

**University of Central Florida  
Department of Electrical & Computer Engineering**



**UNIVERSITY OF  
CENTRAL FLORIDA**

**Smart Parking System**

**EEL 4915 | Senior Design 2 | Fall 2022 | Group B**

**Senior Design 2 Final Documentation**

Oscar Acuna  
Computer Engineering  
oacuna@knights.ucf.edu

Jordan Johnson  
Electrical Engineering  
jordan614407@knights.ucf.edu

M. Ridwan  
Computer Engineering  
mridwan@knights.ucf.edu

Kyle Carpenter  
Computer Engineering  
kylecarpenter@knights.ucf.edu

# Table of Contents

<b>List of Figures</b>	<b>7</b>
<b>List of Tables</b>	<b>9</b>
<b>1.0 Executive Summary</b>	<b>1</b>
<b>2.0 Project Description</b>	<b>2</b>
2.1 Project Motivation	2
2.2 Goals and Objectives	2
2.3 Function of Project	3
2.4 Project Block Diagrams	3
<b>3.0 Project Requirement Specifications, Constraints, and House of Quality</b>	<b>5</b>
3.1 Requirement Specifications	5
3.1.1 General	5
3.1.2 Hardware	5
3.1.3 Web Application and User Interface Integration	6
3.1.4 OpenCV	6
3.1.5 Wifi Connectivity	6
3.1.6 Power	6
3.2 Constraints	6
3.2.1 Standards	7
3.3 House of Quality	7
<b>4.0 Existing Smart Parking Systems</b>	<b>8</b>
4.1 Existing implementation of Intelligent Parking Systems	8
4.1.1 Indect	9
4.1.2 ParkEagle	10
4.1.3 CleverCiti	11
4.1.4 Smart Parking Limited	12
<b>5.0 Technology Research</b>	<b>14</b>
<b>5.1 Computer Vision</b>	<b>14</b>
5.1.1 OpenCV	15
5.1.2 Computer Vision Techniques	15
5.1.2.1 Edge Detection	15
5.1.2.2 Hough Transform	16
5.1.2.3 Regions of Interest	17
5.1.2.4 Object Detection	18
5.1.2.4.1 Cascade Classifiers	18

5.1.2.4.2 You Only Look Once (YOLO)	18
5.1.2.4.3 Object Detection Method Comparison	19
<b>5.2 Cameras</b>	<b>20</b>
5.2.1 OpenCV AI Kit (OAK) Cameras	21
5.2.1.1 OAK-1 PoE	21
5.2.1.2 OAK-D PoE	21
5.2.1.3 OAK-D Pro PoE	21
5.2.2 IP Cameras	22
5.2.3 Camera Comparison	22
5.2.4 Final Camera Selection	23
5.2.4.1 Movidius Myriad X VPU	24
5.2.4.2 DepthAI	24
5.2.4.3 OAK-1 PoE Electrical Characteristics	26
5.2.4.4 OAK-1 PoE Mechanical Information	27
<b>5.3 Microcontrollers</b>	<b>28</b>
5.3.1 Atmel (Microchip)	29
5.3.1.1 ATSAM4E8CA-AN	29
5.3.1.2 AT32UC3A1128-AUT	29
5.3.2 Microchip	29
5.3.2.1 ATSAME70J19A-AN	29
5.3.2.2 PIC32MX664F064L-I/PF	29
5.3.3 Infineon XMC4504F100F512ACXQMA1	30
5.3.4 Comparison Chart	30
5.3.5 Microcontroller Final Selection	30
<b>5.4 LEDs</b>	<b>31</b>
5.4.1 LED Options	31
5.4.2 LED Selection	31
5.5 Local Server	33
5.5.1 Odyssey X86J4125864	33
5.5.2 Raspberry Pi 4	34
5.5.3 UDOO X86 II	35
5.5.4 Local Server Choice	36
<b>5.6 Ethernet PoE Switch and Local Network Internet Access</b>	<b>37</b>
5.6.1 PoE Switches	37
5.6.1.1 PoE Switch Choice	38
5.6.2 Local Network Internet Access	38
5.6.2.1 Cellular Modem	38
5.6.2.2 Smartphone's Hotspot and Tethering.	39

5.6.2.3 Wireless Router	39
5.6.2.4 Internet Access Choice	40
<b>5.7 Web Application Research</b>	<b>40</b>
5.7.1 Web Application Types	41
5.7.1.1 Static Web Applications	41
5.7.1.2 Dynamic Web Applications	41
5.7.1.3 Single Page Web Applications	41
5.7.1.4 Multi-Page Web Applications	42
5.7.1.5 Animated Web Applications	42
5.7.1.6 E-Commerce Web Applications	42
5.7.1.7 Portal Web Applications	43
5.7.1.8 Rich Internet Applications	43
5.7.1.9 Progressive Web Applications	43
5.7.1.10 Decided Web Application	44
5.7.2 Web Development Stacks	44
5.7.2.1 LAMP Stack	44
5.7.2.2 MEAN Stack	45
5.7.2.3 MERN Stack	45
5.7.2.4 MEVN Stack	46
5.7.2.5 Python - Django Stack	46
5.7.2.6 Ruby on Rails Stack	46
5.7.2.7 Decided Web Stack	46
<b>5.8 Mobile Application Research</b>	<b>48</b>
5.8.1 Mobile Application Types	48
5.8.1.1 Native Apps	48
5.8.1.2 Web-Based Apps	48
5.8.1.3 Hybrid Apps	48
5.8.1.4 Mobile App Choice	49
5.8.2 Cross-Platform (Android/iOS) App Development Framework Options	49
5.8.2.1 Ionic	49
5.8.2.2 Flutter	49
5.8.2.3 React Native	50
5.8.2.4 Xamarin	50
5.8.2.5 Cross-Platform App Development Framework Choice	50
5.9 Web Server Research	51
5.9.1 PaaS vs. IaaS	51
5.9.2 Web Server and Database Hosting Providers	52
5.9.2.1 Digital Ocean	52

5.9.2.2 Heroku	52
5.9.2.3 MongoDB Atlas	53
5.9.2.4 Microsoft Azure App Services	53
5.9.2.5 Google Cloud Platform	53
5.9.2.6 Amazon Web Services (AWS)	54
5.9.2.7 Web Server and Database Hosting Provider Summary	54
5.9.2.8 Web Server Hosting Provider Selection	55
<b>6.0 Related Standards</b>	<b>56</b>
6.1 OSHA Standards	56
6.2 Data Communication Standards	57
6.2.1 Ethernet Standards	57
6.3 Programming Standards	59
6.3.1 Programming Naming Standards	60
6.3.2 Programming Syntax Standards	60
6.3.3 Programming Indentation and Bracing Standards	61
6.4 Ingress Protection Code (IP Rating)	61
6.5 Voltage and Power testing Standards	62
6.6 NEMA Ratings for Enclosure Standards	64
6.7 CPSC Standards	65
6.8 Soldering Standards	66
<b>7.0 Design Constraints</b>	<b>69</b>
7.1 Economic Constraints	69
7.2 Environmental Constraints	69
7.3 Social Constraints	70
7.4 Political Constraints	70
7.5 Ethical Constraints	70
7.6 Health and Safety Constraints	70
7.7 Manufacturability Constraints	71
7.8 Sustainability Constraints	71
7.9 Time Constraints	72
7.10 Testing and Presentation Constraints	72
<b>8.0 System Design</b>	<b>73</b>
8.1 Computer Vision System Design	73
8.1.1 Computer Vision System Overview	73
8.1.2 Software Tools	73
8.1.3 Software Design	74
8.1.4 Software Flowchart	75

8.1.5 Hardware Design	76
8.2 LED Display System Design	78
8.2.1 Display Images	78
8.2.2 Hardware	80
8.2.3 Software	81
8.3 Mobile App Design	84
8.3.1 Mobile App Block Diagram	84
8.3.2 Mobile App User Interface Design	85
8.4 Web App Design	85
8.4.1 Web App Use Case Diagram	86
8.4.2 Database Entity Relationship Diagram (ERD)	86
8.4.3 Web App User Interface Design	87
8.5 Control Unit Design	90
8.5.1 Control Unit's Hardware	90
8.5.2 Control Unit's Software	91
8.6 PCB Components	94
8.6.1 Ethernet Components	94
8.6.1.1 RJ45 Ethernet Port	94
8.6.1.2 Ethernet PHY	95
8.6.1.3 External Clock Source	96
8.6.2 MCU LED Interface	97
8.6.3 Step-Down Voltage Converter Circuit	97
8.6.4 IDC Connectors	98
8.6.5 Push Button	98
<b>9.0 Prototyping</b>	<b>99</b>
9.1 PCB Schematic Capture	99
9.2 Bill of Materials	102
<b>10.0 Testing</b>	<b>104</b>
10.1 Hardware Testing	104
10.1.1 Computer Vision System Hardware Testing	104
10.1.2 Microcontroller Hardware Testing	105
10.1.3 PCB Hardware Testing	107
10.1.3.1 Ethernet Port and Cable Testing	107
10.1.3.2 Ethernet PHY Testing	107
10.1.3.3 Step-Down Voltage Converter Circuit Testing	108
10.2 Software Testing	108
10.2.1 Computer Vision System Software Testing	108

10.2.2 Local Server Software Testing	110
10.2.3 Microcontroller Software Testing	111
10.2.4 Web App Testing	111
10.2.5 Mobile App Testing	113
<b>11.0 Mounting and Installation Procedure</b>	<b>116</b>
<b>12.0 Project Operation</b>	<b>119</b>
12.1 Camera and Server	119
12.2 PCB and LED Display	119
<b>13.0 Project Budgeting and Financing</b>	<b>119</b>
<b>14.0 Project Milestones for Each Semester</b>	<b>121</b>
14.1 Semester 1 (Senior Design 1)	121
14.2 Semester 2 (Senior Design 2)	122
<b>15.0 Project Management</b>	<b>123</b>
<b>16.0 Conclusion</b>	<b>124</b>
<b>References</b>	<b>125</b>

## List of Figures

<b>Figure 1:</b> LED Display - Software Overview.....	12
<b>Figure 2:</b> Hardware Block Diagram Overview.....	13
<b>Figure 3:</b> Web and Mobile App - Software Overview.....	13
<b>Figure 4:</b> Parking Space Detection Diagram - Software Overview.....	14
<b>Figure 5:</b> Example of Canny Edge Detector.....	25
<b>Figure 6:</b> Hough Transform on a Parking Lot.....	26
<b>Figure 7:</b> Cascade Classifier for Face Detection using Eyes as Feature.....	27
<b>Figure 8:</b> Example of YOLO Object Detection .....	28
<b>Figure 9:</b> Functional Block of Belago 1.1.....	31
<b>Figure 10:</b> OAK-1 PoE .....	33
<b>Figure 11:</b> High-Level DepthAI Software Architecture.....	34
<b>Figure 12:</b> OAK-1 PoE Mechanical Measurements.....	36
<b>Figure 13:</b> RGB LED Matrix Panel - 32x64 .....	41
<b>Figure 14:</b> PoE Connection Standards .....	68
<b>Figure 15:</b> Acceptable vs Rework Needed Soldering Process .....	76
<b>Figure 16:</b> Acceptable vs Not Acceptable Part Mount .....	77
<b>Figure 17:</b> Flowchart of the Computer Vision System Workflow .....	85
<b>Figure 18:</b> Basic Block Diagram of the OAK-1 PoE Hardware.....	86
<b>Figure 19:</b> LED Example Images (Double Digits) .....	87
<b>Figure 20:</b> LED Example Images (Single Digit) .....	88
<b>Figure 21:</b> LED Display IDC Connection .....	89
<b>Figure 22:</b> Corner Alley Example .....	90
<b>Figure 23:</b> LED Display Program Flow .....	91
<b>Figure 24:</b> Mobile App Block Diagram .....	92
<b>Figure 25:</b> Mobile App GUI Prototype.....	93
<b>Figure 26:</b> Web App User Case Diagram .....	94
<b>Figure 27:</b> Database Entity Relationship Diagram (ERD) .....	95
<b>Figure 28:</b> Front Page Design .....	96
<b>Figure 29:</b> Video Fee Page Design .....	96
<b>Figure 30:</b> User Administration Page Design .....	97
<b>Figure 31:</b> Parking Administration Page Design .....	97
<b>Figure 32:</b> Parking System Control Unit .....	98
<b>Figure 33:</b> Local Server Java Program GUI Design .....	100
<b>Figure 34:</b> Camera Text File Data Format .....	100
<b>Figure 35:</b> Local Server MySQL Database Design .....	101
<b>Figure 36:</b> Simple Schematic of Interface between Ethernet Cable and MCU .....	102
<b>Figure 37:</b> Simple Schematic of LEDs on RJ45 Ethernet Port .....	103
<b>Figure 38:</b> Simplified Schematic for the TPS563201 .....	105
<b>Figure 39.1:</b> PCB Schematics .....	108
<b>Figure 39.2:</b> PCB Schematics .....	109
<b>Figure 40:</b> Luxonis documentation example result of depthai_demo.py running .....	110
<b>Figure 41:</b> Point of View from Camera.....	122
<b>Figure 42:</b> Secondary Point of View from Camera .....	122



