Final Report – 5/2/2022

EE/CpE 4813 – ECE Capstone Design II

Spring 2023

Instructors: Johnathan Votion Ph.D, Professor August Allo.

Team name: OYSMS Digipark

Team members: Preston Ott, Younes Younes, Joshua Swanner, Joshua Moya, Luis Santillan

Description: The Parking Space Indicator is a project that will map a parking lot, develop a scanner to scan parking spaces and develop an app to display occupied/unoccupied parking spots in the parking lot.

Executive Summary

The OYSMS Digipark team's prototype scanner and app were developed with the goal of providing real-time occupancy status of a parking lot to users. The project was designed to alleviate the stress and frustration of finding a parking spot by allowing users to check the availability of parking spots before they arrive at the parking lot. The team incorporated a machine learning model to accurately detect the occupancy of parking spots, which was transmitted to a receiver and stored in a SQL database for easy retrieval by the app. The team also made adjustments to the hardware components, such as the camera and microcontroller, to ensure the reliability and ease of use of the final product.

Overall, the project was a success, with the final prototype demonstrating the team's ability to create an innovative solution to a common problem. The team's ability to adapt and innovate when faced with challenges was crucial in achieving the project goals.

Here are two images of the final the prototype:

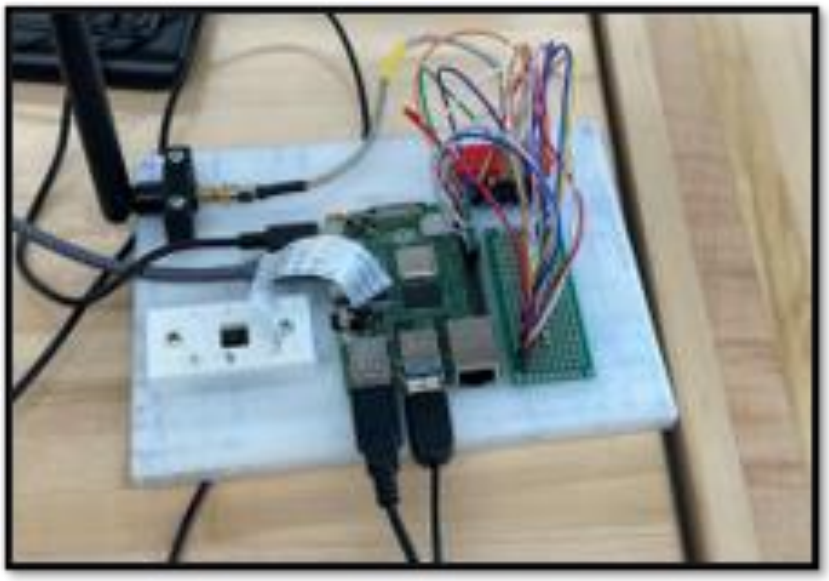


Image 1. Final Prototype

Image 2. Final Prototype working

Additionally, here is a video demonstration of the final prototype:

[Parking Project Demonstration-2023-OYSMS-Digi-Park]

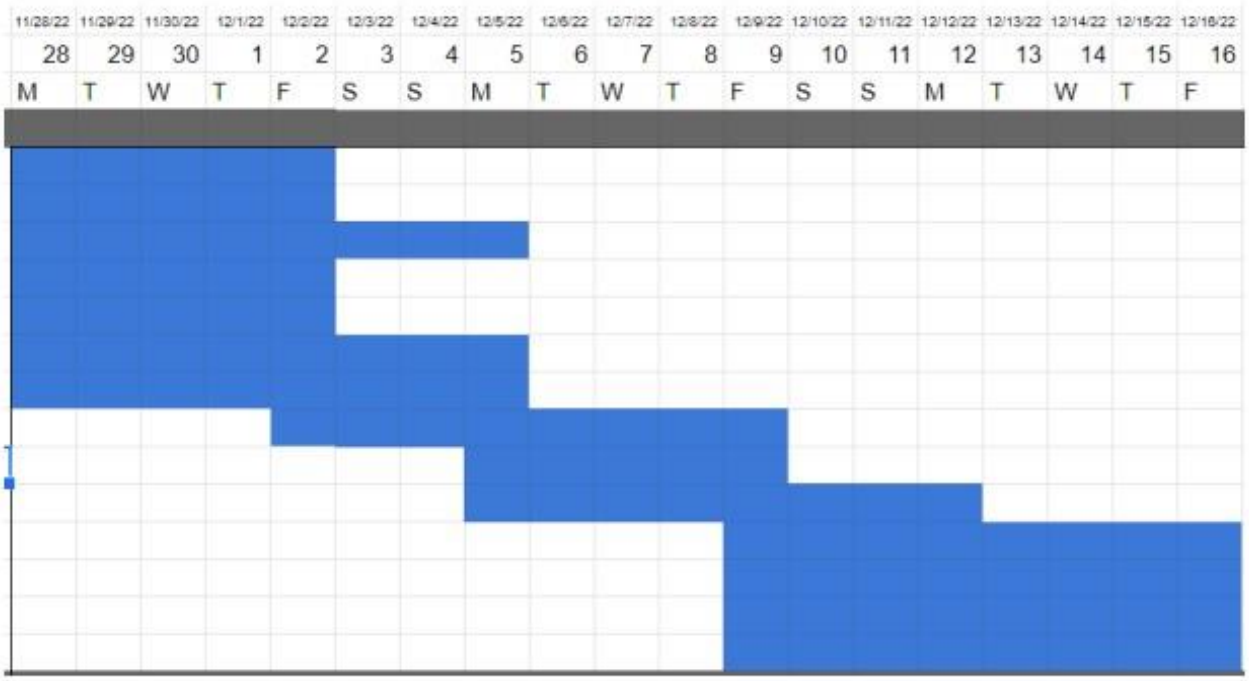
Video 1: Video of Final Prototype Working

1.0 Introduction

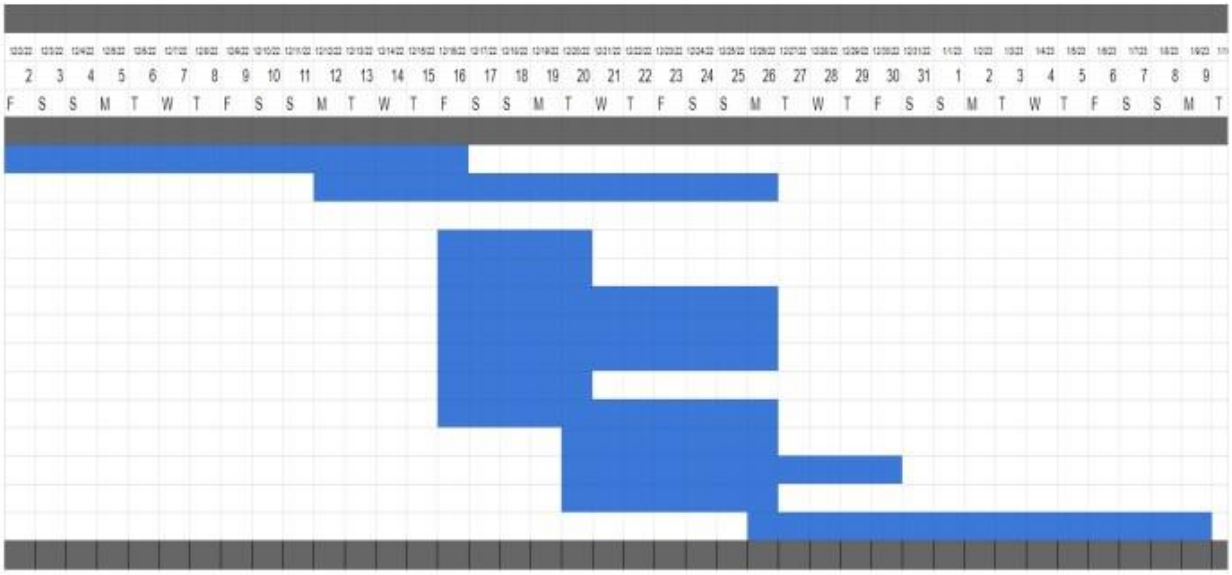
We are OYSMS Digipark. This project aims to set up a scanner in a parking lot which can also update an app that can be viewed on an electronic device. The project will first take a picture through the camera of the desired parking lot that the user is looking for. The camera will then send that photo to the machine learning model to process the Afterwards, a spot array will be created that contains 1's and 0's depicting the occupancy of each parking spot. The transmitter will then send the spot array to the receiver. Once received the receiver will update a SQL database that will be used to store the spot array. The user will type in the desired address in the app which will request a spot array through the server. Finally, the server sends the spot array to the user and the users app will display the parking spots that are occupied and unoccupied with vehicles.

1.1 Project Schedule

6	STAGE 1			
7	Choose Components	11/28/2022	4	12/2/2022
8	Choose Circuit Software	11/28/2022	4	12/2/2022
9	Determine modules/libraries for app	11/28/2022	7	12/5/2022
10	Choose Machine Learning Model	11/28/2022	4	12/2/2022
11	Choose 3d CAD software	11/28/2022	4	12/2/2022
12	List basic classes/functions/flowchart for Server Pr	11/28/2022	7	12/5/2022
13	Determine modules/libraries for microcontroller	11/28/2022	7	12/5/2022
14	Determine modules/libraries for ML Model	12/2/2022	7	12/9/2022
15	Build Circuit Schematic	12/5/2022	4	12/9/2022
16	List basic classes/functions/flowchart for app	12/5/2022	7	12/12/2022
17	Simulate and Test Circuit Schematic	12/9/2022	7	12/16/2022
18	List basic classes/functions/flowchart for mirco con	12/9/2022	7	12/16/2022
19	List basic classes/functions/flowchart for ML Model	12/9/2022	7	12/16/2022
20	Determine modules/libraries for Server Program	12/9/2022	7	12/16/2022

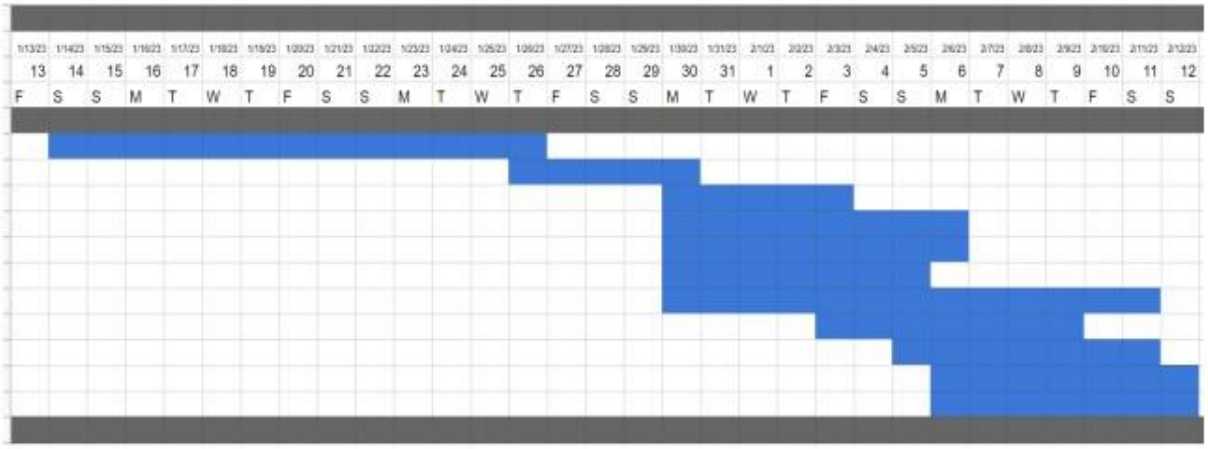


21	STAGE 2			
22	Order/Recieve parts	12/2/2022	14	12/16/2022
23	Write classes/functions for app	12/12/2022	14	12/26/2022
24	Write classes/functions for microcontroller	12/12/2002	10	12/22/2002
25	Assemble Power module	12/16/2022	4	12/20/2022
26	Assemble Scanner Module	12/16/2022	4	12/20/2022
27	Find/write test software for digital camera	12/16/2022	10	12/26/2022
28	Find/write test software for transmitter/reciever	12/16/2022	10	12/26/2022
29	Find/write test software for servo motor	12/16	10	12/26
30	Assemble Transmitter Module	12/16/2022	4	12/20/2022
31	Write classes/functions for ML Model	12/16/2022	10	12/26/2022
32	Test Power Module	12/20/2022	6	12/26/2022
33	Test Transmitter Module	12/20/2022	10	12/30/2022
34	Test Scanner Module	12/20/2022	6	12/26/2022
35	Find Machine Learning Model Test/Train data	12/26/2022	14	1/9/2023

