Flavet Field		08:30:00	15:30:00	{M,T,W,R,F}	{green}
1114af86-d158-4b14-b4	29.640779	-82.341599	Parking Garage 10		08:30:00	15:30:00	{M,T,W,R,F}	{visitor,motorcycleScoot
11945155-ac9e-45c7-86	29.647512	-82.342973	Inner Road Motorcycle		07:30:00	16:30:00	{M,T,W,R,F}	{motorcycleScooter}
1284e0bd-fe3f-4156-84	0.083723	0.871200	test name	1234 Main St	12:38:42	12:38:42	{M,T}	{Red,Green}
128ce783-ff24-4a47-be:	29.639699	-82.356552	IFAS Parking 3		08:30:00	15:30:00	{M,T,W,R,F}	{orange,motorcycleScoot
1429020c-9ba3-4662-a	29.646837	-82.350160	East Hall		08:30:00	15:30:00	{M,T,W,R,F}	{orange,motorcycleScoot
18a4bda1-b294-47a9-af	29.634154	-82.351701	Veterinary Medicine Wes		08:30:00	15:30:00	{M,T,W,R,F}	{parkAndRide,green}
18a9b8f3-7f9c-41bc-act	29.637907	-82.367623	Southwest Rec Center		08:30:00	15:30:00	{M,T,W,R,F}	{parkAndRide,disabledSt

Figure 12. PostgreSQL LOTS table

The information in the LOTS table is used to track real-time parking availability, being updated by the RPi hardware or read from by users' mobile applications. Both the CARS and LOTS tables are interacted with through a RESTful API written in Golang hosted on an AWS EC2 instance, implementing 10 REST endpoints to perform CRUD (Create, Read, Update, Delete) operations on both tables through HTTP requests from the other system components.

Mobile App (Alpha Build)

With permission from the original developer, Drew Gill, a UF Computer Science alum, the mobile application component and the real-time parking information feature were built on top of his codebase. The app is built using the Flutter framework using the Dart programming language, which allows applications to be developed using a single codebase. Google Maps API is used to display the map and location pins within the app. Parking location data is both hard coded into the code and read from the database, though information read from the database is prioritized. Upon starting the app, hard-coded data is overwritten with information from the

database. When a parking location is selected and the information page is visible, the app polls the database periodically, ensuring that the user has the most up-to-date information.

Effort

Effort for the project is documented in many forms, but it is difficult to aggregate and quantify in a single report. All effort is logged across the GitHub repository, the GitHub Kanban board, the Contribution Log Excel sheet, and the meeting check-in sheet. All of these resources can be accessed in the project's GitHub repository, according to the README: https://github.com/emeurrens/parking-availability-system.