## List of Tables

<b>Table 1:</b> House of Quality .....	16
<b>Table 2:</b> Engineering Trade Off Matrix.....	17
<b>Table 3:</b> Object Detection Method Comparison .....	29
<b>Table 4:</b> Camera Comparison.....	32
<b>Table 5:</b> DepthAI SDK Classes and Functions .....	35
<b>Table 6:</b> Absolute Maximum Ratings of the OAK-1 PoE.....	35
<b>Table 7:</b> Recommended Operating Conditions of the OAK-1 PoE.....	36
<b>Table 8:</b> Microcontroller Comparison.....	39
<b>Table 9:</b> Specifications for Odyssey Mini PC.....	42
<b>Table 10:</b> Raspberry Pi 4 Specifications.....	43
<b>Table 11:</b> Specifications for UDOO X86 II.....	44
<b>Table 12:</b> Summary of PoE Switches.....	46
<b>Table 13:</b> Summary of Internet Solutions.....	48
<b>Table 14:</b> Web Server and Database hosting plan and Pricing.....	63
<b>Table 15:</b> Data rate based on IEEE 802.11.....	66
<b>Table 16:</b> Ethernet Wiring Standards.....	67
<b>Table 17:</b> IP Code First Digit Meaning .....	70
<b>Table 18:</b> IP Code Second Digit Meaning .....	71
<b>Table 19:</b> Software Development Tools .....	83
<b>Table 20:</b> Ethernet PHY Port Interface/MCU MAC Interface Pins .....	104
<b>Table 21:</b> Ethernet PHY Clock Interface Pins .....	105
<b>Table 22:</b> MCU LED Interface Pins .....	106
<b>Table 23:</b> PCB Bill of Materials .....	109
<b>Table 24:</b> Budget Breakdown .....	125
<b>Table 25:</b> Senior Design 1 Milestones .....	126
<b>Table 26:</b> Senior Design 2 Milestone .....	127

## 1.0 Executive Summary

With the large number of students attending the University of Central Florida, parking congestion is a recurring issue that has caused a variety of problems for students and faculty. The current methods of dealing with this problem, an LED sign outside of each garage indicating “open” or “full” and a website that displays the percentage of open parking spots in each garage, are not reliable, and they are simply not enough to handle the volume of people that the parking garages at UCF endure. In this document, we suggest a different way of handling this problem by introducing a smart parking system that quickly communicates to students where open parking spots are within a garage using small LED signs and computer vision.

This project takes a unique approach compared to available parking management systems. We do not use proximity sensors to monitor individual parking spots but computer vision to monitor cars entering and leaving each parking row with a parking garage. The combination of needing a large number of sensors to monitor every parking spot, power supply requirements for each sensor, complexities relating to installation, and maintenance requirements for all of the sensors makes for a solution that could become quite costly, especially when looking at large scale applications such as a UCF parking garage. Instead, we use cameras and computer vision to detect how many cars are entering a parking row and subtract it from the number of available spaces in the row. This information is then relayed to an LED sign at the end of a row of parking spots which indicates the exact number of open spots within that row.

While our group would like the opportunity to design a parking management system for an entire garage at UCF, we had time and budget constraints to adhere to. Therefore, we designed a system just as a proof of concept that could be scaled to fit a garage of virtually any size. This is an essential factor of our project, as the entire motivation for doing it comes from UCF’s parking garage issues. With this said, we market this project as an efficient, accurate, and cost-effective solution for dealing with congested parking garages.

The physical deliverables for our project include a camera, an ethernet switch, a local server, a custom printed circuit board (PCB), and an LED sign. The camera is a power-over-ethernet (PoE) device embedded with OpenCV, which detects the entering and leaving cars and updates a database running on the server. The update includes a -1 for every vehicle that enters and a +1 for every vehicle that leaves. The server computes the total available spaces on that parking row. Then, it sends the new number of available spaces to the PCB via ethernet, which updates the number on an LED sign driven by a microcontroller on the PCB.

Initially, the design included a web and a mobile app to receive the data from the server. These apps would provide parking information, including the number of available spots and where they are located, data analytics on the best times to find parking, and how long certain spots have been occupied. Unfortunately, due to time constraints, the web app, mobile app, and analytics were scrapped from the project.

## **2.0 Project Description**

### **2.1 Project Motivation**

While the number of students that attend UCF's campus increases every year, the number of parking garages around campus does not. With a growing student body, naturally, there is an increase in the number of vehicles flowing in and out of campus throughout the day. During times of the day when there is a lot of overlap between class sessions that are either starting or concluding, a large number of students and faculty are entering and exiting the garages around campus, which creates a bottleneck problem. This causes the garages to become quite congested as people navigate them. This congestion inside of the garages leads to several issues, including long lines inside and outside of parking garages, backed-up traffic around the perimeter of the campus, and late arrivals to class due to difficulties finding a parking spot.

Our group decided that UCF's current solution to these issues is ineffective. A website that indicates the percentage of open parking spots inside each garage, along with signs outside of the garages indicating whether it is open or full, is simply not enough to ease the large flow of traffic that UCF's parking garages endure. Therefore, we developed a smart parking system to mitigate the above mentioned problems.

### **2.2 Goals and Objectives**

UCF is a growing school with more than 70,406 students currently enrolled. With such a large number of students in attendance in addition to faculty, this leads to the parking garage issues that students and the faculty face every year.

With our project, the goal was to aid the UCF population by having a budget-friendly smart parking system that reduces the time it takes to get in the garage, the time it takes to get out of the garage, and the time it takes to find a parking spot. The objectives we had to help reach this goal are listed below:

- Use video cameras embedded with computer vision computation to detect entering and leaving cars from a parking row.
- Develop a website and mobile application where UCF students and faculty can get a detailed description of the available parking spots promptly.
- Use LED signs to communicate open parking spot locations for people navigating the garage.
- Develop our system in a way such that power consumption is low.
- Develop an ethernet-wired network for transmitting data between the different components in our project.

With a successful implementation of our smart parking system, we hoped to conquer the problem of crowded parking garages with great results. Our smart parking system allows UCF's students and faculty to visualize and access parking data more efficiently without wasting their valuable time and energy focusing on parking issues anymore. After our

project, we aimed to successfully build a realistic, budget-friendly smart parking system that uses low power consumption. Still, unfortunately, the ethernet connection between the PCB and the server was not completed. The ethernet programming of the microcontroller proved to be more difficult than we initially anticipated, and thus, we ran out of time, and it could not be completed.

### 2.3 Function of Project

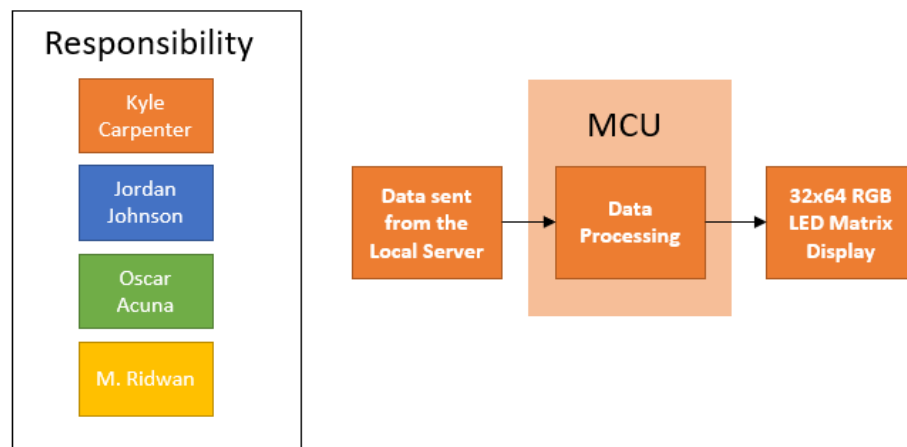
The smart parking system should be capable of recognizing all vehicles entering and exiting the parking garages on campus and making real-time analysis of available parking spots to direct vehicles to new open locations. Using a camera system and computer vision, the vehicles are tracked through portions of the garage. A notification system built into the garage composed of LED signs that indicate open parking spots and how many are available on a specific section or level is the primary guidance for the drivers.

Depending on whether stretch goals are achieved, the LED could vary in complexity. One of the target capabilities is showing available parking spot counts at each level of the garage, with a final stretch goal of having a complex LED system that guides drivers to individual parking spots. The LEDs point to specific parking spots and make for a very unambiguous directing system.

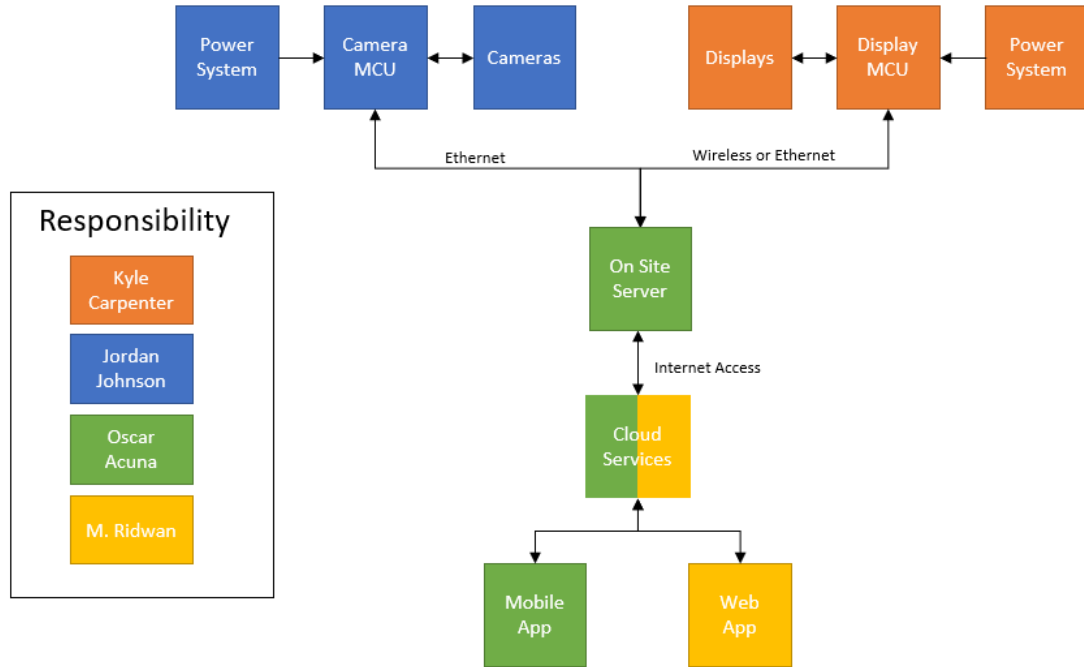
In addition to physical signage in the garage, there were plans to also have a mobile app component for parking status updates. The UCF parking app would have been overhauled to feature higher accuracy of how many spots are available and additional details, like the rate of vehicles entering the garage and which levels are full.