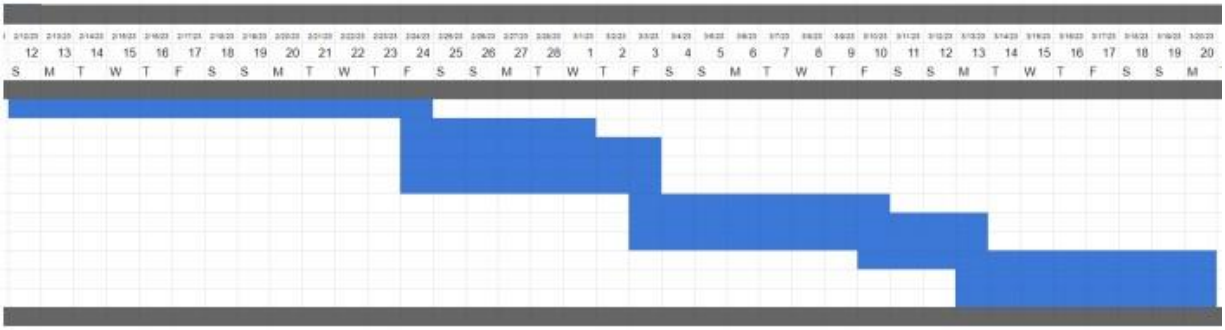


44	STAGE 3			
45	Assemble Modules into early prototype	12/30/2022	7	1/6/2023
46	Write basic program for micro-controller	12/30/2022	12	1/11/2023
47	Write basic program for server	12/30/2022	12	1/11/2023
48	Write basic program for app	12/30/2022	12	1/11/2023
49	Train Machine Learning Model / have working mod	1/9/2023	5	1/14/2023
50				

54	STAGE 4			
55	Test early prototype	1/14/2023	12	1/26/2023
56	List bugs,errors,issues in prototype	1/26/2023	4	1/30/2023
57	Isolate bugs and errors in circuit	1/30/2023	4	2/3/2023
58	Isolate bugs and errors in app	1/30/2023	7	2/6/2023
59	Isolate bugs and errors in server program	1/30/2023	7	2/6/2023
60	Isolate bugs and erros in microcontoller program	1/30/2023	6	2/5/2023
61	Improve Server/Mircocontroller program for testing	1/30/2023	12	2/11/2023
62	Debug errors in circuit	2/3/2023	6	2/9/2023
63	Debug error in microcontroller program	2/5/2023	6	2/11/2023
64	Debug errors in app	2/6/2023	6	2/12/2023
65	Debug errors in server program	2/6/2023	6	2/12/2023
66				

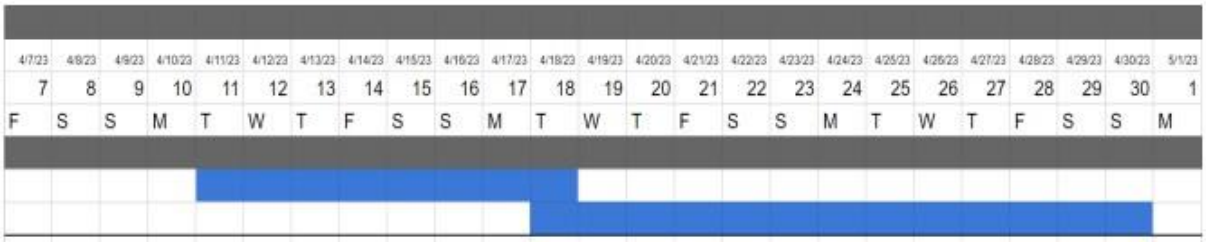


STAGE 5			
Test prototype in small parking lot	2/12/2023	12	2/24/2023
Finalize circuit into PCB or Perf board	2/24/2023	5	3/1/2023
Develop 3d printed case for circuit	2/24/2023	7	3/3/2023
List Functions/Classes/flowchart for parking naviga	2/24/2023	7	3/3/2023
List Functions/Classes/flowchart for parking statisti	2/24/2023	7	3/3/2023
Print/Assemble 3d printed Case	3/3/2023	7	3/10/2023
Add Navigation features to App Program	3/3/2023	10	3/13/2023
Add Statistics features to App Program	3/3/2023	10	3/13/2023
Test finalized circuit	3/10/2023	10	3/20/2023
Test/Debug Navigation features in app	3/13/2023	7	3/20/2023
Test/Debug Statistics features in app	3/13/2023	7	3/20/2023



STAGE 6			
Test Improved prototype in parking lot	3/20/2023	12	4/1/2023
Deploy Parking App on store	3/20/2023	7	3/27/2023
Debugging/Testing for Final Prototype App	4/1/2023	10	4/11/2023
Debugging/Testing for Final Prototype server progr	4/1/2023	10	4/11/2023
Debugging/Testing for Final Prototype Circuit	4/1/2023	10	4/11/2023

STAGE 7			
Final Test of Prototype	4/11/2023	7	4/18/2023
Preperation for Tech Symposium of Project	4/18/2023	12	4/30/2023



2.0 Need Being Addressed

A problem that has been an issue for some time now is the parking around campus. Finding a parking spot on campus can be tough because there are so many people trying to go to class, and with a limited number of spots it is hard to know if other parking lots have open parking spots. This will help many people find parking on campus with little to no struggles. All businesses and buildings that have parking lots will benefit from this solution because customers

avoid shops due to inadequate parking space. All people who use a motor vehicle will benefit from this solution through fuel and time savings.

3.0 Literature and Patent Search Results

Researching Parkam's AI-based deep machine learning software provided the team with a vision of how our product should function. Their patent for *a training system for automatically detecting parking space vacancy* providing inspiration towards training our own machine learning program. The ability to identify parking spots and their states accurately being the main inspiration. With Cleverciti's parking sensor providing the basis for the physical aspect of the OYSMS scanner. Although ours is made of a more cost-effective material, uses a single camera and customizable clamps for adjustable attachment to various surfaces.

4.0 Marketing Analysis and Market Strategy

The market potential of this product is vast and promising. In a full parking lot, customers will get frustrated and start to leave the premises, meaning the company will start to lose money. Companies could use this app to help reduce the amount of potential customers from going to a different store. With this product implemented into businesses parking lots, customers would want to come more often because of how convenient it is for them to find a parking space. This would potentially make the business grow and the product grow as well because then customers would start to look for businesses that are associated with the app to start shopping at.

5.0 Engineering Design Constraints

The main constraint of the application are the guidelines placed on by the app stores and the capabilities of the programming language being used to develop the app. For the device there are rules and standards given by the FCC when transmitting radio frequencies. Another constraint would be communicating the devices' information to the server running our application.

6.0 Product Requirements/Specifications

This section of the document lists specific requirements and specifications for OYSMS DigiPark. Requirements and specifications are divided into the following sections:

1. User requirements and specifications. These are requirements and specifications written from the point of view of end users, usually expressed in narrative form.
2. System requirements and specifications. These are detailed requirements and specifications describing the functions the system must be capable of doing.
3. Interface specifications. These are requirements and specifications about the user interface, which may be expressed as a list, as a narrative, or as images of screen mock-ups.

6.1 User Requirements and Specifications

6.1.1 User interface

The user interface will be available from a mobile app that the user will be able to use on their cellular device. They will be able to look at the parking spaces available and the parking lots that are available and the ones that aren't available.

6.1.2 Ergonomics

This product will be in the air above the reach of anyone, so that it will be able to scan the designated area. This will make sure it is not in anyone's way.

6.1.3 Training or skills required

Technical understanding and experience of operating a smart device is required for the app. To be able to use the app you just must know how to use your phone. Being able to navigate through the app without any trouble. You also need to be able to drive and/or have navigation skills to get to the designated spot.

6.2 System Requirements and Specifications

6.2.1 Physical Characteristics

- Raspberry pi 4
  - Weight: 46 g
  - Dimensions: 85.6mm x 56.5mm
- Transmitter
  - Weight: 15.4 g
  - Dimensions: 0.63x0.63x0.07 in
- Camera
  - Weight: 34 x 24 mm
  - Dimensions: 20 g

6.2.2 Material Requirements

- 3-D Printed case

6.2.3 Electrical Requirements

- Raspberry pi 4:
  - Power: 5V DC, 2.5A
  - Operating System: Python
  - Operating Temperature: 0-50 degrees C

6.2.4 Abilities

- Take an image and processing that image

6.2.5 Limitations

- Not enough range to cover every parking lot simultaneously
- Wi-Fi signal not strong enough

6.2.6 Equipment or materials required to use the product

- Smart Phone
- Mobile App installed

6.2.7 Equipment interface requirements

- SD memory storage reading capabilities

6.2.8 Handling and storage requirements

- Needs to be up high in the air

6.2.9 Cleaning and Sterilization

- Cleaning the camera is going to be essential for us to get a clear picture of the parking lot

6.2.10 Product maintenance and serviceability

- Cleaning the camera to get a clearer picture
- Fixing the bugs of the app as we go along
- Wi-Fi needs to be available

6.2.11 Operating parameters

- OV5647
    - Power Supply: 3.3V - 5V
    - Pixel Size: 1.4µm x 1.4µm
    - Temperature: -10°C～+55°C
- Raspberry pi 4
    - Power Supply: 5V DC, 2.5A
    - Temperature: 0-50 degrees C
- RFM95
    - Power Supply: 3.3V
    - Temperature: -55°C～+115°C

6.2.12 Repeatability and reproducibility

- The project is going to be able to refresh and be able to repeat each process every 15 seconds. If the camera is going to be clean, then there should be no problems.

6.2.13 Reliability

- Depending on the weather conditions and other conditions, this product will be reliable on any day. Up to the user to refresh the app to see availability.

6.2.14 Mechanical safety features

- 3-D printed case

6.2.15 Electrical safety features

- Raspberry Pi 4
- Wires
- Transmitter
- Camera
- PCB

6.3 Interface Requirements and Specifications

Access and translation between the digital camera and transmitter should be monitored by our microcontroller. For the sake of consistent image upload to a database server which will translate said information into a readable form for our users. From this, our users will interact with the given parking information via an application that will take user input and guide them to their desired location. The device will be detached from the elevated location for maintenance if physical repairs are necessary and updates to the server can be done at a separate location.

7.0 Engineering Codes and Standards

Below are the IEEE standards for transmitting data over radio frequencies and other wireless forms of communication. As our devices require wireless connectivity to communicate with the server and application, these standards and regulations are required:

IEEE Standard for Radio-Frequency Energy and Current-Flow Symbols: Symbols to inform people about the presence of potentially hazardous levels of radiofrequency energy or the presence of contact current hazards in the frequency range of 3 kHz to 300 GHz are specified in this standard. Guidance is given about how these symbols should be used on warning signs and labels.

IEEE Recommended Practice for Radio Frequency Safety Programs, 3 kHz to 300 GHz: Elements of a radio frequency (RF) exposure safety program that provide reasonable and adequate guidance for preventing exposures in excess of recognized limits to electromagnetic fields from RF sources that operate in the frequency range of 3 kHz to 300 GHz are described in IEEE Std C95.7™-2014. The means for accomplishing this are classifying exposure locations into one of four categories based on the potential hazard, as defined by exposure limits, and specifying appropriate controls for each category. Such controls include engineering and administrative controls as well as the use of personal protective equipment, placement of appropriate RF safety signage, designation of restricted access areas, the use of personal RF monitors, and RF safety awareness training. These recommendations are not intended to apply to the purposeful exposure of patients by or under the direction of medical practitioners but can be

used in the development of safety programs for medical staff and other persons working with or incidentally exposed to RF fields, and for those wearing implanted or external medical electronic devices. Although designed to complement IEEE Std C95.1™, this recommended practice may also be used for the development of programs to ensure conformance with other guidelines, standards, or regulations for controlling human exposure to electromagnetic energy.

IEEE Guide for EMF Exposure Assessment of Internet of Things (IoT) Technologies and Devices: In the wireless communication field, 5G and Internet of Things (IoT) solutions are the main emerging technologies and future wireless communication will rely on them. A methodology for classifying IoT devices based on radio frequency (RF) exposure characteristics is provided. Classification is based on frequency, bandwidth, radiated power, and typical installation configuration. Links between device class and available measurement/computational standards are provided.

8.0 Design Concepts

When discussing how the go about this project our team settled on three concepts to compare and discuss. Below is a Pugh Matrix with weighting of importance displayed within the second column alongside the determining criteria. From which the scanner module design was selected with a total score of 785.

Criteria	Weight	Drone	Sensor	Scanner Module
Cost:	20	5	4	8
Efficiency:	30	6	8	8
Information Quality:	10	8	10	8
Transmit speed:	10	8	8	8
Temperature ratings:	15	9	8	8
Scan Rate:	15	6	8	7
Score =	100	539	740	785

8.1 Drone Concept

A concept utilizing a drone to periodically survey an area and collect parking lot information. However, as power constraints are quite limiting, scan rates are not as frequent as desired and less efficiency in larger areas, this idea was inevitably the least favored amongst the three options.

8.2 Sensor Concept

The light sensor method would utilize a scanner device that would project a signal upon a sensor on the ground in a parking spot. The sensor would receive and reflect the

signal back unless something such as a vehicle blocks the signal, which would notify the main device that the spot is occupied. While this method is effective for condensed areas, it's not as effective in larger lots where several receiver nodes would be required. Also, the signal could be blocked by something not actually occupying the space such as a person blocking the sensor.

8.3 Scanner Module Concept

Using a scanner module and a machine learning algorithm, this device could observe a lot and detect changes within real time like the sensor concept. Trading some detection speed and a bit of information quality for more accuracy and cost efficiency in larger areas, the scanner module is the more desirable option by comparison. This device could even identify the difference between a vehicle and a person or tree branch that's simply obstructing view with more advanced Artificial Intelligence.

9.0 High Level Block Diagram

The parking lot scanner block consists of a scanner, a control system, a transmitter, and a power supply. Firstly, the power supply subsystem will provide voltage to the other subsystems in the parking lot scanner. Between 3.3 and 5 volts of power will be input to the power supply, note that many circuits operate. The first major subsystem is the scanner. The scanner will be given an input of a physical car and/or parking spot. The scanner may be given an individual car/parking spot, or it may be given several cars/parking spots dependent on the type of scanner being used (ultrasonic, photodiode, camera, etc.). The control system will act as a brain for the whole parking lot scanner. The main purpose of the control system will be to take the data from the scanner and interpret it. It may then simplify the data and output the data to the transmitter along with instructions. The instructions sent to the transmitter may consist of how often to transmit, what data to transmit, what frequency to transmit at, when to turn off, etc. The main input of the transmitter will be the data of the scanner (from the control system) and will output the same data over the air. The main purpose of the transmitter is to communicate and update the server with the important information sent from the control system. The parking space app will be decomposed into 2 main parts: the client app and the server program. We'll require a server for our project which can be any desktop computer. The server will take in 2 types of data, and these are client requests and scanner data. The second part of the parking space app will be the client app. This app will run on the client's device, which will most likely be a smartphone.

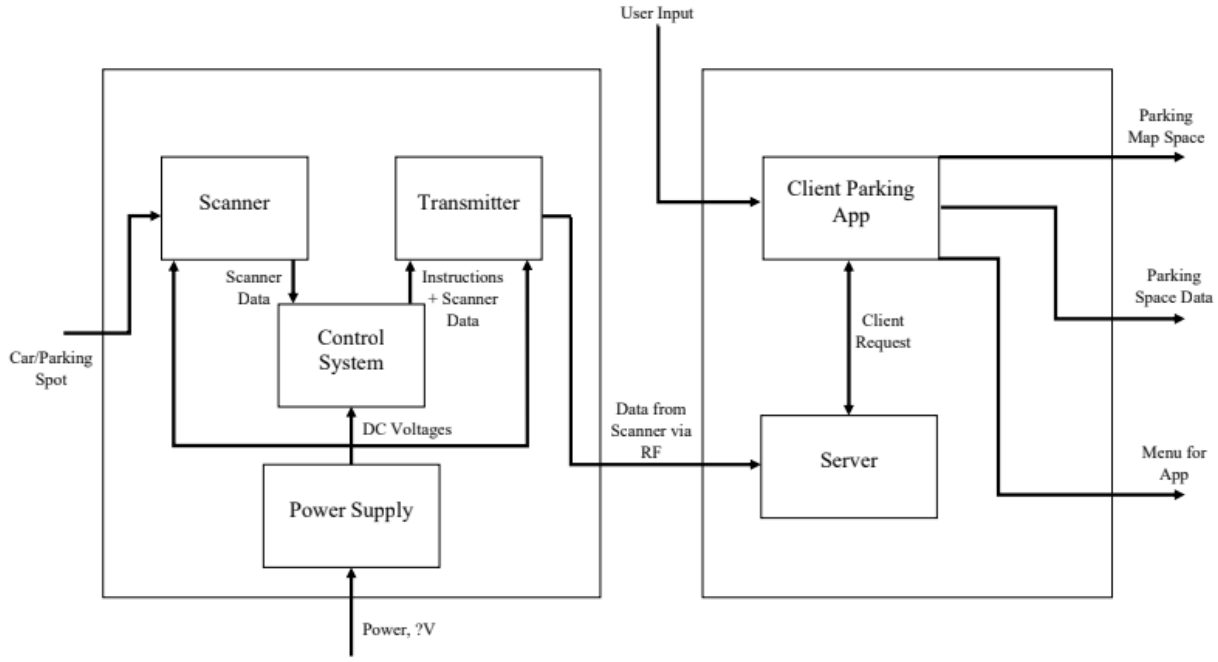


Figure 1: High Level Block Diagram

10.0 Major (Critical) Components

10.1 Camera Module

Criteria	Weight	OV5647 OmniVision	MT9P11 ONSemi	S5K4ECGX Samsung
Cost:	15	7	5	8
Output Formats:	17	9	6	7
Image Quality:	30	10	10	10
Interfacing Capabilities:	19	9	7	7
Temperature ratings:	10	8	9	8
Pixel Size:	9	9	9	9
Score =	100	989	781	833

Functional requirements for the digital camera Pugh matrix are cost, output formatting options, image quality, interfacing capabilities, temperature ratings, and size of pixels. The three digital cameras being analyzed are the OV5647, MT9P11, and the S5K4ECGX. The cost was weighed as the middle of the pack (15) because most camera modules have around the same price for similar specifications. Output formatting was rated similarly (17) because of how

common the most important format options such as RAW, YCbCr422, and JPEG are across various models. Image quality is a crucial factor in determining our device (30) as the clearer the images provided, the smoother our program can operate.

Temperature and pixel size are the least consequential factors (10 and 9 respectively) because after observing various models most are more than capable of operating in extreme temperatures. With pixel size in a comparable situation as most camera modules have a 1.4um x 1.4um pixel ratio amongst the similar specifications. The OV5647, MT9P11, and the S5K4ECGX got scores of 989, 781, and 833, respectively. Taking all the functional requirements into account the OV5647 surpasses the others in terms of quality.

10.2 Control system

Criteria	Weight	OV5647 OmniVision	MT9P11 ONSemi	S5K4ECGX Samsung
Cost:	10	8	9	10
I/O Ports:	20	10	7	5
Programmability:	20	10	6	8
Wi-fi Capabilities:	10	9	7	10
Processor Speed:	20	10	7	9
Storage:	20	7	9	10
Score =	100	910	740	840

Functional requirements for the microcontrollers Pugh matrix are cost, I/O ports, programmability, Wi-Fi capabilities, processor speed, and storage. The three microcontrollers that were being analyzed are the Arduino uno rev3, raspberry pi4, and the esp32. The cost weighed about 10 percent of the total with all of the microcontrollers ranging from a certain range, but with some slightly cheaper than the others. The I/O ports were rated at (20) because each microcontroller had either only 2 or 3 I/O ports while others had 5 or more. The Programmability was also weighed the same because some microcontrollers can only be programmed in one language while others can be programmed using more than one.

The Wi-Fi capabilities were weighed at (10) because it is a feature that is useful without having to use an ethernet cable. The Processor speed is the same weight of (20) and we need this to be able to make sure that the microcontroller can process what we send to it fast enough to keep up with what we are doing. Storage is also a big part of the weight with (20) having a lot of storage is good so that we can store the program code. With all these functional requirements, we can see that the Raspberry pi4 is the best option with a score of 910, the ESP32 is the second-best option with a score of 840 and then the Arduino uno rev3 is the last option with a score of 740.