### 2.4 Project Block Diagrams

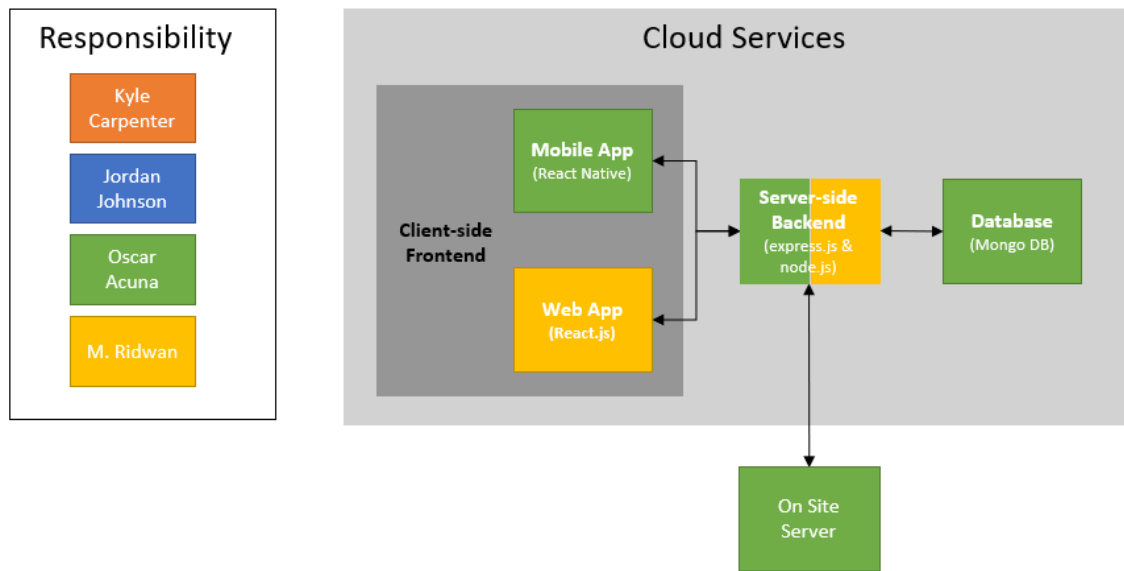
The preliminary project system is summarized in the block diagrams shown in Figures 1 through 4.



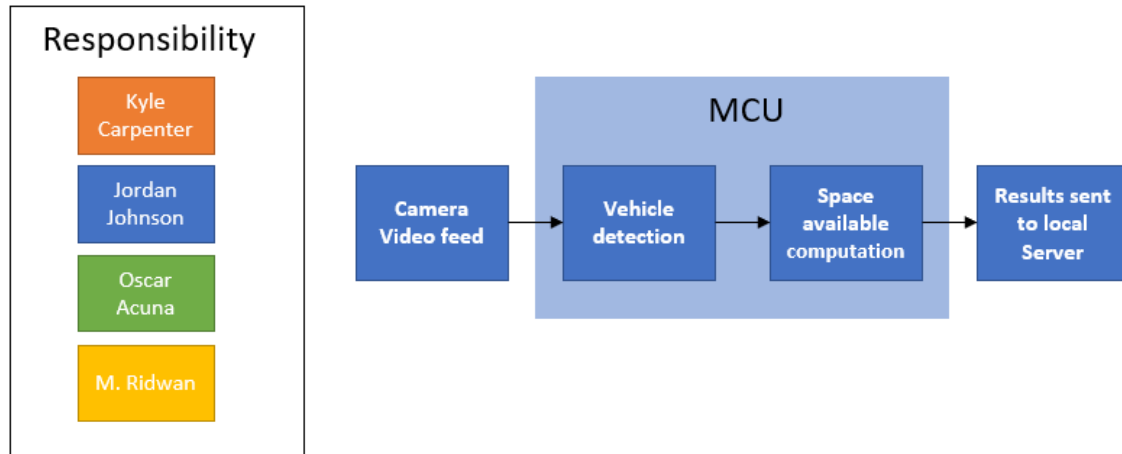
**Figure 1:** LED Display - Software Overview



**Figure 2: Hardware Block Diagram Overview**



**Figure 3: Web and Mobile App - Software Overview**



**Figure 4:** Parking Space Detection Diagram - Software Overview

## 3.0 Project Requirement Specifications, Constraints, and House of Quality

### 3.1 Requirement Specifications

#### 3.1.1 General

- Our system should be able to monitor at least 15 parking spots.
- The system must detect available spaces using OpenCV in less than 2 seconds.
- The system must inform drivers of the number of available parking spots using an LED sign
- The system should be hoisted a minimum of 14 feet above the ground for more accurate video capture.
- Our system should be 100% accurate within 60 seconds of a change in # of open spots
- Our system should be 90% accurate within 30 seconds of a change in # of open spots
- The above specifications were changed by the end of senior design 2. The camera now detects cars entering and leaving a parking row instead of monitoring individual spots. The system accuracy goal was changed to 95% within 10 seconds. The project's scalability was designed with 100 cameras and 100 LED signs in mind.

#### 3.1.2 Hardware

- LED signs must have at least a 1,000 nit rating.
- LED signs must be at least 24 inches by 32 inches. A 5-inch by 10-inch sign later proved to be enough.
- A Microcontroller should have Wi-Fi and a communication module to drive the LED signs and transfer data. Wi-Fi was later scrapped from the project.
- Cameras must be OpenCV compatible.

- Cameras must be able to see the required number of parking spots at a height of 14 feet. This requirement was later changed to whatever height the garage's ceiling allowed.

### 3.1.3 Web Application and User Interface Integration

- Easily maneuverable Graphics interface to help end-user navigate and connect with the implemented video transmission technology.
- Stream real-time video footage of parking spots for better navigation aid.
- Display the total number of vacant parking spots within the designated area.
- Allow end-users to be able to reserve parking spots.

### 3.1.4 OpenCV

- OpenCV must detect when a parking spot is free or occupied.
- Train OpenCV over time to differentiate between vacant and non-vacant parking spots successfully.
- GIS must be implemented to monitor the parking spot outlines and keep a count of the total parking spots.

### 3.1.5 Wifi Connectivity

- Cameras, PCB, and local server will all need to connect to a local network via Wi-Fi with a speed of at least 5 Mbps using the 2.4GHz frequency.
- A Wi-Fi router/access point will provide a local network for the cameras, microcontrollers, custom PCBs, and LED signs to connect to the local server. It should provide at least 10 Mbps upload/download speed in the 2.4GHz frequency.
- An ethernet switch with PoE-enabled ports or PoE injectors will be implemented to provide power to the cameras.

### 3.1.6 Power

- Cameras must be powered by power-over-ethernet (PoE)
- PCB must be battery powered. This requirement was later changed to "PCB to be powered by PoE."

## 3.2 Constraints

With less than a year to deliver this project, time is the primary constraint that our group must adhere to. To ensure we are making considerable progress in a reasonable amount of time, we have created a list of project milestones we will follow over the next year. Additionally, since our project consists of expensive components, we must ensure that our budget does not get out of control. Our group has agreed that we will spend no more than \$2,000, and an initial budget is shown in section 4.0. Additional constraints that we must adhere to for our system to work correctly are listed below.

- The system must be able to work in the daytime and nighttime. For nighttime, the camera must be able to work with at least 0.001 lux.

- The local Server must be able to be accessed remotely. If Windows OS or a Linux OS is used, software such as TeamViewer allows controlling the server remotely.
- The site must provide Internet access of at least 5Mbps.

### 3.2.1 Standards

- All electronics must be enclosed in cases designed following the IP65 standard to protect the equipment from dust and water.
- Cameras will be powered using Power over Ethernet standard 802.3af.
- Wi-Fi network must be standard 802.11ax to provide 2.4GHz network connectivity.
- The site must provide an energy source of 120V/60Hz
- Network communication of cameras and LED signs should use TCP/IP standards.

### 3.3 House of Quality

The house of quality, shown in Table 1, shows the positive and negative correlations between engineering and marketing requirements. In addition, the table helped in determining the engineering-marketing trade-off. In Table 2, the engineering trade-offs are shown, which indicates how one category affects others.

**Table 1:** House of Quality

			Engineering Requirements				
			Power	Waterproof	Day/Night Functionality	Accuracy	Cost
			-	+	+	+	-
Marketing Requirements	Speed	+	↑			↓	↓
	Outdoor Functionality	+	↓	↑↑	↑↑	↑	↓↓
	Visual Indicators	+			↑		↓
	Area Coverage	+	↓		↓	↑	↓↓
	Mobile/Web Application	+				↑	↓
	Cost	-	↑	↓	↓↓	↓	↑↑
Target Engineering Requirements				≥ IP65	≤ 0.001 lux	90% in 30s, 100% in 60s	≤ \$2000



**Table 2: Engineering Trade-Off Matrix**

		Power	Waterproof	Day/Night Functionality	Accuracy	Cost
		-	+	+	+	-
Power	-			↓	↓	↑
Waterproof	+					↓
Day/Night Functionality	+					↓
Accuracy	+					↓
Cost	-					

## 4.0 Existing Smart Parking Systems

Over the years, many companies have implemented different solutions to solve the global parking issue. Each of the implementations has its own innovative and optimized solutions to aid the population and introduce the intelligent parking concept in day-to-day lives. These solutions have steered the project's motivation to improve some of the existing solutions to improve parking problems. This section will cover the existing systems to solve the growing parking congestion issue.

### 4.1 Existing implementation of Intelligent Parking Systems

The project we chose to pursue gave us quite a bit of freedom with the type of design we could develop with the fact that there were many ways to go about making parking garages more efficient. The team decided to use cameras and computer vision techniques in our design, though it is common for existing smart parking systems to use video cameras, various sensors, or a combination of both. With that said, some existing smart parking systems motivated our project. Brainstorming ideas gathered from existing solutions and improving upon those ideas and implementations is a great way of solving global problems. The majority of the big companies bounce ideas off of each other and create better solutions to support the previous implementations. Following the same strategy, for this Smart Parking project, a couple of other companies and their innovations were utilized. Some of their system providers have already created ample opportunities for growth within their systems. Indect's parking system allows camera sensors, state of the art casing that would withstand any natural calamities. Cleverciti has taken customer experience to another step with a subscription-based system. Companies like Parkeagle helped the parking garages and corporate building parking solutions and took the initiative to handle large implementations of urban city parking issues. Then companies like Smart Parking Limited, with their globally scaled solutions, have incorporated the

usage of cloud-based platforms along with customizable applications to cater to the audiences. The immense range of innovative solutions has paved the way for further improvement of parking dilemmas and the implementation of smart city concepts with technological advancements.

#### 4.1.1 Indect

One of the biggest companies in the United States, Disney, have implemented the concept of Smart Parking in one of their newer addition of theme parks. Disney Springs, at the heart of Orlando, has become one of the most prominent tourist spots in Central Florida. So to tackle the parking problem for the massive number of people visiting the area daily, Disney Springs has adopted the new smart parking system provided by Indect. Indect achieves the solution to the parking issue by monitoring every parking spot in the garage and indicating their status, whether empty or not. This status is shown on top of each parking spot with a LED light. The LED light turns green if the parking spot is vacant, and in the case of the spot being occupied, the LED light turns Red. In addition, Indect provides endless level-counting and wayfinding solutions for Disney Springs by integrating LED signs at different locations (i.e., entrance, exit, and turns) inside the garage for convenience. It also allows license plate recognition and locating the whereabouts of a car as a feature if someone mistakenly forgets the location of the parked vehicle.