11.0 Detailed Design

11.1 Hardware

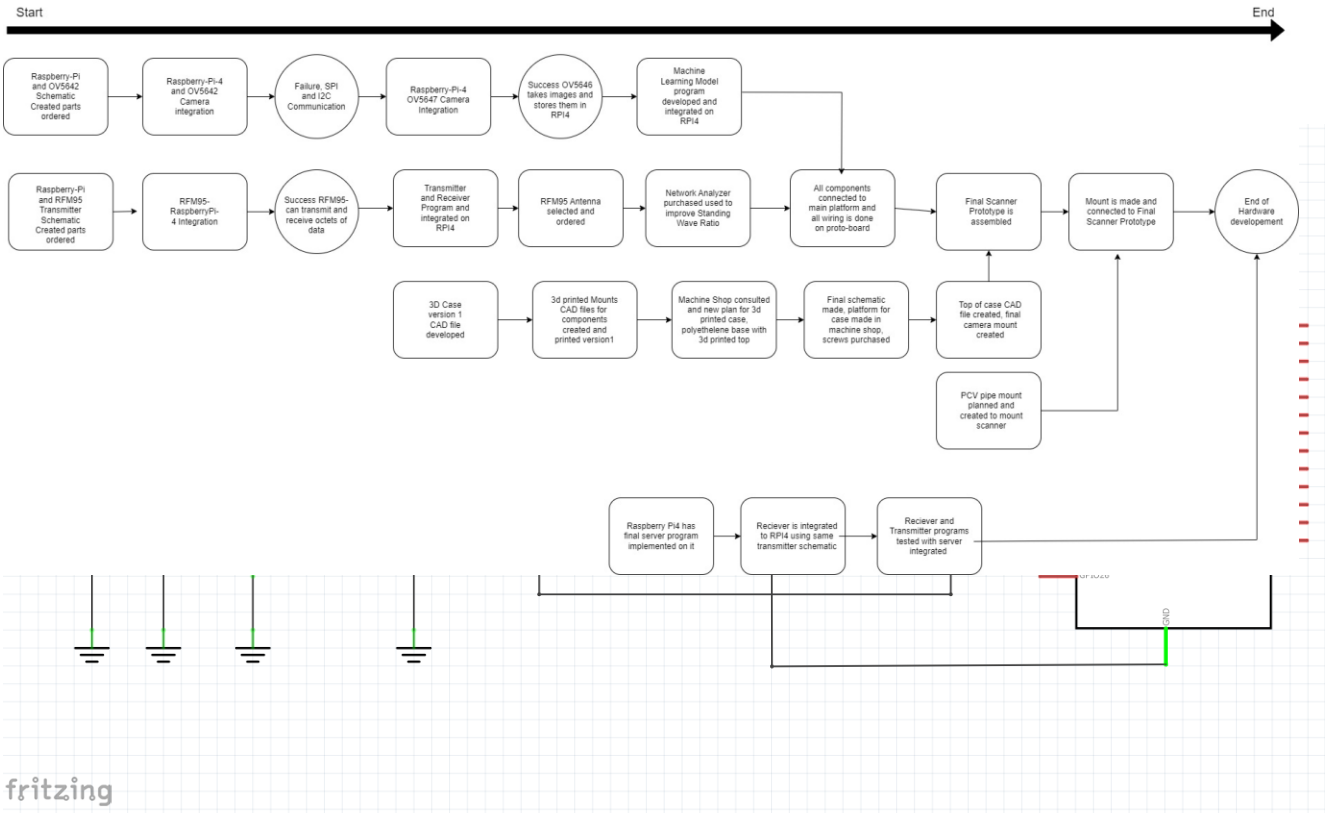


Figure 3: Initial Schematic

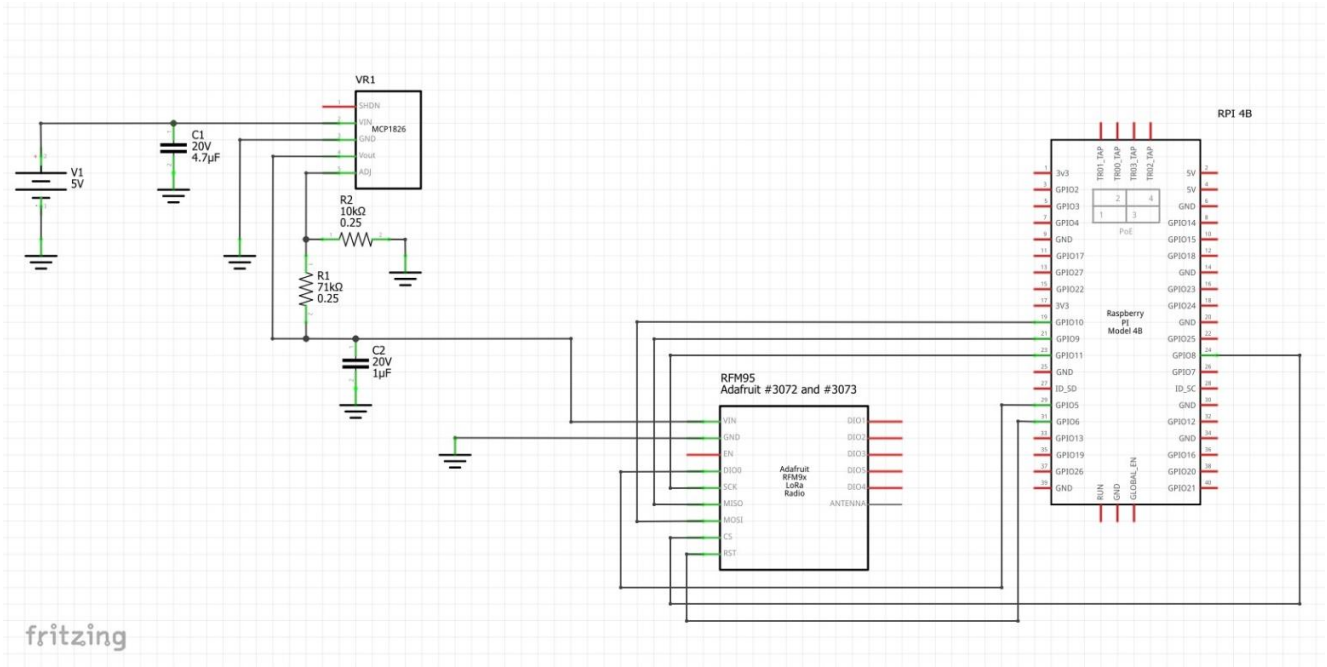


Figure 4: 2nd version of schematic without camera

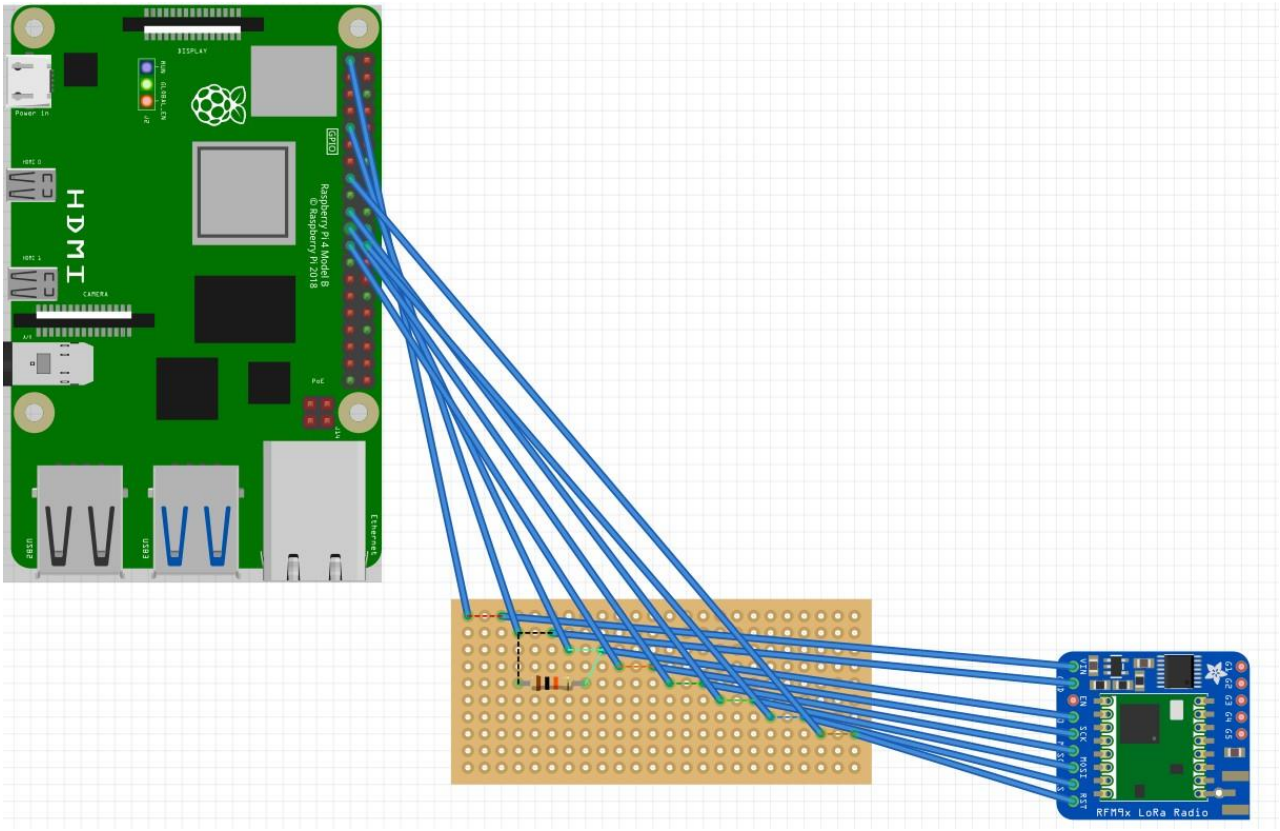


Figure 5: Final Schematic without camera

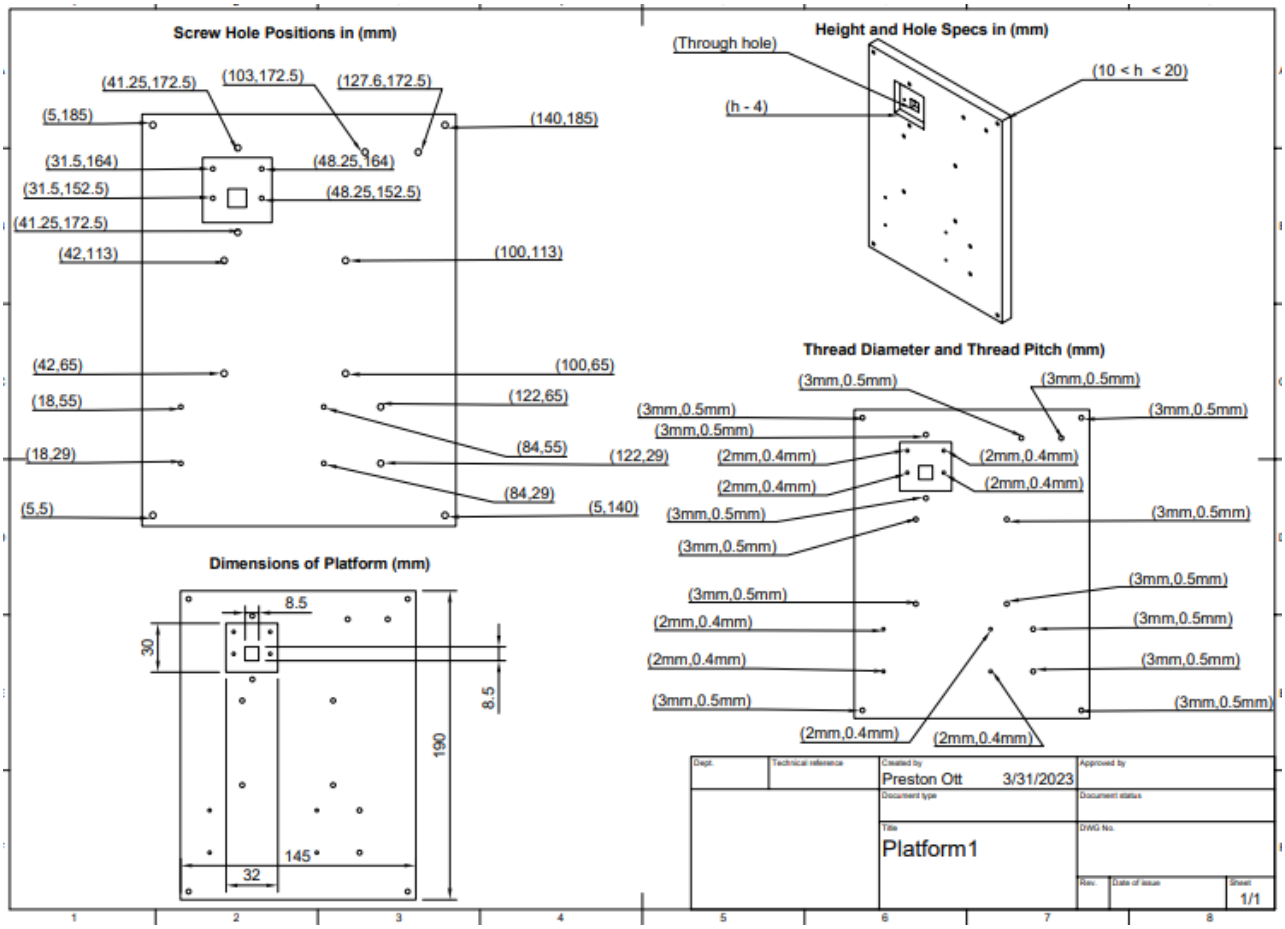


Figure 6: Plan for enclosure

The hardware ideas were split up between the majors. The computer engineers were to determine the best microcontroller to be used for the purpose of the project. While the electrical engineers were to focus on the transmitter/receiver, power supply, and the scanner.

11.1.1 Power

As seen in Figure 2, the raspberry pi was initially going to be powered with a voltage regulator. We then switched which regulator to be used after checking the power requirements for the raspberry pi as seen in figure 3. The resistors and capacitors were added to configure the regulator for the raspberry pi based of the data sheet. As time progressed, we saw more use with using the regular power supply made by a company and get rid of the regulator all together.

11.1.2 Scanner

The first iteration of the project used a different scanner (camera) than the final version. The scanner chosen ended up being a camera due to the capability of scanning multiple parking spots, which would otherwise be challenging with other sensors, such as ultrasonic. Early in the process the OV5647 was chosen to be the scanner, but due to

difficulty interfacing with the raspberry pi, we decided to switch to the OV5642. Which was one of the first cameras produced to interface directly with the raspberry pi. This made the process of taking images much simpler, while still being an efficient camera for the project.

11.1.3 Transmitter/Receiver

The transceiver chosen held from beginning to end as seen in the schematics. The RFM95W was chosen to serve as the transmitter and receiver. One of the team members has had previous experience with the module at a past internship and has held this module with high regard. The transceiver chosen needed to be able to have a wide range to adapt to different parking lots, it also had to be able to send at least 80 parking spots worth of data. The RFM95, which has LoRa modulation scheme, has both long range while being able to send up to 255 bytes (or in this case parking spots) of data making it a good fit for Digipark.

11.1.4 Enclosure

Although not included in the hardware block diagram, the device needs to have an enclosure to secure and stabilize all of the parts. The process to design the case was to make a form factor efficient in size so that it could fit all the parts while being as small possible. The material to be used for printing was a resistant material to weather conditions while not conducting electricity. The material chosen that fit these requirements was ABS.

11.2 Software

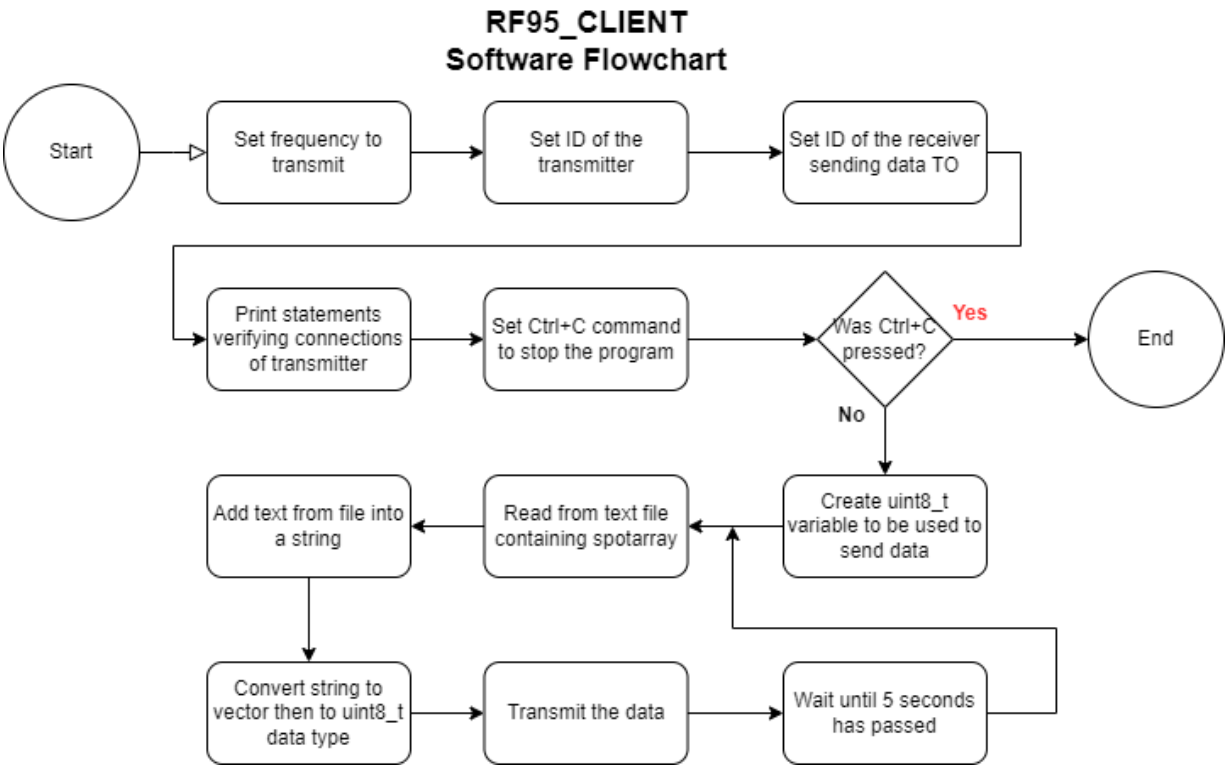


Figure 7: Transmitter Flow Chart

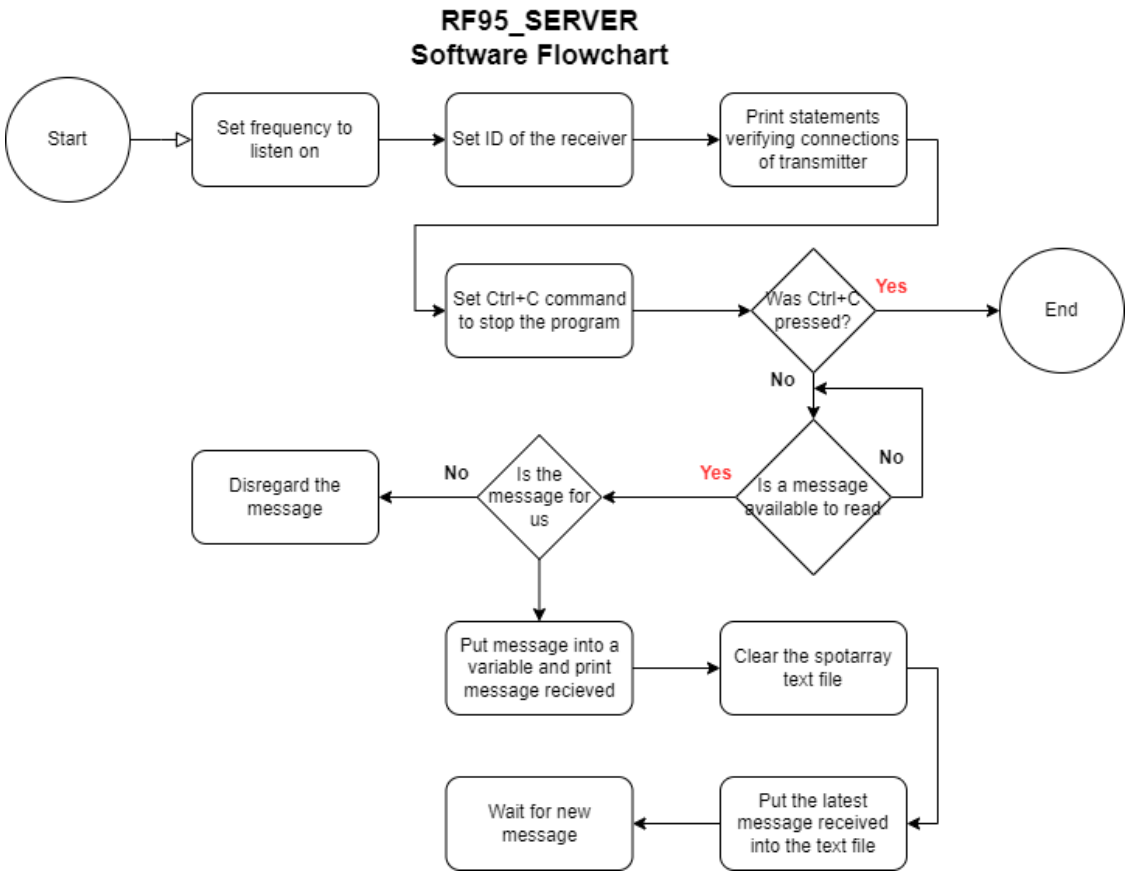


Figure 8: Sever Software Flowchart

The division of the software development is relatively straight forward. Having a member work on the code for whichever component they are most familiar with. While those with more advanced coding experience wrote code for the server and app.

11.2.1 Transmitter Software

The software starts by initializing the module itself, there is a required set of information for the first 8 bytes of every LoRa message, which contains information such as the name of the transmitter and where to send the data to. The program is setup to type Ctrl+C in the Linux terminal to end the program, but otherwise continue the program forever, since the device would always scan for parking spots. The Ctrl+C command would be used during less busy hours of a parking lot to save power. The way the module work, the data needs to be formatted into an array of the uint8_t data type, each array spot would contain 1 byte of information. The data being transmitted is the spot array which comes from a text file, which the transmitter program has to read. After reading the text file, a few data type conversions occur since there is no direct conversion from a text file to a uint8_t data type. After the conversions are complete, the transmitter finally sends the data to the receiver, and continually does this every 5 seconds.

11.2.2 Receiver Software

The receiver software starts by initializing the module, similar to the transmitter. The difference is that the ID will be used in a function that detects if the LoRa message was for the specific receiver. A similar loop of using Ctrl+C to end the program is used in the receiver as well, except the receiver always listens for a message. Once a message is received, the code checks if the message was for the receiver and discards the message if not. If the message is for the receiver, a second function is called to read the message and store the data in a variable. Afterwards, the program puts the message into a text file to be read and be manipulated by the server and database and continues to listen for a new message.

11.2.3 App Software

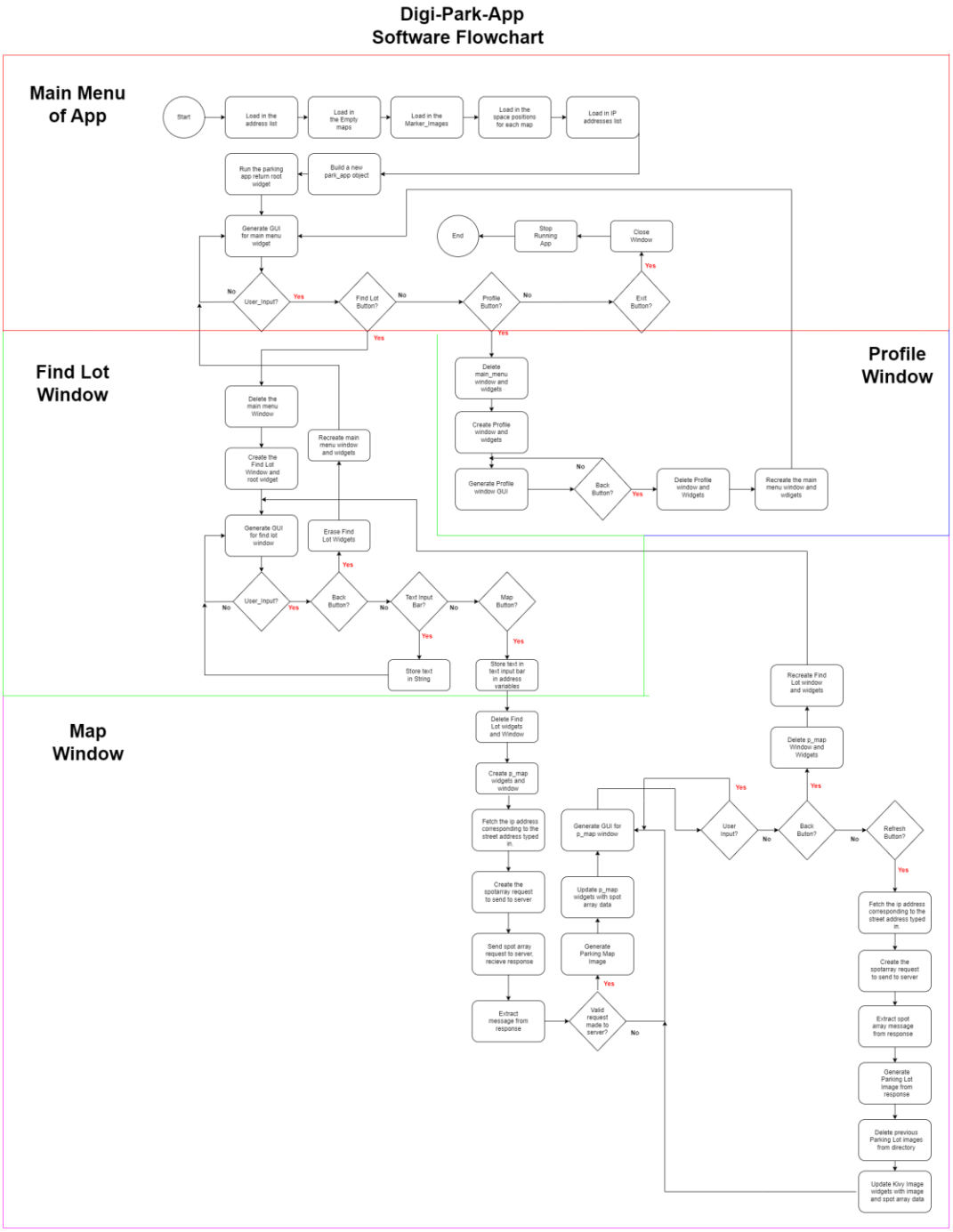


Figure 9: App Software Flowchart

The app software starts by loading in all the content Empty maps, position of parking spots, ip addresses, street addresses of parking lots, that it will need to properly function. Then an app object is created to generate the main menu window. From the main menu

the user can navigate to profile window, find lot window. The profile window is not yet created and just contains some text and a back button. The Find lot window allows users to search through the street addresses and search for their own street address. Once a street address is typed into the text input the user can select the map button to receive the current parking lot map for the street address typed in. The app will send a request to the appropriate server for the (spot array) for that parking lot. The app will receive a (TCP) response from the server containing the spot array and close the (TCP) connection. The appropriate map from the Empty map folder will be selected and the parking lot map image will be drawn. Then statistics on the parking lot will be generated from the (spot array) received. The map window will then be drawn with the map image displayed to the user and the parking lot statistics. The user can then choose to refresh the page with the refresh button which will send another (spot array) request to the server to update the parking map image and statistics again. The user can also go back to the find lot window by clicking the back button. The user can navigate back to the exit button and click it, once clicked the window for the app will be closed and the app will stop running. The Kivy library documentation was used to develop the main Kivy app reference.

11.2.4 Machine Learning and Camera Software

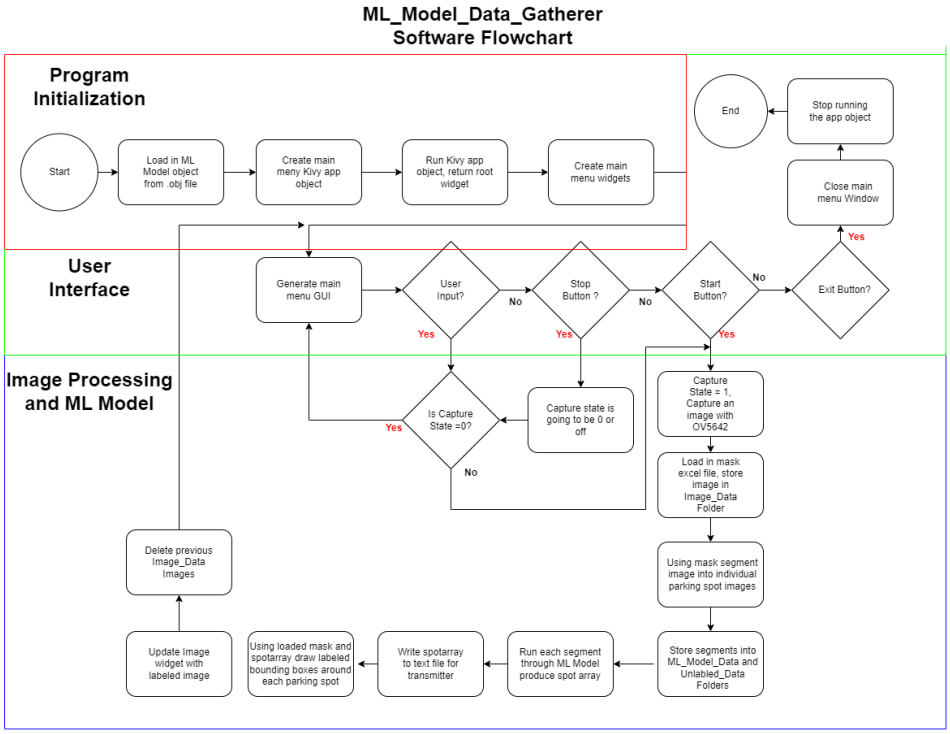
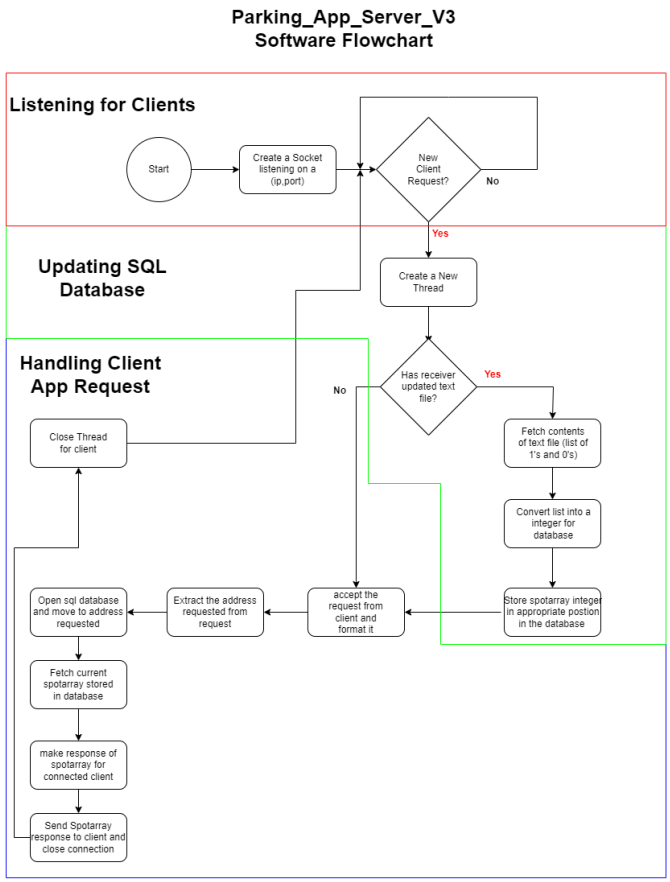