Indect's Single Space Ultrasonic sensors provide the ability to integrate RGB color (red, green, blue) LEDs through parking spaces to be visible to consumers. These also have the ability of IVIS interface, which immensely helps with Day/Night lighting modules to reserve energy. They also allowed space to be customizable to different colors depending on the need. Since the embedded light system communicates with a framework via a wireless network, a simple change from a computer can trigger any necessary changes in the light's colors or mode within a matter of seconds. The Ultrasonic Mini Sensor is EMC and CE certified and comes with ultrasonic transducers for better size efficiency. With the ultra-flat design, this module also has an ultrasonic sound sensor for wayfinding capabilities that are inaudible to humans. One more variant of the Ultrasonic Diagonal Sensor uses sound sensors to decide parking spot availability. It achieves this feat with the emission of ultrasonic waves and echoes to calculate the distance to receive data and make decisions based on that. This version is also CE and UMC-certified.

Indect's camera-based Sensors are the most sophisticated and state-of-the-art implementation of smart parking. This system has been fully automated and does not wait or require manual human inputs for any of its functionalities. This line of camera-based solutions is called UPSOLUT, and one of these sensors has the capability of detecting up to six parking spots at a time. Their cameras are also waterproof and dust resistant to keep up with weather calamities. The IP67 sensor, the only implementation of such sensors in commercial production, allows the UPSOLUT to even function submerged in water and airtight sealing, making the maintenance of the sensors a very minimum. With video recording and sound alert features, this system allows wayfinding for end-users, which is convenient. This particular line of products also comes with kiosks installed in the parking garages for car-finding activities. Since this particular system comes with

license plate recognition, utilizing the UPSLOUT app, consumers can quickly locate their cars. Any activity in the parking lot gets sensed by the sensors within the cameras that trigger the video recording process, which is later streamed. The fisheye camera allows video capture of up to QXSGA resolution at 25 frames per second. The PTZ support in the video streams makes license detection and recognition possible. The stream is also defaulted to a black-and-white format with 100 seconds max intervals to maximize storage efficiency. The advanced machine-learning algorithms for vehicle detection within the system make UPSOLUT the most accurate camera-based system available.

Other than this, Indect also has a Surface Mounted Sensor embedded product line. This particular product line utilizes induction pads that are embedded with electromagnetic vehicle detection sensors. This enables extra features like vehicle count calculation at the toll booths. These magnetic field vehicle detectors use X, Y, and Z-axis detections of the earth's magnetic field. Each sensor is continually on the lookout for any abnormalities within the components of the magnetic field. Any new data entry within the X, Y, and Z components suggests vehicle movement and makes decisions on parking space detections.

The many different implementations of different sensors, as well as the usage of camera sensors to implement a smart parking environment, makes Indect one of the pioneers in the smart parking industry. The usage of mobile applications to stream video in real-time and the car way-finder feature also sets the company apart from other companies in the innovation aspects. The Smart Parking project implements a lot of theories and technologies that Indect has already utilized more commercially. This infrastructure of technologies can be an immediate motivation for the Smart Parking project to base off of.

#### **4.1.2 ParkEagle**

Parkeagle is one of the up-and-coming companies researching smart parking and broadening its focus to smart city communications. Parkeagle enables its smart parking system and caters the solution to the end-users through a real-time information update through mobile apps. Parkeagle utilizes ultra-low-power smart parking sensors to detect the parking spot status. These sensors communicate via server to display the data calculated on an app and even digital road signs. A lot of cities in Europe are utilizing Parkeagle's smart parking system. Parkeagle also implemented a StreetEagle cloud system to take the data from the sensors as input and send it to a digital server, where algorithms are run to make decisions on parking occupancy, traffic flows, and vehicle types. Later these data are presented in the Park Eagle management system for the end-users. The app also has a geofencing feature that allows the drivers to know traffic flow and parking conditions at a given spot ahead of time. The dashboard is catered to fulfill the needs of parking condition information and traffic data for end-users.

Parkeagle has also created technological advancements in modern parking systems and vehicular communications flow in general. The usage of sensors and their capability of communicating directly with cloud systems can indeed revolutionize the concept of smart parking. Since all the algorithmic computation is being done on the cloud, real-time communication between the cloud and end-user side is much more efficient. Their mobile

parking app and digital parking signs also add to the convenience of the catered audiences. This project's smart parking app could use the cloud methodologies of Parkeagle for better communication between web apps and camera sensors. The machine learning algorithm taking place in the cloud instead of the web server would enable quick run time of real-time video streaming as well as quick data update on the consumer user end.

#### 4.1.3 CleverCiti

CleverCiti adds to the advancements in the parking system with a turn-by-turn parking guide for their end-users. CleverCiti works with municipal companies to reduce local traffic and unnecessary emissions and solve the drivers' current parking issues. They are helping local companies to streamline and boost parking revenues with the advancements in parking technology. They have introduced battery-powered lamppost sensors to monitor traffic and parking availability in city areas. The company is improving the on-street parking situation in urban areas, such as parking lots at shopping centers, corporate offices/campuses, and train stations.

CleverCiti utilizes overhead sensors made in Germany that allows the detection of parked cars and vacant parking spots. However, for smoother communication and to optimize run time, these sensors only send the GPS coordinate data sets to the server of the available spaces only. This helps the system gather relevant data and also maximizes memory storage uses. The sensors on the lamp posts can also support up to four smart sensors and power them around the clock. This aids in smart city deployment with the ability to send traffic congestion data, vehicle mode data, and of course, street-side parking data. These sensors also can communicate via an LTE network as well as the existing power line connected to the posts.

The smart parking system of CleverCiti has a unique feature that accurately guides the drivers to the available parking spots. They have implemented this GPS-like feature only within specific areas of municipal parking areas. This saves a lot of time for the end-users as they know exactly which spot to go towards to attain a vacant parking spot. Along with these, CleverCiti has commercialized a subscription-based mobile app system that allows their customers to purchase or subscribe to their interface and solution set. This way, the permit holder and monthly subscribers get optimum parking notifications and facilities through the app. They also have taken extra steps for their customers to opt-in for a dynamic reservation feature, where customers may look ahead to reserve a vacant parking spot ahead of time and avoid all the hassles.

CleverCiti has taken smart parking to another level by commercializing a subscription-based system. They have implemented and focused on the customer service aspect of parking and emphasize catering their goals and vision to the end users' comfort. The subscription-based system and the dynamic reservation set the system apart from any other smart parking technology. The current smart parking project can implement the innovative features of CleverCiti's mobile app to incorporate further assistance and improve the overall user experience.

#### 4.1.4 Smart Parking Limited

Award-winning company Smart Parking has been a world leader in technological design and advancement in the development and management of parking systems. With their intelligent parking structure, they have improved parking experience in shopping and retail, supermarkets, airports, hospitals, and even universities. The company is emphasizing the concept of a smart city through the development and improvement of smart parking systems. The connection of IoT devices across the city through their SmartCloud system would aid in the global implementation of the intelligent parking concept of the company. SmartCloud, developed by Smart Parking Limited, is a globally scaled, real-time internet of things platform. This cloud platform was deployed in parallel with the deployment of the Google cloud. This platform enables the company to optimize in-house data collection and management.

The company's smart sensors and smart devices communicate directly with this platform. The integration of the SmartCloudAPI enables complete connection with the assets to provide real-time support. Even the firmware support for the smart sensors and remote management has been made possible and widespread with the integration of SmartCloud. With these advantages, this company has revolutionized the parking experience with a robust mobile app called Tessera. This application is developed to aid urban city areas and municipalities efficiently with their parking management. This end-to-end compliance management system incorporates all the advantages provided by the company to provide an efficient solution suited for any parking management system. This app provides dashboards that are supported in real-time to showcase the live visual activity of the parking site in a visual map form.

Google Maps is greatly incorporated within the app, which provides detailed real-time information on traffic tariffs and parking citations for specific areas at any given time. It helps users find vacant bays for parking and offers a feature to give directional support to the users to avoid any parking discomfort. The app goes as far as detecting disabled parking availability, limited-duration parking as well as pay booths for paid parking. Within the app, users can locate the history of their parking experiences and review specific invoices or location experiences. This app also can be modified depending on industry use. The company allows a fully customizable parking app for its client according to their own needs. Certain companies in certain cities may have particular needs that one generic application may not be able to provide. Thus with the implementation of SmartCloud, the company has implemented the opportunity of customizing the mobile app according to consumer needs.

The network gateways implemented within the parking system require multiple sensors across areas. Thus for any parking system, multiple sensor units called SmartSpots are implemented. These individual units are scattered to gather precise data on the parking area status from the sensors embedded within. Some of the SmartSpots use sensors that can communicate with the cloud system in wireless form, which is important in areas where devices do not need to communicate with each other. However, there are also sensors deployed that communicate with each other via fiber optic or ethernet.

Along with all these web-based technologies implemented, there are also real-time supported display signs that the company has issued. These signs are called signage and are used to communicate with the users and show the traffic flow status, available parking spot count, and much more. Using digital network communication, these signs provide live updates with the data gathered directly from SmartCloud. One neat feature these LED signs offer is the directions. According to the number of available spots, these signs provide directional data flashed on the LED. There are multiple uses for these signs. For level-by-level implementation, signage comes with the total vehicle count and the directional data view. Other than these overhead displays, some signs can be mounted on walls to provide support for outdoor parking areas as well as the parking entrances and exits. The color is dynamically customizable by the administrator with different daytime and nighttime modes for energy conservation. The sensors implemented have their customizations as well. Inground sensors have been developed to better communicate with SmartSpot gateways using ANPR. These sensors are water-resistant, and casings are designed to withstand natural calamities. Mainly for outside parking spots, these sensors are utilized. The surface-mount sensors aid in the easy installation aspect. Their sensors work perfectly with exposed sites with not fully supported surfaces. Roof, cable support - where further construction work is impossible, these surface mount sensors work perfectly.

Embedded within the floor, these sensors are designed to provide better visibility with LED settings to support visibility for increased traffic and visual challenges on the surfaces. Lastly, the most popular overhead indicator sensors are used for parking garages with many complex routes for parking. These sensors are supported with high visibility LED lights to make it easy for motorists to locate the desired parking spot. These parking sensor lights can be controlled dynamically according to the business needs of the administrator. The RGB LED colors are utilized for every one of the sensors, which also communicate directly with the SmartCloud API to provide data in real-time to the digital application at the end-users; discretion. Along with all these innovative technologies already implemented, the Automatic Number Plate Recognition feature aids in the police and traffic management facilities. Otherwise known as License Plate Recognition, the technology allows the monitorization of vehicle registration plate recognition features to better secure and manage the car parking infrastructures. For toll agencies and government pay-per-use highways, this ANPR technology has aided. High-capacity cameras with advanced machine learning algorithms incorporating the perspective of lighting conditions and angles allow an extremely accurate ability to scan and decipher each license plate. Depending on mirrored plates and lighting conditions, these camera sensors also use IR optimization to work with different lighting modes, allowing efficient and accurate picture capturing at any time. This technology has been used for paid parking lots as well, where the sensors send the data directly to SmartCloud, where an automated payment script is in charge of payment collection and billing the users based on license number.

Smart Parking Limited indeed provides one of the most developed and robust support for day-to-day parking complexities. With the implementation of the company's cloud platform, this system has the potential to grow and advance vastly along with the tide of

time. The business perspective of this system has a lot of potentials, with the ability to cater to the consumers based on metrics like locations, terrains, and other necessary needs. A global mobile application keeps the implementation of the system up to date and easy to maneuver for the client. The different sensor applications also cater to different needs based on infrastructure and location. License plate recognition brings another realm of possibilities in automated billing opportunities for municipal bus areas and government facilities. Smart Parking Limited is a pioneer in changing the path of parking technological advancements. Even though implementing such a large-scale system is not possible for the current smart parking project, the abundant opportunities for growth and improvement within the intelligent parking business are, without a doubt, a primary motivator for the project.

## **5.0 Technology Research**

In this section, we introduce the potential technologies used in our Smart Parking System. The areas discussed include computer vision, cameras, microcontrollers, LED matrix displays, and servers.

### **5.1 Computer Vision**

Computer vision is an area of artificial intelligence that allows computers and systems to derive meaningful information from images, videos, or other visual inputs and make recommendations based on that information. Ultimately, computer vision aims to give computers and systems the ability to see and perceive the world as humans do. For our project, we use computer vision for our system to accomplish five tasks:

1. Detects the position of all available parking spaces.
2. Identify whether that parking space is open or occupied.
3. Uniquely identify cars by color or license plate.
4. Differentiate between a parking space occupied by a human or a vehicle and react accordingly.
5. Calculate how long a car occupies a spot.

We needed to use a computer vision library to achieve these tasks. While there are a variety of libraries that could meet our needs, including OpenCV, Tensorflow, and SimpleCV, our group has decided to work with OpenCV. This is mainly due to OpenCV having more algorithms to work with than any other library, a large support community that helped us become proficient with this library, and our group already has some experience with using OpenCV from classes we have taken at UCF.

The compute vision tasks were changed to detecting vehicles entering and leaving a parking row within the garage; therefore, it does not identify individual spaces for occupancy. How long a car occupied a spot was scrapped from the final implementation.

### 5.1.1 OpenCV

OpenCV is an open-source computer vision and machine learning software library. It has a broad area of applications and provides thousands of computer vision algorithms relating to image processing, video analysis, object detection, and object identification segmentation, just to name a few. This library was written in C++, and its primary interface is in C++; however, it supports a wide variety of other programming languages, including Java, Python, and Matlab. In the case of Python, this is due to the use of wrappers where the computationally intensive code in C++ is working in the background while the user can use customizable pre-defined functions. For our project, we would like to use Python because it is a general-purpose programming language that is simple, easy to learn, and more readable than other languages.

What separates OpenCV from other computer vision libraries is that it was designed for real-time vision applications. This will be useful for our smart parking system since we will always work with a live dataset rather than a pre-recorded dataset. Additionally, we would like the correct number of open parking spots to be reflected on our LED sign within 30 seconds of a change, and working with a computer vision library that is designed for real-time applications could help out with this.

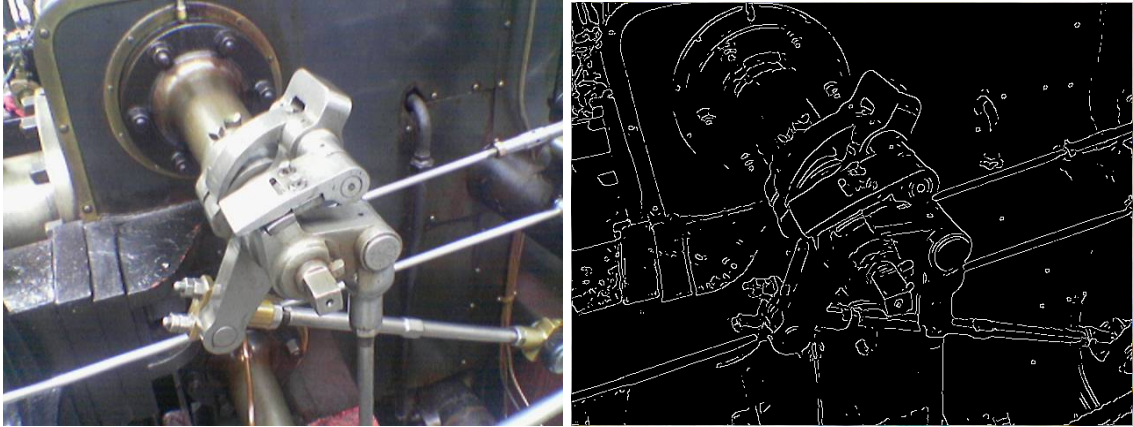
### 5.1.2 Computer Vision Techniques

To achieve the goals described in previous sections, we will need to implement a combination of computer vision techniques. This section will describe the techniques that we could expect to apply in our system.

#### 5.1.2.1 Edge Detection

Edge detection is a computer vision technique used for identifying the points in a digital image where there is a sharp change in image brightness. These points where the image brightness changes sharply are called edges or boundaries. This technique is effective as it can extract useful structural information from an image and drastically reduce the amount of data to be processed. This was likely a technique to be used in our system as we needed our camera to be able to identify what a parking space is from a digital video feed. Since there is a sharp change in brightness between the pavement and the lines painted on the ground to designate a parking spot, these would show up as edges in the image, as shown in Figure 5.





**Figure 5:** Example of Canny Edge Detector [15]

There are a variety of edge detection techniques, including the Prewitt edge detector, Laplacian edge detector, and the Sobel edge detector; however, the most commonly used edge detection technique is the canny edge detector which is shown in the figure above. While this method is more complex than other edge detection methods, it is also the most effective. This technique uses a multi-stage algorithm defined by the six steps below:

1. Convert digital image or video to grayscale
2. Apply a Gaussian filter to smooth the image and reduce noise
3. Find the intensity gradients of the image to help identify edge intensity and direction
4. Apply non-maximum suppression to thin the edges in the image and eliminate false responses to the edge detector
5. Apply a double threshold to determine strong, weak, and irrelevant edges in image
6. Use hysteresis edge tracking to convert weak edges into strong ones only if there is a strong edge close to it

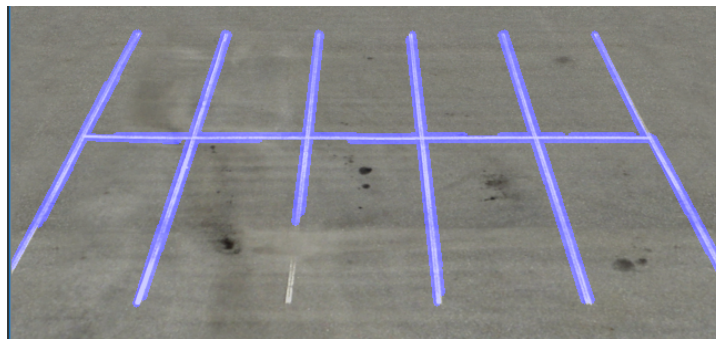
While edge detection is an effective method for extracting useful structural information from an image, there are a couple of drawbacks that should be mentioned. The first is that the size of the output image will be shrunken compared to the input image because the filters used for edge detection shrink the size of the image. Since the size of the image is shrunken, this creates another drawback where there is a loss of a lot of valuable information, especially from the outer perimeters of the image. With this said, there is a technique to combat these drawbacks called image padding, where additional pixels are added to the perimeters of the image to avoid losing valuable information from the input image. In the final product, edge detection was not implemented because a vehicle detection library, which was already developed and trained, was used instead. Therefore, the team did not have to go through the intricacies of applying this technique.

#### 5.1.2.2 Hough Transform

The Hough transform is a feature extraction technique that can detect any shape in an image as long as that shape can be represented in a mathematical form. Since a straight

line can be represented as  $y = mx + b$  in the cartesian coordinate system or as  $\rho = x\cos\theta + y\sin\theta$  in the polar coordinate system, this method can detect the straight lines that define parking spaces. With this being said, the polar coordinate system is primarily used in the Hough transform, as the cartesian coordinate system cannot define vertical lines.

This method is most effective after edge detection has been applied to the image, as the information needed for the Hough transform has already been extracted. The hough transform looks at each edge point defined by the edge detector and then looks at other edge points within the image to see if there is an intersection. Once the hough transform sees that there is a high amount of intersections between a set of edge points, it can conclude that a line can be marked through those points, as shown in Figure 6.



**Figure 6:** Hough Transform on a Parking Lot [17]

This is effective because the output image from an edge detector is still an image described by its pixels; however, after the Hough transform has been applied, we now have an image that is defined by the characteristic equations representing a parking spot, further refining the amount of data to be analyzed from the image. With this said, there is one thing to mention about using this method, being that for it to perform best, we must begin with a set of empty parking spaces. Using parking spaces that are already occupied could create some inaccuracies from the output of the hough transform. In the final product, Hough transform was not implemented by us because a vehicle detection library, which was already developed and trained, was used instead. Therefore, the team did not have to go through the intricacies of applying this technique.

### 5.1.2.3 Regions of Interest

After the lines defining a parking spot have been drawn onto the image by the hough transform, we need to define the specific areas of interest, i.e., each parking spot. One method of doing this is using the intersections between the vertical and horizontal lines defined by the hough transform and the endpoints to extract a list of x and y coordinates. From this list of 2d points, we could map out the corners of each parking spot and group them into sets of 4 such that a unique box is created for every spot.

An alternate method of mapping out the regions of interest would be to use the OpenCV Mouse as a Paint-Brush function. This would allow us to manually draw the bounding boxes for each area of interest. However, our system would require human input before it

can work properly. This means that our system would not be completely autonomous, and we believe it is a method that should only be pursued if the autonomous method described above does not work for us.

#### 5.1.2.4 Object Detection

After we obtain a layout of the parking spots to be monitored, we will then need our system to be able to identify whether a parking spot is occupied or not. This task relates to object detection, and there are quite a few ways we could accomplish this. This section describes a few available methods and some initial thoughts to consider for each, then compares them and presents a final selection. In the final product, none of the object detection techniques were implemented by us because a vehicle detection library, which was already developed and trained, was used instead.

##### 5.1.2.4.1 Cascade Classifiers

Object detection using Haar feature-based cascade classifiers is a machine learning-based approach to this concept. With this method, a cascade function is trained from many positive and negative versions of images and then used to detect objects in other images. Essentially, this method tells OpenCV exactly what features to look for in an image and then makes the decision of whether a certain object exists or not through the use of convolutional matrices. For example, cascade classifiers can be used for face detection as shown in Figure 7 below. The cascade classifier looks for eyes and then uses this information to conclude if there is a face or not.



**Figure 7:** Cascade Classifier for Face Detection using Eyes as Feature [3]

For our case of needing to detect cars in an image, pictures with cars occupying parking spots would be the positive images, and pictures with empty parking spots would be the negative images needed to train the cascade function. The program would then need to extract features from each of these images and see any similarities between images to conclude if a car is in the image or not. With this said, there are already cascade classifiers for car detection. However, we could probably make this method more accurate by taking sample pictures of parking spaces at UCF. While our current method of counting the cars in an image involves detecting a car, haar cascades could also be effectively used for other strategies of counting cars, as explained in Section 10.2.1.

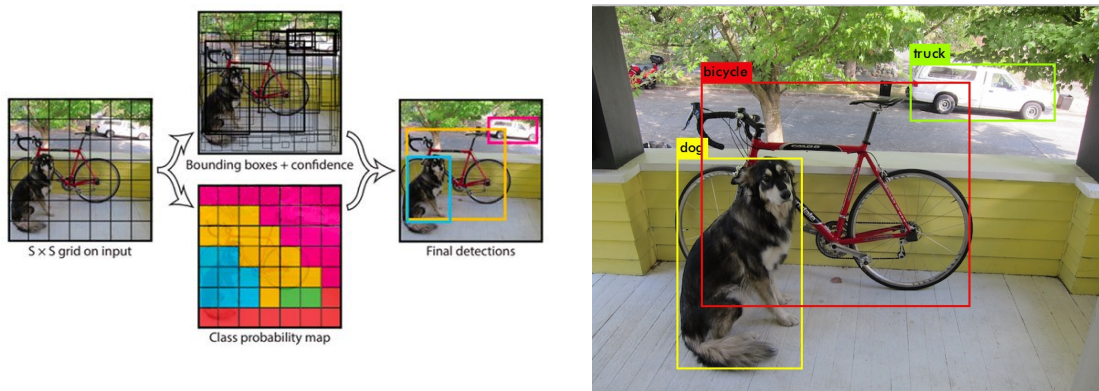
##### 5.1.2.4.2 You Only Look Once (YOLO)

The You Only Look Once (YOLO) algorithm is a deep learning-based approach to real-time object detection. This method differs from other deep learning methods as the YOLO detector can be defined as a one-stage detector. This means that YOLO will

outperform other deep learning methods, such as the Region-Based Convolutional Neural Network (R-CNN), which is a two-stage detector; however, what it gains in speed, it loses in accuracy. Though this is not to say that YOLO is not an accurate algorithm, it is just not as accurate as two-stage detector algorithms.