Figure 10: ML Software Flowchart

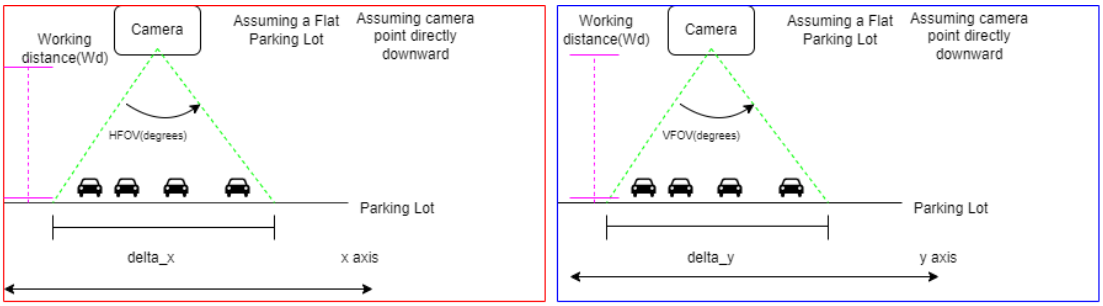
The Machine Learning Model and Camera and Software is within a program called ML_Model_Data_Gatherer.py. The program basically goes through the process of scanning and analyzing the parking lot. The program starts by loading a sci-kit-learn MLP Machine Learning Model object into the program. Then it creates a kivy app object and runs it. The app object creates the user interface menu for the user. The user can type in the path to the mask that defines the bounding boxes for the parking lot image. The user can press the stop button to stop scanning the parking lot, the user can press exit to close the kivy window and stop running the application. The start button in the user interface will first take an image of the parking lot and store it. Then it will use the mask to extract images of each individual parking spot. Each parking spot image will be fed through the sci-kit-learn Mahine Learning Model where it will predict if image contains a car (1) or doesn't contain a car (0). Each 1 and 0 is concatenated together to form the spot array for the parking lot. The spot array is then written to a text file spotarray.txt where the transmitter program can read it. Then using the spot array, we generate an image of the parking lot with a bounding box drawn. Then feed the image to the kivy image object to display the image. Then we delete the previous images of the parking lot that are taken and continue to take and process images until user clicks the stop button or exits the application. All code for the Machine Learning Model was made with reference to the sci-kit-learn documentation reference 9 and book Hands on Machine Learning with sci-kit-learn, Keras, and TensorFlow reference 13 . All code for the OV5642 camera was developed using the lib-camera documentation reference 10.

11.2.5 Parking App Server Software

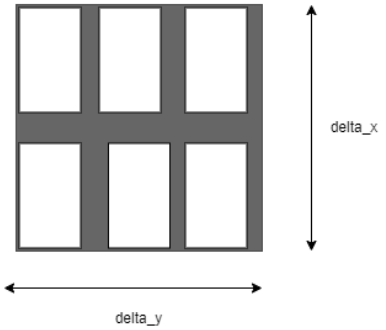


The parking app software is within a program called Parking_App_Server_V3.py. The server, which is a Raspberry PI 4, is integrated with the receiver and receives the spot array from the transmitter and updates a SQL database. It also handles processing the requests from clients using our parking app. The Server starts by creating a TCP socket and listening for any TCP connections from clients. Once a new client connection is made, we make a new thread to handle them in, we check if the spotarray.txt text file holds new spotarray data. If it has changed then we update the database with the new spot array for the parking lot street address. Then we accept the request from the client connected and extract the street address of the parking lot requested. We navigate through the SQL database to find the latest spot array for the street address. We format the spot array into a response to send to the client. We send the response to the client and close the TCP connection and end the thread. Then we return to listening for more clients TCP connections. The client and server functions of our project were made with reference to the book Learning Python Network Programming, reference 14.

11.3 Engineering Analysis and Calculations



Calculation of Parking Lot Area Scanned



1.) Area Scanned = delta_x * delta_y

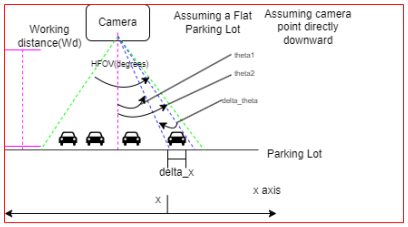
2.) delta_x = 2 * Wd * tan(HFOV/2)

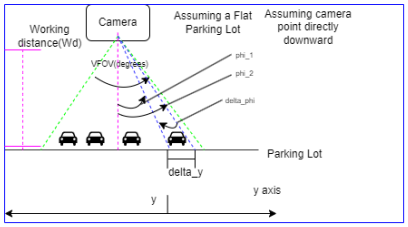
3.) delta_y = 2 * Wd * tan(VFOV/2)

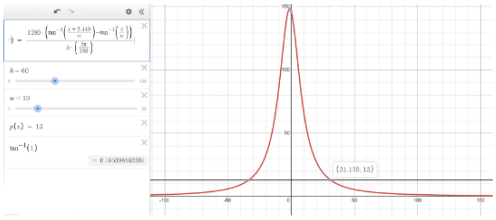
4.) Area Scanned = 2*Wd*tan(HFOV/2)*2*Wd*tan(VFOV/2)

5.) Area Scanned = 4 * (Wd)^2 * tan(HFOV/2) * tan(VFOV/2)

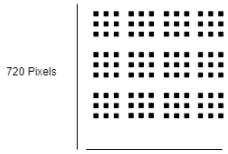
Figure 11: Area Scan Calculations







Maximum x distance of Vertical Camera Calculation



(Pixels in CMOS sensor of OV5642 Camera)

1.) Minimum pixel size per parking spot = 30pixels x 40 pixels

2.) Size of Texas Parking Spot = 2.44m x 3.350m

3.) delta_x = 2.44m

4.) delta_y = 3.350m

5.) minimum linear x pixel density = 30pixels/2.44m = 13 pixels/m

6.) minimum linear y pixel density = 40pixels/3.35m = 12 pixels/m

7.) Linear pixel density x = 1280 * (atan(x+delta_x/Wd) - atan(x/Wd)) / (HFOV)

8.) Linear pixel density y = 720 * (atan(y+delta_y/Wd) - atan(y/Wd)) / (VFOV)

9.) 13 = 1280 * (atan(x+2.44/Wd) - atan(x/Wd)) / (HFOV)

10.) 13* (HFOV)/1280 = atan(x+2.44/Wd) - atan(x/Wd)

11.) 13* (HFOV)/1280 = atan(x+2.44/Wd) - atan(x/Wd)

12.) Non-Linear equation, has to be solved by numerical methods on a computer assuming Wd and HFOV is known for camera.

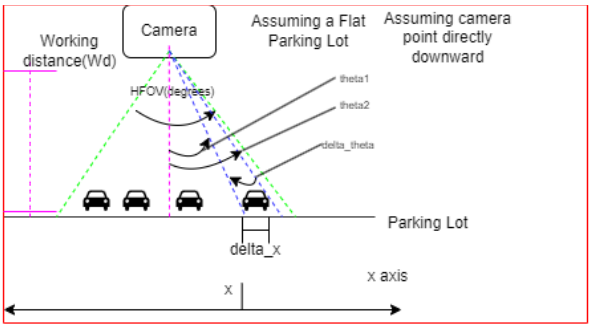
Example of Maximum x distance from camera made in Desmos

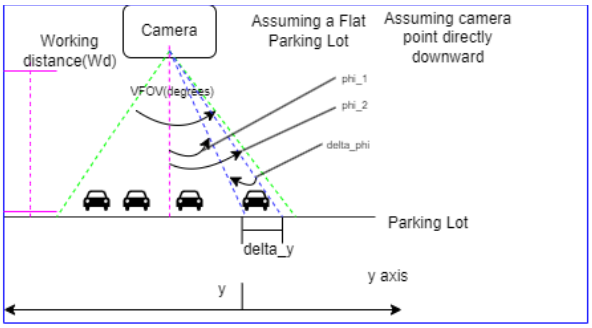
1.) Working distance (Wd) = 10 meters

2.) HFOV = 60 degrees

3.) max x distance = 31.16 meters

Figure 12: Maximum x Distance Vertical Camera Calculations

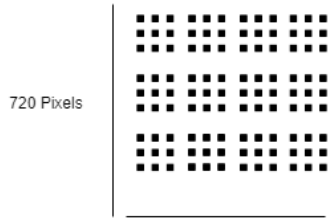




Calculation of Pixel density per Parking Spot

(Used to compute the maximum parking lot area our scanner could possibly scan)

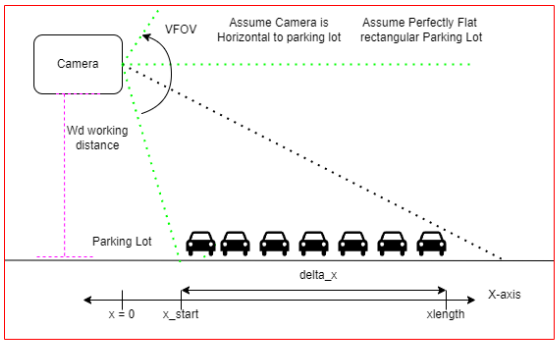
((x ,y) is the pos of the parking spot from the camera orgin)

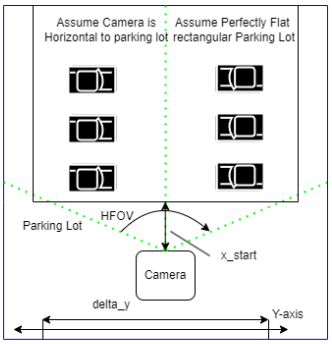


Pixels in CMOS sensor of OV5642 Camera

1.) Minimum pixel size per parking spot = 30pixels x 40 pixels
2.) delta_x = 2.44m
3.) delta_y = 3.350m
4.) Size of Texas Parking Spot = 2.44m x 3.350m
5.) Size of Texas Parking Spot = 2.44m x 3.350m
6.) delta_theta = theta2 - theta1
7.) delta_phi = phi2 - phi1
8.) theta1 = atan(x/Wd)
9.) phi1 = atan(y/Wd)
10.) theta2 = atan(x+delta_x/Wd)
11.) phi2 = atan(y+delta_y/Wd)
12.) delta_theta = atan(x+delta_x/Wd) - atan(x /Wd)
13.) delta_phi = atan(y+delta_y/Wd) - atan(y /Wd)
14.) Linear pixel density x = 1280* (delta_theta/HFOV)
15.) Linear pixel density y = 720* (delta_phi/VFOV)
16.) Linear pixel density x = 1280 * (atan(x+delta_x/Wd) - atan(x/Wd)) / (HFOV)
17.) Linear pixel density y = 720 * (atan(y+delta_y/Wd) - atan(y/Wd)) / (VFOV)

Figure 13: Pixel Density Calculations





Calculation of Area Scanned for Horizontal camera

1.) Assuming a Perfectly flat parking lot

2.) Area Scanned = delta_x * delta_y

3.) The xlength of the parking lot will be completely scanned.