Functioning as a stage-detector means that YOLO requires an image or video feed to pass through its network only once, hence the name. This is possible because YOLO reasons with an entire image rather than successively examining several regions in an image to detect the objects in it.

This algorithm begins by dividing an image into ‘S’ x ‘S’ cells. It then uses a unique neural network using the characteristics of an entire image or video to predict multiple boxes, with every one containing a specific object. If the center of an object is in one of these cells, this cell will now be responsible for detecting the object. Depending on if an object exists in a cell or not, the algorithm will now assign a score to each cell representing the level of confidence for the object present in the box. If an object does not exist in the cell, the score should be zero. From here, a convolutional neural network based on the GoogLeNet neural network is used to provide a class-specific confidence score for each box. The output of the YOLO detection network will be the original input image or video with localized object detection and their respective classes attached to it, as shown in Figure 8 below.



**Figure 8:** Example of YOLO Object Detection [33]

There is a variation of the YOLO object detector called Tiny-YOLO, which is a smaller version of YOLO; however, it is less accurate than YOLO. We will need to verify just how accurate each of these methods is through testing.

#### 5.1.2.4.3 Object Detection Method Comparison

To summarize the object detection methods described above, Cascade Classifiers are a straightforward method to go about object detection as you simply use convolution matrices to extract the features from an image. This is a simple, quick, and easy method; however, we expect it to not be as accurate as other methods. On the other hand, the YOLO detector is a highly accurate method for object detection; however, implementing

this method would be very complex. A method that could prove to be a bit less complex but still share many similarities with the YOLO method is the Tiny-YOLO method, though this will need to be verified.

After considering each method, we realized that the one that suits us best would depend on our strategy to count the number of available parking spots. If we go with our original method described in previous sections of detecting the car itself, then YOLO or Tiny-YOLO would be the best option for us. If we decide to pursue another strategy to count available parking spots, such as the ones described in section 10.2.1, Haar Cascades may be the more suitable option to fit our needs. We did not know which method best suited us until it was tested. The table below compares the object detection methods we were considering. It is important to mention that this table is not 100% accurate and is only based on speculation from the research we have done. In our final implementation, none of the object detection techniques were implemented by us because a vehicle detection library, which was already developed and trained, was used instead.

**Table 3:** Object Detection Method Comparison

Method	Complexity	Speed	Accuracy	Power Need
Haar Cascades	Low	High	Low	Low
YOLO	High	High	High	High
Tiny-YOLO	Medium	High	Medium	Medium

## 5.2 Cameras

The camera is a vital part of our Smart Parking System because it is the sensing element that is capturing the data to be used by the other components within our system. Our system used one camera mounted on a tripod facing the entrance to a row of parking spaces. The camera was angled such that it had enough room to track cars passing by.

With such a large variety of cameras that could be implemented in our project, our group created a list of specifications that we wanted to see from the camera. First, we expect the camera to have at least an IP65 weatherproof rating, so the system can remain functional in rainy, humid, dusty, and hot conditions. Second, the camera needed to handle lowlight conditions such that the system could still function at night time. Third, we expected the live video feed coming from the camera to be easily extracted to easily showcase how the computer vision was working in real-time. Fourth, we expected the camera to be power-over-ethernet (PoE) compatible to easily transmit data, increase the distance we could wire our camera, and decrease the number of cables needed for our system. Lastly, we expected our camera to have a field of view between 70 and 105 degrees such that the amount of required parking spaces in our system is captured while preserving video quality.

With this said, our group did not want to spend more than \$2,00 on our project and were willing to change some of the specifications for our camera to adhere to this constraint. The types of cameras we decided to consider for our project included the OpenCV AI Kit PoE and IP cameras.

### **5.2.1 OpenCV AI Kit (OAK) Cameras**

The OpenCV AI Kit (OAK) is a family of cameras that embed performant spatial sensing, neural inference, and computer vision functionality. They are driven by Intel's Movidius Myriad X Vision Processing Unit (VPU), which is a type of microprocessor that allows all computations to be done at the camera level, completely offloading the robotics perception of our system to the camera itself. The OAK cameras employ Sony's IMX378 RGB image sensor, which allows for a maximum frame rate of 60 fps at a resolution of 12 megapixels, autofocus, and a display field of view of 81 degrees. Additionally, each camera is fitted with a ¼ - 20 tripod mount on the bottom of the unit, which eases lifting the camera to a high enough level to view the parking spaces.

The OAK camera family is separated into three groups, including the USB line, the PoE line, and the IoT line. We have decided to work with the PoE cameras as using ethernet cords will provide sufficient cable length between the camera, our ethernet switch, and our PCB. Additionally, the PoE cameras come with IP67-rated housing. The PoE line consists of three cameras, including the OAK-1 PoE, OAK-D PoE, and the OAK-D Pro PoE. Each of these cameras is further described below.

#### **5.2.1.1 OAK-1 PoE**

The OAK-1 PoE is the baseline camera of the OAK PoE line. It employs just a single RGB image sensor camera which pipes its video feed directly into the Myriad X VPU and DepthAI for AI processing. Since the OAK-1 PoE utilizes onboard python scripting, it provides a variety of functionalities at the camera level, including object tracking, corner detection, feature tracking, custom computer vision functions, and neural interference. Additionally, if we did want to extract a recorded video or live video stream, this camera also provides H.264, H.265, and MJPEG encoding. With all of these functionalities, the unit price of this camera is \$249.

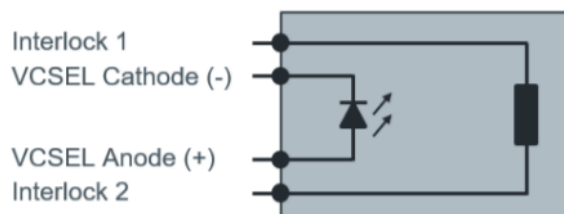
#### **5.2.1.2 OAK-D PoE**

The OAK-D PoE differs from the OAK-1 PoE because it uses three onboard cameras rather than just one. In addition to the single RGB image sensor, it includes two OmniVision OV9282 grayscale image sensors capable of producing video at a max frame rate of 120 FPS with a resolution of 1 MP. Using these two additional cameras gives the OAK-D PoE the capabilities of 3D object localization, 3D object tracking, and depth perception on top of the existing functions with the OAK-1 PoE. The addition of the capabilities brings the unit price of this camera from \$249 with the OAK-1 PoE to \$299.

#### **5.2.1.3 OAK-D Pro PoE**

The OAK-D Pro PoE is essentially the same camera as the OAK-D PoE; however, the OAK-D Pro PoE is designed to handle low-light situations. It accomplishes this through

active infrared illumination in the Belago 1.1 Laser Dot Projector. Referring to Figure 9 below, the functional block of the Belago 1.1 is shown.



**Figure 9:** Functional Block of Belago 1.1 [2]

Vertical cavity surface-emitting lasers (VCSEL) are a type of laser diode where laser beam emission happens perpendicular to the top surface. This laser dot projector illuminates the area in the camera's field of view using 4700 laser dots and acts as a flashlight for the camera. This helps with disparity matching in low-light situations, especially for blank surfaces with little to no texture, such as a wall or a floor. It is also important to mention that this laser dot projector meets the class one specification for lasers meaning that it will cause no harm to the human skin or eyes. In addition to the laser dot projector on this camera, it also uses an infrared LED floodlight to assist in low-light situations. Adding these night vision abilities brings the unit price of the OAK-D PoE from \$299 to \$399.

### 5.2.2 IP Cameras

The second type we considered for our project was an IP camera. While using an IP camera provides a very budget-friendly option, it would also introduce some unique challenges that could have a negative impact on our system.

One of these challenges was that no computer vision computations would happen at the camera level. Since these types of cameras do not include a microprocessor, we would need to extract the live video feed into a Python program via IP address using the OpenCV library and then break the video into individual frames before being able to apply computer vision techniques. Not only would this make our system have to handle large amounts of data being transmitted between the camera and the local server, but there would also run the risk of us having to work with a python program that could become very complicated. Additionally, IP cameras typically only come with two mounting options: a wall mount or a ceiling mount, in contrast to the OAK cameras, which come with a tripod mount. This would have added complexities since we could mount cameras wherever we wanted because we had to test our system on public parking lots and garages. After looking at multiple IP cameras, the two that best fit the needs described in section 7.1 were the ANNKE C800 and the RLC-810A.

### 5.2.3 Camera Comparison

Table 4 shows the criteria we used to compare each of these cameras.

**Table 4: Camera Comparison**

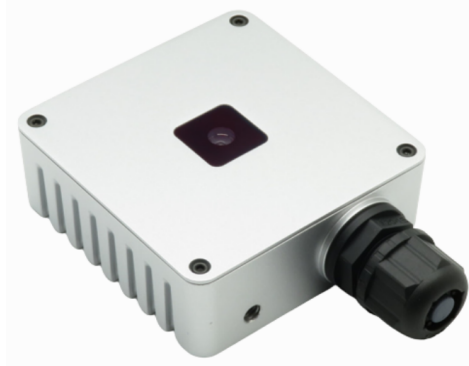
Model	OAK-1 PoE	OAK-D PoE	OAK-D Pro PoE	ANNKE C800	RC-810A
Type	Microprocessor	Microprocessor	Microprocessor	IP Camera	IP Camera
Unit Price	\$249.00	\$299.00	\$399.00	\$79.99	\$84.99
Size (WxHxD)	81.9 mm x 81.9 x 31 mm	130 mm x 65 mm x 29.9 mm	111 mm x 47 mm x 31.1 mm	155 mm x 70 mm	192 mm x 66 mm
Video Resolution	12 MP	12 MP	12 MP	8 MP	8 MP
Speed	60 fps	60 fps	60 fps	30 fps	25 fps
FOV (degrees)	81	81	81	102	87
Power	PoE 802.3af	PoE 802.3af	PoE 802.3af	PoE 802.3af	PoE 802.3af
Designed for Low Light	No	No	Yes	Yes	Yes
Depth Perception	No	Yes	Yes	No	No
Weatherproof Rating	IP67	IP67	IP67	IP67	IP66
Easy Implementation	Yes	Yes	Yes	No	No

### 5.2.4 Final Camera Selection

After considering the cameras analyzed in the sections above, we decided that the OAK-1 PoE was the best camera to use for our project. Although this camera was not designed to work in low light conditions, it still satisfies all of our other requirements while remaining friendly to our \$2,000 budget, as we paid \$249 per camera. In addition, in parking garages where good lighting was present, such as in parking garage C at UCF, the camera performed well. The OAK-D Pro PoE was the only OAK camera designed to function in low-light situations; however, this camera also offers a variety of features that would be overkill for the system we are designing, and we are not willing to spend \$399 per camera.

The OAK-1 PoE, shown in Figure 10, offers full IEEE 802.3af, Class 3 PoE compliance with 1000BASE-T speeds (1gbps) for communication and power. As mentioned in previous sections, this camera is driven by the Movidius Myriad X VPU, and DepthAI is what we will be using to communicate with this VPU and complete all AI and computer vision computations.