4.) The ylength that we can guarantee be completely scanned = delta_y

5.) delta_y = 2 * x_start * tan(HFOV/2)

6.) delta_x = xlength

7.) Area = xlength*2*x_start*tan(HFOV/2)

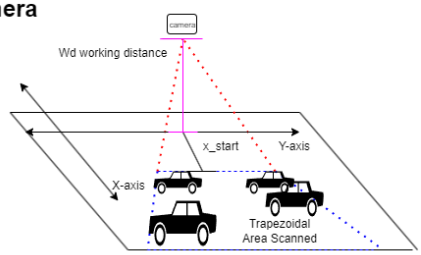
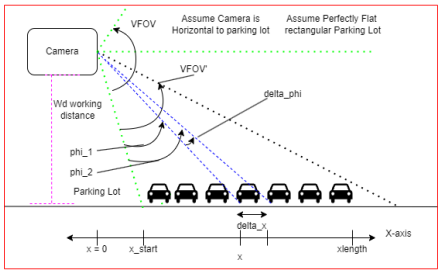
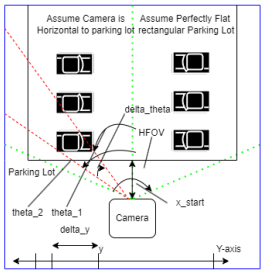
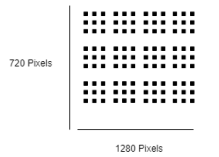


Figure 14: Area Scanned Horizontal Camera Calculations







Calculation of Horizontal Camera Pixel Density

1.) delta_x = xlength scanned

2.) delta_y = ylength scanned

3.) Area scanned = delta_x * delta_y

4.) x = x position of length scanned

5.) y = y position of length scanned

6.) delta_phi = atan(x+delta_x/Wd) - atan(x/Wd)

7.) delta_theta = atan(y+delta_y/x+delta_x) - atan(y/x)

8.) VFOV' = atan(Wd / xlength-x_start)

9.) x pixel density = (delta_phi / VFOV')* 1280

10.) y_pixel density = (delta_theta / HFOV')*720

11.) Horizontal pixels = 1280

12.) Vertical pixels = 720

13.) x pixel density = (1280 / VFOV')* (atan(x+delta_x/Wd) - atan(x/Wd))

14.) y_pixel density = (720 / HFOV')*atan(y+delta_y/x+delta_x) - atan(y/x)

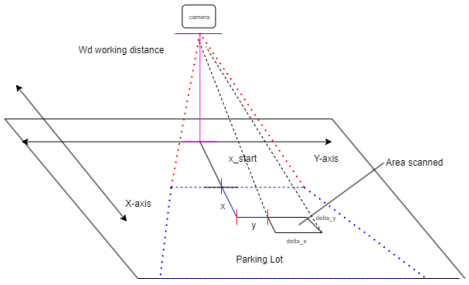


Figure 15: Horizontal Camera Pixel Density Calculations

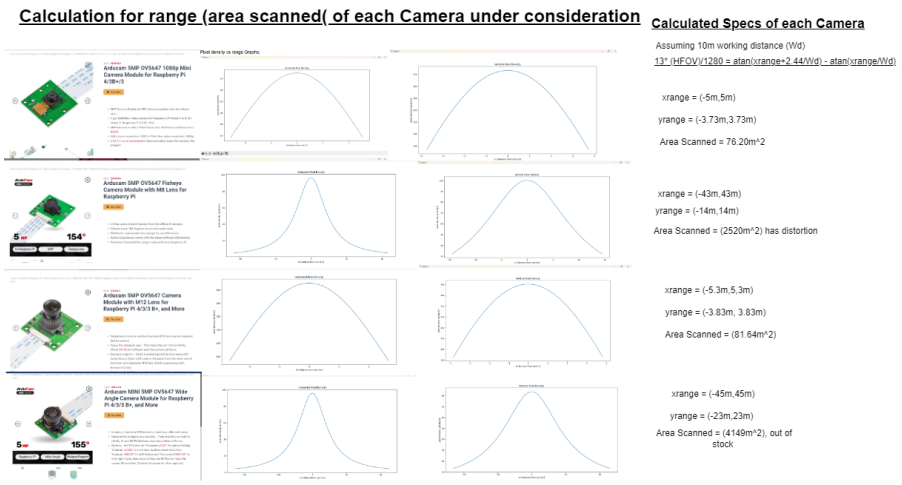


Figure 16: Calculations of range of different possible Raspberry Pi 4 cameras

3D Printed Case for Scanner Dimensions Calculation

Dimensions = 85.6mm x56.5mm x 20mm

Dimensions = 25mmx24mm

Dimensions = 16mm x 16mm

1.) Combine all x lengths = 85.6mm+25mm+16mm = 126.6mm

2.) Combine all y lengths = 56.5mm+24mm+16mm = 96.5mm

3.) Multiply x length by 1.5 for possible error = 126.6mm*1.5 = 190mm

4.) Multiply y length by 1.5 for possible error = 96.5mm*1.5 = 145mm

5.) RPI4 highest x length multiply by 1.5 for error = 20mm*1.5 = 30mm

6.) Case rectangular prism dimensions = 190mm x 145mm x 30mm

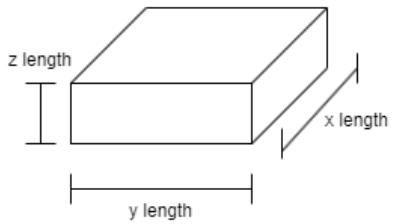
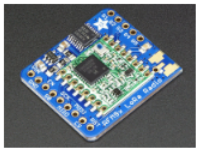


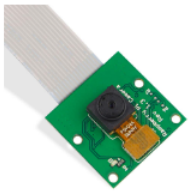
Figure 1: #D Case Dimensions

Power Consumption Calculations

1.) USB-C Power Cable
2.) 5.1V DC input
3.) 3A DC input max current draw
4.) Max Power Consumption = 15.3watts



1.)Power Pins Vin and GND
2.) 3.3V DC Input Voltage
3.)Max transmitting current draw = 150mA
4.)Max power = 3.3V * 150mA = 0.5watts
5.) Powered by the RPI4



1.)Connector Strip with power lines Vin and GND
2.) 3.3 DC Input Voltage
3.) Max current draw = 300mA
4.) Max power = 3.3V * 300mA -= 1w
5.) Powered by the RPI4

Voltage, Power, Current Needs

1.) Input Voltage = 5.1V
2.) Max Current Draw = 3A + 150mA + 300mA = 3.45A
3.) Max Power Draw = 15.3watts+0.5watts+1w = 16.8w

120V Walloutlet and standard RPI4 AC to DC adaptor option 1

1.) 120V AC, Can provide 20A maximum
2.) 120V AC, Can provide 1500w maximum
3.) RPI4 AC-DC adator Vin = 100V-250V AC,I = 0.5A 50-60HZ
4.) RPI4 AC-DC adaptor Vout = 5.1V reguated DC, I = 3.0A

5 units, 2S, 5200mA hour, 4watt* hour LIPO, battery, and a voltage regulator, option 2

12 hour operating time
16.8w max power drawn
12hours *16.8watts = 201.6watt*hours
2S LIPO provides = 7.4V Input Voltage
7.4V*5.2Amp*hours*5 = 192.4watts
%power difference = 4.56%, 11.45 hours, close enough

Figure 1: Power Consumption Calculations

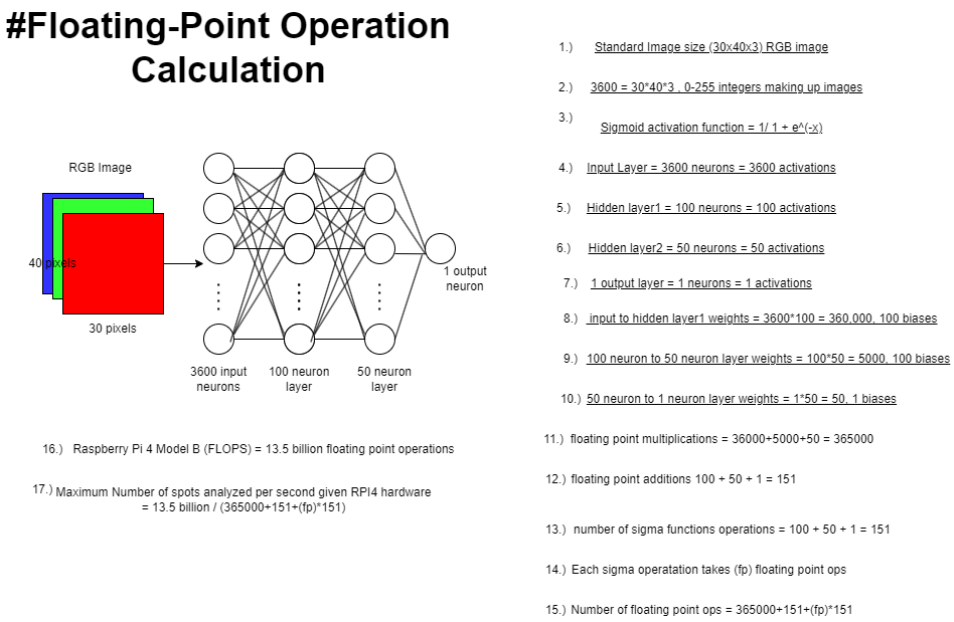


Figure 19: Machine Learning Model Floating Point Operations Calculations

There are 9 Engineering Analysis Diagrams. 6 diagrams analyze the area and pixel density of the scanner. The other 3 are calculations for power consumption and requirements of our scanner module, 3d printed enclosure dimensions, and the Machine Learning Model number of floating-point operations. The area and pixel density of the scanner calculations were made with reference to the document by Jax Zhang reference 7.

Figure 11 shows the calculations of area scanned for a vertically oriented camera (camera point straight down at parking lot from above).

Figure 12 shows the calculations for the maximum distance we can scan up to and have sufficient pixel density for our Machine Learning Model to work.

Figure 13 shows the pixel density calculations for standard Texas parking space a position (x,y) away from the vertical camera.

Figure 14 shows the calculations needed to compute the area scanned of a horizontally oriented camera (camera pointed along the parking lot).

Figure 15 shows the calculations needed to compute the pixel density of a patch of area in the parking lot a distance (x,y) away from the camera and a patch size of (x_delta * y_delta).

Figure 16 shows a set of different Raspberry Pi 4 camera and the range of the scan they can provide. Given a 10meter working distance and a vertcially oriented camera.

Figures 11-16 are important in knowing how many spaces you can scan with a single module and determining how many modules you may need to fully scan a parking lot.

Figure.

Figure 17 shows how the dimensions of the 3d printed enclosure were estimated based on the size of each component making up the scanner module.

Figure 18 shows how the power consumption, voltage and current draw of the scanner module were estimated by looking at each components required voltage, max current draw, max power

draw. 2 options for powering the scanner are proposed, a mobile LIPO battery, or a wall outlet with an AC-DC voltage regulator.

Figure 19 estimates the number of floating-point operations our Machine Learning Model will need to perform in order to make a (occupied) (unoccupied) prediction on an image of a parking space. This is important because it defines the limitations of our hardware and gives us an estimate of how many seconds, we'll need to fully scan a parking lot with parking spaces.

12.0 Major Problems

12.1 Error Code with Transmitter and Server

An issue would occur occasionally between the server and transmitter where the encoding of the transmitter wasn't UTF-8 and therefore couldn't be properly put into a text file. The purpose of the text file was to communicate between the different programs because the transmitter/receiver was written in C++ and the other programs were written in Python. The server would read from the text file, and if the text file didn't have UTF encoding then the server would run an error and stop the program completely. The work around was to run the program again, because the error wouldn't occur frequently, another solution would be to solder the antenna on with better connections.

12.2 Obtaining a Testing Location and Footage

Procuring a location to test our prototype would become a reoccurring issue, with the UTSA campus not wanting anything attached to light fixtures. After consulting our instructor, a compromise we came to with the campus was to build a temporary stand and set up the scanner inside one of the buildings next to a lot, looking outwards towards the lot from a window, if a class isn't in session during recording.

12.3 Machine Learning Model %accuracy and computational expense

The final Machine Learning Model trained had high accuracy on occupied parking lots with a high % of spots occupied. The Machine Learning model had lower accuracy on parking lots with high% of spots unoccupied. This major problem most likely comes down to the low parameter count for Machine Learning model and the dataset used only containing images from 3 parking lots over the course of a year. The Raspberry PI 4 hardware limits the parameter count for our Machine Learning Model, since a certain number of floating operations must be done for the model to make a prediction. This number is proportional to the number of parameters for a model. Better hardware such as a Nvidia Jetson Nano can provide more floating-point operations and support a more complex Machine Learning Model.

13.0 Integration and Implementation

Hardware wise, ensuring the parts were compatible and communicating with each other was vital. Thus, each component was wired together to confirm connection and programmability of the raspberry pi was possible. Also, the team would test the individual software compatibility with each other. From the transmitter to the server which would then connect with the app, several test cases were designed to ensure information was effectively passed from one device to another and accurately translated by the recipient. Below are the various tests that were performed:

13.1 Machine Learning Model Training and Prediction Capability

- Machine Learning Model is successfully trained and can make predictions on test images.
- Results: ML Model successfully identified most objects and spaces within the given images. The processing of the image data is sucessful converting the 30x40 RGB images into a 1d array. The training of the ML Model is sucessful stochastic gradient descent improving its %accuracy on each gradient descent step. The model is able to make predictions on the same 30x40 RGB images.

13.2 Machine Learning Model %accuracy requirement

- Machine Learning Models accuracy on real world data is greater than 85%.
- Results: The Machine Learning Models %accuracy was higher than 85% on parking lots that are mostly occupied. On parking lots where most of the spots are empty the %accuracy fell too 70% for the parking lot. The Machine Learning Model has difficulty detecting when spots are empty.

13.3 Machine Learning Model Overfitting Requirement

- Machine Learning Models (%error) between training and validation data is less then 5%.
- Results: The ML Model Overfitting %error between validation data accuracy and training accuracy was well below 5%. The graph below shows how the ML Model's %accuracy over epochs of training, red being training set accuracy and orange being validation accuracy. There may be an issue with the validation data, and training data, being too similar since they take from the same training set because the ML Model's %accuracy on real world data is low.

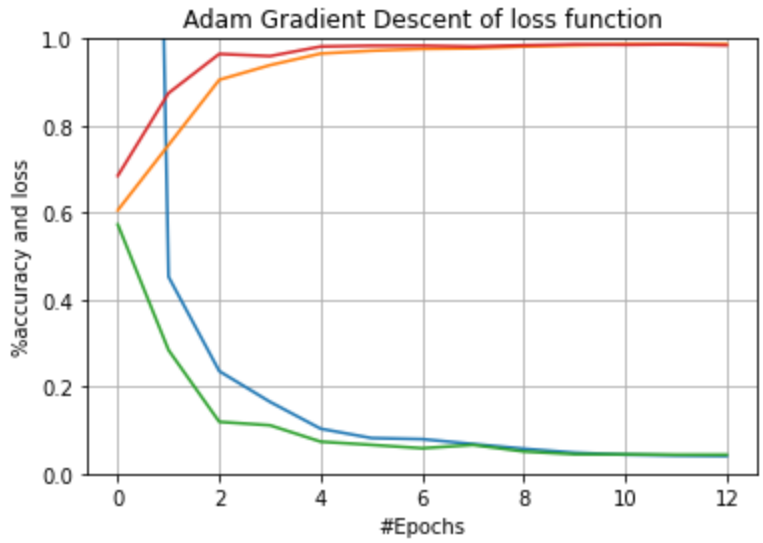


Figure 16: Adam Gradient Descent of loss Function

13.4 Transmitter/ Receiver Reliability

- 95% of data is communicated between the two devices
- Results: Over 95% of data was successfully communicated, initially without the antenna (the test setup) not every piece of data transmitted would successfully be read by the receiver. With the new set up (proper antennas) the transmitter/receiver was able to achieve a higher percentage of over 95%. The takeaway is that the transmitter and receiver chosen was sufficient for the needs of the project.

13.5 Transmitter/ Receiver Range

- Data is communicated between 2 devices at a minimum range
- Results: For the purpose of Digipark, the device located in the parking lot must be properly equipped to cover multiple sized parking lots. The determined range to be sufficient for the project is 500 meters. In this test case, the device was able to transmit and receive data just a little bit over 500 meters. An urban environment was used for this test, similar to that of a parking lot. The results prove that the transmitter and receiver can adjust to different sized parking lots, specifically in urban environments.

13.6 Camera Functionality

- Camera takes images
- Results: Images were successfully captured via camera module.

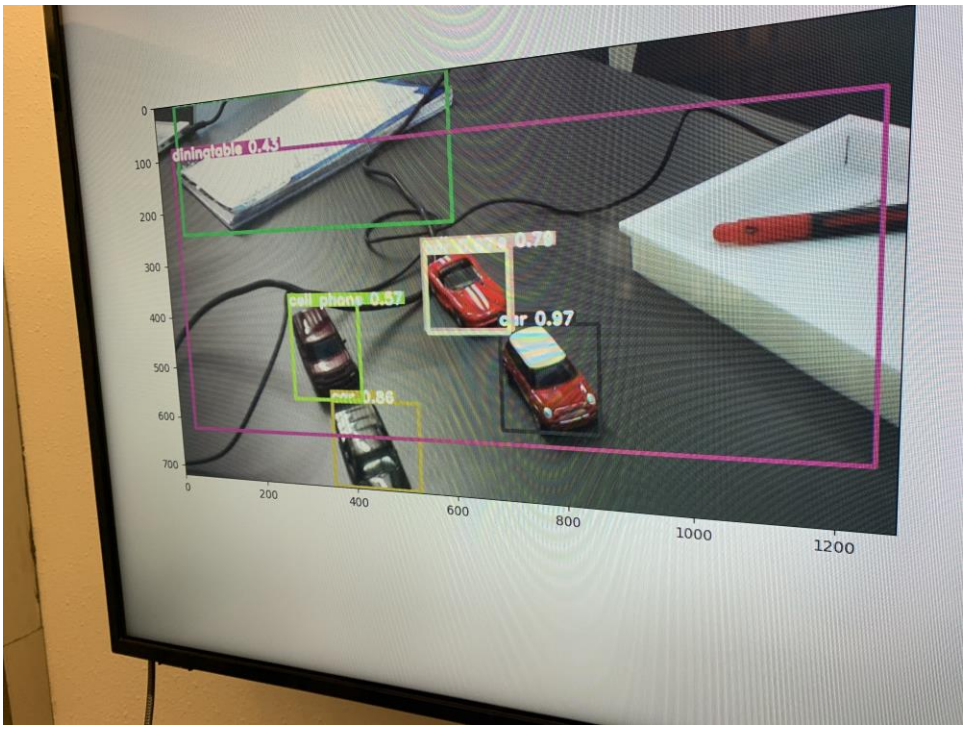
13.7 App Functionality

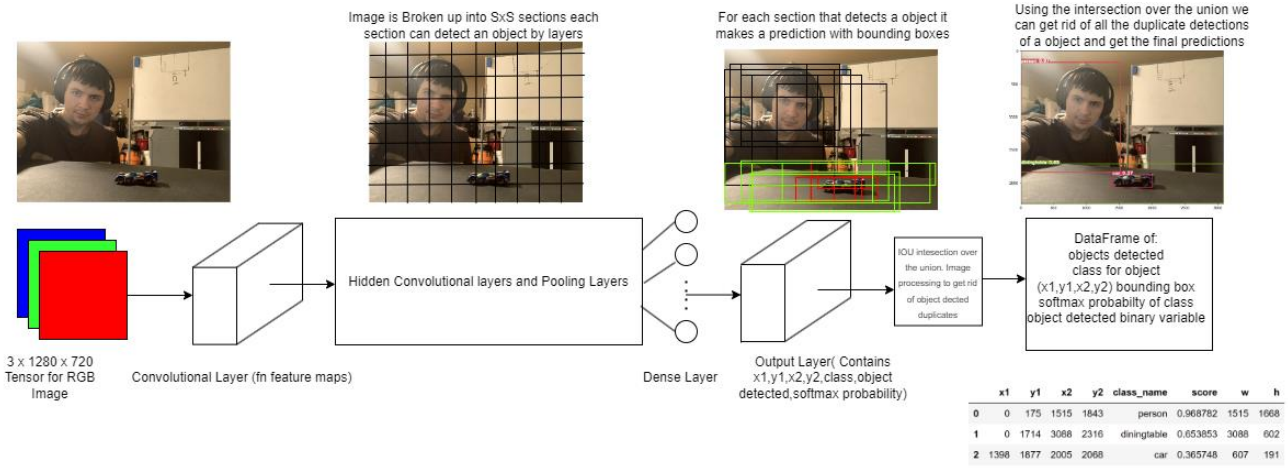
- Information Accurately Presented for User
- Results: Application successfully retrieved information from server and translated that data into an understandable form of information for the user. App is heavily dependent on server being operational for most functionality but still effective.

14.0 Comments and Conclusion

The development of Digipark was a challenging yet rewarding experience for our team. We encountered technical issues such as error code problems and securing a suitable testing location, but we worked collaboratively and persistently to overcome these obstacles. Our efforts led to the successful integration and thorough testing of all hardware and software components, including tests for machine learning model capability, accuracy, overfitting, transmitter/receiver reliability, and range. Throughout this project, we gained valuable insights into the importance of effective communication, collaboration, and thorough testing in engineering design. Ultimately, the development of Digipark demonstrated the practical application of engineering design and testing in addressing real-world problems

Comment on Machine Learning Model. The original plan for the project was to train and use a convolutional neural network as our Machine Learning Model to analyze images taken of parking lot. An effort was made to learn Tensor Flow and Keras API in order to do this. A few low parameter convolutional neural networks were trained on the same dataset as the Multi-Layer-Perceptron model. These models did not perform better than the Multi-Layer-Perceptron, most likely again from the dataset not generalizing to all parking lots and low parameter count and because of this they were not utilized. At the tech symposium a pretrained YOLO convolutional neural network was implemented that had a higher parameter count ,but could do object detection as well as object classification. This would be the Machine Learning Model used for the next version of the project. The model is a fully convolutional neural network which means it only utilizes convolution layers of neurons and pooling layers and can take in images of various sizes. Below is image of it working on a set of hot wheels cars and a diagram of the basic architecture of the YOLO CNN model. The diagram made and the use of this Machine Learning model came from the original paper on the YOLO model from Redmon, Joseph reference 6. The convolutional neural network models were developed using the TensorFlow documentation reference 11.





15.0 Team Members

15.1: Preston Ott, Design Lead - Contributed across each module of the project. The Machine Learning Model data processing and training, the framework and code for the parking app, entire program for the camera and Machine Learning Model, development of CAD files for 3d printed enclosure/case, 3d printing of each part for the enclosure/case, machine shop fabrication of the platform. Integration of the transmitted and scanner. Integration of the server and the app. Test cases for the ML Model, conceptual work and engineering analysis for the parking app, Machine

Learning Model, app server, 3d printed enclosure, optics of the camera. Research was also done on Machine Learning to improve models. Pretrained Yolo Convolutional Neural Network integrated into main program for Machine Model and camera for tech symposium presentation.

15.2: Younes Younes, Embedded Design Engineer - Contributed to various aspects of the project, including the creation of the schematic design, assembly of the prototype, platform case assembly, and the development of transmitter/receiver test cases. Researched similar projects.

15.3: Joshua Swanner, Secretary and App developer – Kept times in meetings and co-devolved application for user interaction. Also, tests various application functions alongside other aspects of the projects and researched similar projects for initial inspiration.

15.4: Joshua Moya, App developer and prototype - Assisted with the app that was used up until version 3 of the app. Continued to then work on the 3D model for the case and made the case for the prototype along with the mount for the prototype. Assembled the final prototype and developed camera test cases.

15.5: Luis Santillan, Embedded Design Engineer – Contributed to a lot of the hardware needs, from schematic development and selection of parts. Had the largest influence on the transmitter and receiver aspect of the project. Helped write the code for this module and integrate and implement with the other parts of the project, such as the ML model and server programs.

16.0 References

"IEEE Recommended Practice for Radio Frequency Safety Programs, 3 kHz to 300 GHz," in *IEEE Std C95.7-2014 (Revision of IEEE Std C95.7-2005)* , vol., no., pp.1-58, 8 Aug. 2014, doi: 10.1109/IEEESTD.2014.6874474.

"IEEE Standard for Radio-Frequency Energy and Current-Flow Symbols," in *IEEE Std C95.2-2018 (Revision of IEEE Std C95.2-1999)* , vol., no., pp.1-28, 5 Oct. 2018, doi: 10.1109/IEEESTD.2018.8486934.

"IEEE Guide for EMF Exposure Assessment of Internet of Things (IoT) Technologies and Devices," in *IEEE Std 1528.7-2020* , vol., no., pp.1-90, 11 Jan. 2021, doi: 10.1109/IEEESTD.2021.9319817.

Cleverciti sensor. Cleverciti. (2012). Retrieved May 2, 2023, from https://www.cleverciti.com/en/innovations/cleverciti-sensor

ATTAR, O., HASSON, O., & NAAMANI, A. (2021, May 20). *a training system for automatically detecting parking space vacancy*.

Redmon, Joseph, et al. “Homes.Cs.Washington.Edu.” *Https://Homes.Cs.Washington.Edu/*, June 2016, homes.cs.washington.edu/~ali/papers/YOLO.pdf.

Zhang, JAX. “What’s the Mean of FOV, HFOV, VFOV, and DFOV for Security Cameras? " Kallglow.” *KallGlow*, 9 Apr. 2022, www.kallglow.com/whats-the-mean-of-fov-hfov-vfov-and-dfov-for-security-cameras/.

Authors, The Kivy. “Welcome to Kivy¶.” *Kivy*, 2022, kivy.org/doc/stable/.

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

-Learn, Sci-Kit. “Learn.” *Scikit*, 2011, scikit-learn.org/stable/.

Foundation, Raspberry-Pi. “Raspberry Pi Documentation.” *Camera Software*, 2012, www.raspberrypi.com/documentation/computers/camera_software.html#libcamera-vid-2.

Developers, Google. “API Documentation : Tensorflow V2.12.0.” *TensorFlow*, 2021, www.tensorflow.org/api_docs.

Hallard, Charles. “Hallard/Radiohead: Radiohead Packet Radio Library for Embedded Microprocessors.” *GitHub*, github.com/hallard/RadioHead. Accessed 10 May 2023.

Géron, Aurélien. *Hands-on Machine Learning with Scikit-Learn, Keras and Tensorflow: Concepts, Tools and Techniques to Build Intelligent Systems*. O’Reilly, 2020.

Faruque, Sarker M O, and Sam Washington. *Learning Python Network Programming: Utilize Python 3 to Get Network Applications up and Running Quickly and Easily*. Packt Publishing Limited, 2015.