**Figure 10:** OAK-1 PoE [1]

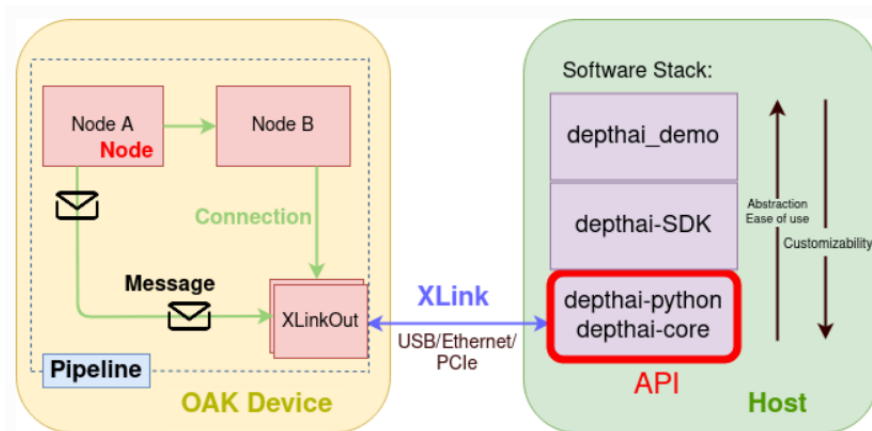
#### 5.2.4.1 Movidius Myriad X VPU

The Movidius Myriad X VPU is the component of the OAK-1 PoE that allows all AI and computer vision computations to be completed from the moment video is captured before any type of data communication happens through ethernet. This VPU contributes heavily to making our system fast as it promises 4 Trillion Operations Per Second (TOPS) processing power (1.4 TOPS for AI applications).

This version of the Myriad X VPU used in the OAK-1 PoE makes use of a new deep neural network developed by Intel called The Neural Compute Engine, which is specifically designed to run deep neural networks at high speed and low power. This was effective for our project as deep neural network inferences are a possible method we could use to detect vehicles and make the decision of whether it is entering or leaving a parking area. With this said, the Myriad X VPU offered many features, including the ability to run any custom-built AI models, video encoding, computer vision, and object tracking. We had access to these features through DepthAI, further explained in the next section.

#### 5.2.4.2 DepthAI

DepthAI is an open-source hardware, software, and AI-training platform built around the Movidius Myriad X VPU. It focuses on the combination of 5 key features which are artificial intelligence, computer vision, depth perception, performance (high resolution and FPS, multiple sensors), and being an embedded, low-power solution. Together, these features allow DepthAI to be a spatial AI + computer vision platform that gives robots and computers the ability to perceive the world as a human can. DepthAI uses a Python Application Programming Interface (API) to connect to, configure, and communicate with the OAK devices within a system, as shown in Figure 11.



**Figure 11:** High-Level DepthAI Software Architecture [1]

The following list highlights the important aspects of Figure 11:

- The **Host Side** is the computer that the OAK device is connected to. This can be a Raspberry Pi, Windows PC, or another compatible computer. Multiple OAK devices can be connected to one host and uniquely identified.
- The **Device Side** is the OAK device itself. If anything happens on this side, it is running on the Movidius Myriad X VPU.
- The **Pipeline** is a complete workflow that consists of nodes and the connections between them. When we received our OAK cameras, we first needed to create this pipeline, populate it with nodes, configure the nodes and the connections between them, and then load it onto the OAK device.
- The **Nodes** are the building blocks of the pipeline. Each node provides a specific functionality, configurable properties, and inputs/outputs. An example of a node would be the EdgeDetector which receives two inputs, an input image, and an input configuration, and then produces one output, the output image. Once a node is created and configured as desired, it can be linked to other nodes.
- The **Connection** is the link between one node's input and another node's output. These connections define the pipeline data flow and establish where messages should be sent to achieve an expected result.
- **Messages** are the communication that happens between linked nodes. Messages sent between linked nodes are the only way they can communicate with one another.
- **XLink** is a middleware capable of exchanging data between an OAK device and the host. XLinkIn sends data from the host to the OAK device, and XLinkOut sends data from the OAK device to the host.

It is also important to note the DepthAI Software Development Kit (SDK), built on top of the Python API library. The DepthAI SDK is a Python package containing classes and functions that help in performing common tasks while using the Python API. The

package primarily consists of managers which handle different aspects of development, as shown in Table 5.

**Table 5:** DepthAI SDK Classes and Functions

Classes and Functions	Description
depthai_sdk.managers.PipelineManager	Helps in setting up processing pipeline
depthai_sdk.managers.NNetManager	Helps in setting up neural networks
depthai_sdk.managers.PreviewManager	Helps in displaying video previews from OAK cameras
depthai_sdk.managers.EncodingManager	Helps in creating videos from OAK cameras
depthai_sdk.managers.BlobManager	Helps in downloading neural networks as MyriadX blobs
depthai_sdk.fps	For FPS calculations
depthai_sdk.previews	For frame handling
depthai_sdk.utils	For various most-common tasks

#### 5.2.4.3 OAK-1 PoE Electrical Characteristics

Protecting the OAK-1 PoE from any electrical damage was vital as this would have a negative impact on its reliability; therefore, it was important to consider the electrical characteristics of the camera. Below are two tables, Table 6 and Table 7, showing the absolute maximum ratings and the recommended operating conditions of the OAK-1 PoE.

**Table 6:** Absolute Maximum Ratings of the OAK-1 PoE

Symbol	Ratings	Min	Max	Unit
$V_{PoE}$	802.3af, Class3 input supply voltage range	37	57	V
$I_{PoE}$	Maximum Input Current Requirement		350	mA
$T_{stq}$	Ambient Temperature	0	60	C

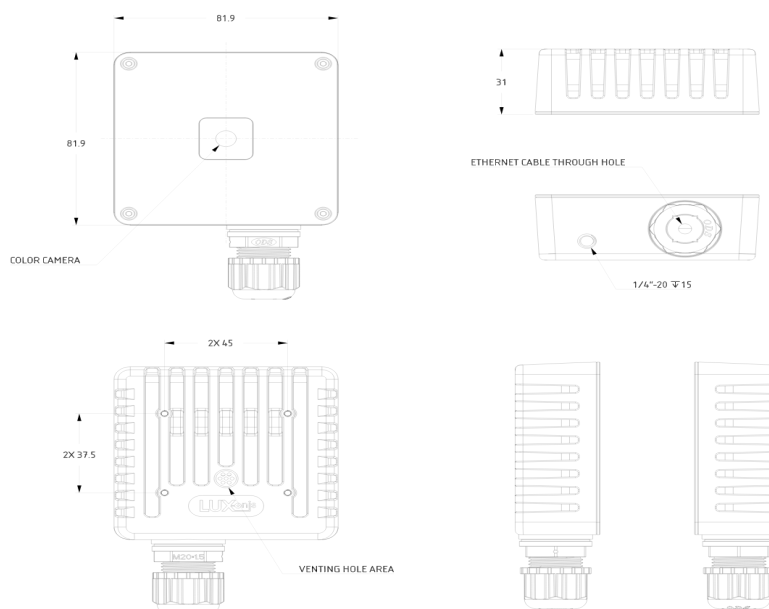
**Table 7: Recommended Operating Conditions of the OAK-1 PoE**

Symbol	Ratings	Min	Typ	Max	Unit
$V_{PoE}$	802.3af, Class3 input supply voltage range	37		57	V
P	Power Consumption Requirement	4	5	7.5	W
$P_{IDLE}$	$V_{BUS}$ Idle Power Draw (Myriad X Booted)		2.5		W
$T_A$	Ambient Operating Temperature			50	C

According to Luxonis, the power usage for the OAK-1-PoE ranged between 1.94 W (at standby) and 4.56 W (at max power consumption). During normal operation, we can expect the camera to pull about 4.2 MW. With this said, the ethernet cable used in our system should have a CAT5E rating or higher. This allowed us to achieve the 1 gigabit per second of data transfer promised by this camera and meet PoE requirements. Additionally, on the device end, the ethernet cable should use a shielded 8-pin RJ45 connector which should not be an issue since this is standardized.

#### 5.2.4.4 OAK-1 PoE Mechanical Information

The OAK-1 PoE is a fairly small camera with a height of 81.9 mm (114 mm if you include the tripod mount) and a width of 81.9 mm, a length of 31 mm, and a weight of 294 grams. Figure 12 shows the physical dimensions of this camera.

**Figure 12: OAK-1 PoE Mechanical Measurements [1]**

### 5.3 Microcontrollers

The microcontroller is responsible for driving the LEDs to display where available parking is. Since the local server handles the computation of available parking, the microcontroller is relieved of performing any space computations on its own. Instead, the MCU will solely display messages to drivers indicating how many spaces are available in a region and which way to drive to the available space. Through daisy-chaining, multiple LED matrixes can be driven by the same microcontroller. This relieves the need for multiple boards, each driving individual displays.

The PCB on which the MCU was integrated is connected directly to an external power source. Therefore, no particular power constraints must be considered for the microcontroller selection. Another factor was choosing a microcontroller with an integrated Ethernet controller, which could greatly reduce the complexity of the PCB design. The tradeoff was that the maximum data rate supported is 100 Mbps compared with the 1 Gbps supported data rate from a separate Ethernet controller. This tradeoff was a total non-factor, though, in a practical sense because the amount of data to be transmitted to the microcontroller is so minuscule compared to the allowable bandwidth. Therefore, the supported 100 Mbps data rate for most integrated Ethernet controllers was sufficient.

To drive the LED display(s), there had to be enough available general-purpose input/output pins (GPIOs) from the MCU to address all of the LEDs. Normally, the number of pins needed could be simply calculated from the below equation.

$$Pins = Rows + Columns$$

Since the 32 x 64 LED display is 2048 LEDs in total; the MCU must have enough pins to address the LEDs via multiplexing. Charlieplexing was considered, but it has limitations to its abilities in displaying RGB and changing brightness. But these pin considerations are not relevant in our case since the selected display makes use of shift registers and demultiplexers that allows for much fewer pins (16) to address a greater number of LEDs. Since each LED has three colors (RGB), they take approximately 3 bytes in RAM. This results in 6144 bytes. When accommodating for a second display to be daisy-chained, the byte count doubles to 12288 bytes. This puts the bare minimum RAM size at around 12 KB. According to SparkFun's product description, for a similar LED display at 32 x 32, a minimum clock frequency of 16 MHz is needed to display an image with tolerable flicker, so a microcontroller capable of running beyond that frequency was needed to drive the LEDs. Most microcontrollers with integrated Ethernet controllers have a minimum frequency of 66 MHz, so that issue was avoided.

Aside from technical requirements, the only factors left to consider were the recency, availability, and price of the microcontrollers. Additionally, the form factor was an element to consider since the board design had to be prototyped on a breadboard. With this in mind, a DIP package MCU made the most sense for prototyping, with the potential to pivot to more traditional form factors for the final PCB design. However, there were no MCUs in the DIP form factor with an integrated Ethernet controller, so our hand was

forced into that area. That left us with standard form factors like SOP (Small Outline Package) and QFP (Quad Flat Package), both surface-mount packages.

### **5.3.1 Atmel (Microchip)**

The selected Atmel processors date themselves with the name of Atmel still attached to the product. Atmel was bought by Microchip in 2016 but has continued releasing its product lines only under the new Microchip name.

#### **5.3.1.1 ATSAM4E8CA-AN**

The ATSAM4E8CA-AN is an older microcontroller released in 2016 that fulfills all of the earlier requirements. It has a 32-bit bus width and is built on the ARM Cortex M4 core. The MCU has 512 KB of flash memory, 128 KB of SRAM, and a 120 MHz clock. The form factor is the low-profile quad flat package (LQFP). It currently is very cheap, \$2.04 per MCU. Despite all these positives, it has the one downside of being a 100-pin chip. This is an excess of pins we do not need for our design and could create unnecessary complications in the PCB design phase.

#### **5.3.1.2 AT32UC3A1128-AUT**

The AT32UC3A1128-AUT is one of the older microcontrollers, having been released in 2012. It has similar specifications to the above MCU except for a reduced frequency of 66 MHz, 128 KB of flash memory, 32 KB of SRAM, and the AVR architecture. The form factor is the thin quad flat pack (TQFP). With this MCU also being 100-pin and more expensive at \$8.39 per chip, this MCU quickly became the first choice to ditch. Additionally, its age would be a source of liability when attempting to flash it or perform other operations that would require new vendor tools that may no longer support the MCU.

### **5.3.2 Microchip**

The following are Microchip's microcontrollers, one of which is much newer and has a far better chance of being supported in Microchip's ecosystem of vendor tools. Of all microcontroller producers/designers, Atmel and Microchip were the only companies to consistently produce MCUs with integrated Ethernet controllers.

#### **5.3.2.1 ATSAME70J19A-AN**

The ATSAME70J19A-AN is a continuation of the shared line with the ATSAM4E8CA-AN. This time, the core is an ARM Cortex-M7 on a 32-bit bus. The MCU has 512 KB of flash memory the 256 KB of multi-port SRAM and runs at a max frequency of 300 MHz. The form factor is LQFP. Not only does this MCU blow the others out of the water, but it has only 64 pins, making it a great candidate for the final PCB. It costs \$11.48 per chip.

#### **5.3.2.2 PIC32MX664F064L-I/PF**

The PIC32MX664F064L-I/PF was released in 2010, making it the oldest microcontroller on this list. It is based on the MIPS32 4K processor core. It runs at a max frequency of 80

MHz and has 64 KB of flash and 32 KB of SRAM. It has 100 pins and is in the TQFP form factor. This MCU may be the worst option of the bunch because of its lackluster showing in the memory category. At \$7.51 per chip, it's not the choice to make.

### 5.3.3 Infineon XMC4504F100F512ACXQMA1

The XMC4504F100F512ACXQMA1 is the one microcontroller that is not connected to Microchip, which also fits the desired criteria. It was released in 2017 and is based on the ARM Cortex-M4 core. It runs at a max frequency of 120 MHz and has 512 KB of flash and 128 KB of SRAM. Its form factor is LQFP and has 100 pins. The cost was \$11.27 per chip. The Infineon MCU was a decent option, but the reality that it matches or underperforms the Microchip ATSAME70J19A-AN in every major category makes it a secondary option. The major problem was the 100-pin layout of the chip, which could be a problem when designing the PCB.

### 5.3.4 Comparison Chart

Table 8 shows the criteria we used to compare the microcontrollers.

**Table 8:** Microcontroller Comparison

<b>Part Number</b>	<b>Atmel ATSAM4 E8CA-AN</b>	<b>Atmel AT32UC3A 1128-AUT</b>	<b>Microchip ATSAME70 J19A-AN</b>	<b>Microchip PIC32MX66 4F064L-I/PF</b>	<b>Infineon XMC4504F 100F512A CXQMA1</b>
<b>Release Year</b>	2016	2012	2021	2010	2017
<b>Unit Price</b>	\$2.04	\$8.39	\$11.48	\$7.51	\$11.27
<b>Form Factor</b>	LQFP	TQFP	LQFP	TQFP	LQFP
<b>Pins</b>	100	100	64	100	100
<b>Flash Size (KB)</b>	512	128	512	64	512
<b>SRAM (KB)</b>	128	32	256	32	128
<b>Frequency (kHz)</b>	120	66	300	80	120
<b>SPI/I<sup>2</sup>C/USB</b>	Yes	Yes	Yes (No SPI)	Yes	Yes
<b>Ethernet Controller</b>	Yes	Yes	Yes	Yes	Yes
<b>LCD Controller</b>	Yes	Yes	Yes	No	No

### 5.3.5 Microcontroller Final Selection

Putting together the chart made the decision as obvious as possible. We selected the most powerful microcontroller that fulfilled our purposes, the Microchip ATSAME70J19A-AN. It was not the cheapest, but for only a couple of dollars more than

the competitors, it could far outperform in speed, total flash, SRAM, and being the better form factor with a smaller pinout. It was the best fit for our needs.

## 5.4 LEDs

In the grand scheme of the garage parking enhancements, the LEDs play the role of being the immediate indicator for drivers to find open spots. The LEDs purely indicate where open parking spots are, nothing more. The microcontroller receives data from the local server and then drives the LED to display whatever data was indicated by the server. That could be an arrow or a number indicating the total open spots in that region. There were several types of displays that could have gotten the job done, but it was up to how much we wanted to customize the look and feel of what was to be displayed.

### 5.4.1 LED Options

The first option was a seven-segment display. These displays are very popular and straightforward to interface with. They can display numerical values and some letters, but they are pretty limited on that front. If one wants to display arrows or sentences, they are out of luck. Another problem is that most seven-segment displays for sale are far too small to work for our use case. To have the display large enough, we would have most likely had to manufacture our display. Manufacturing a display was not worth the effort since it takes focus away from where we were trying to innovate on our project.

The second option was to use a dot-matrix LED display. This would give the ability to have far more precise images displayed, like arrows, full sentences, patterns, etc. The main trade-off was that because there is a higher density of LEDs, more GPIO pins are needed to drive the display. However, there is a workaround to limit the increase in pins from the microcontroller driving the display. If, in addition to multiplexing, shift registers are used, single pins can be stretched to control many more LEDs. This limits control of the LEDs and disallows the use of Pulse Width Modulation (PWM), but it offers far greater flexibility by freeing up the MCU's resources.

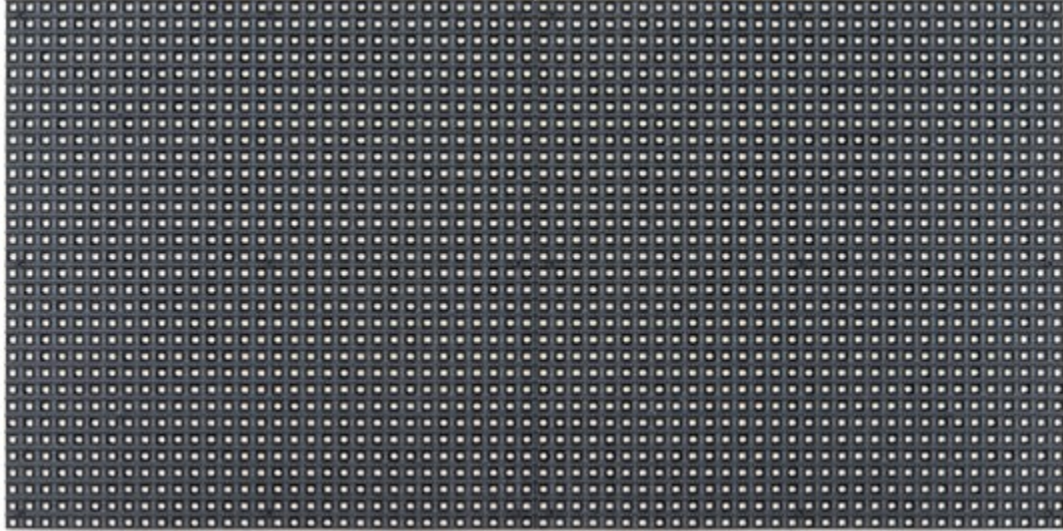
The third option was to use a Liquid Crystal Display (LCD). LCDs are popular for many applications. They provide a high-definition image and display vibrant colors. But an LCD can be considered a continuation of the philosophy behind a dot matrix LED display. The LCD has an even higher resolution and, generally, a smaller form factor to go along with it. This results in LCDs often being the most demanding display to drive of the three options listed. LCDs can be great in their specific applications, but when trying to display a large image powered by a microcontroller, a different display works better.

### 5.4.2 LED Selection

Since seven-segment displays cannot show details and LCDs are too demanding, the dot matrix LED display is what made the most sense for our project. It can display arrows directing drivers where to go and even show letters and numbers in combination to potentially indicate specific open parking spots. There were multiple options for which matrix LED display to choose from, but SparkFun had the best offerings for what we needed in the 32 x 64 dot matrix LED display, as shown in Figure 13. Their displays



could do better than directly addressing pixels through multiplexing by using shift registers to help address the different colors of the LEDs (because they are three-color RGB). This way, only 16 MCU pins were needed to drive the LED display, rather than the 96 that would normally be needed.



**Figure 13:** RGB LED Matrix Panel - 32x64

Another advantage of this display is that it can be daisy-chained with another to form a bigger LED sign. Or, through clever programming, the daisy chain can be treated as separate displays so that different messages can be displayed concurrently. Either way, this display provides a lot of flexibility for how we choose to communicate where drivers should look for open parking spots.

## 5.5 Local Server

The parking system relies on a central processing computer (local server) for communication between the cameras, LED displays, and the web server supporting mobile and web applications. This local server receives the available spaces from the cameras, updates its local database, and sends the open space data to the display's microcontroller and the cloud database server. The server also operates as the gateway providing internet access to the parking system. One of the goals in choosing an appropriate server is to ensure the server is capable of handling the parking system for an entire parking garage, such as Parking Garage C at UCF. Another goal is to choose a computer to perform as a server with all the components built-in; this feature helps in our cost-effective goals of the project. The technical term for a computer that meets this description is called a Single Board Computer (SBC), a complete computer where the microprocessors, memory, and input/output functions are built-in on a single circuit board. It normally does not contain expansion slots for additional peripherals, and the amount of RAM is usually predetermined [30]. One feature that is not required for the normal function of the server but is needed during development and a demonstration is a display port. Therefore, the server or computer board to choose from needs to have a display port. As mentioned before, the final implementation of the project does not include the web and mobile app, and thus, there is no connecting from the local server to any webserver on the internet.

In this research, three boards were selected to be analyzed and discussed that could potentially meet the parking system's needs. These three boards are discussed in the following three sections.

### 5.5.1 Odyssey X86J4125864

Edge computing devices are playing an increasingly important role in the field of IoT devices. ODYSSEY is a series of Single Board Computers for edge computing applications. This mini PC has an Intel processor and an ARM microcontroller on the same board. Its large built-in heat sink covers the whole bottom part of the board, and in conjunction with the CPU fan, both can efficiently dissipate the processor's heat and ensures that the system stays well within its operation temperatures despite the Florida Summer heat. The board supports Windows 10 or Linux. This board presents a coprocessor as an additional feature, the Microchip ATSAMD21G18 microcontroller based on the ARM Cortex M0+; however, at this moment, there is no particular use for it, and therefore, it would remain unused if this board is picked. In addition, the board offers an incredible amount of computational power; at a 2.0 GHz base CPU frequency with bursts of 2.7GHz is more than capable of processing the data input from the video cameras, space availability calculations, data outputs to the LED displays, and data output to the web server in the cloud. The Intel processor comes with a built-in graphics processing unit, and therefore it supports a display. One downside of this board is the price; for \$238, it is somewhat pricey and will probably play a significant role in the decision stages of the project. The specifications for this model are summarized in Table 9.

**Table 9:** Specifications for Odyssey Mini PC

<b>Specifications</b>	<b>Description</b>
<b>Name and Model</b>	Odyssey X86J4125864 by Seeed
<b>CPU</b>	2.0-2.7GHz 64-bit quad-core Intel Celeron J4125
<b>Coprocessor</b>	Microchip ATSAMD21G18 32-bit ARM Cortex M0+ onboard
<b>Graphics</b>	Intel UHD Graphics 600 (Frequency: 250 – 750MHz)
<b>Memory RAM</b>	LPDDR4 8GB
<b>Wireless</b>	Dual Band Wi-Fi 802.11 a/b/g/n/ac @ 2.4 and 5 GHz HT160 and Bluetooth BLE 5.0
<b>Networking</b>	Intel I211AT PCIe Dual Gigabit LAN
<b>Storage</b>	64GB eMMC
<b>USB</b>	2.0 Type-A x2, USB 3.1 Type-A x1, USB 3.1 Type-c x1
<b>Video Interfaces</b>	HDMI 2.0a, DP1.2a
<b>Expansion Slots</b>	M.2 (Key B, 2242/2280), SATA III, M.2 (Key M, 2242/2280), PCIe 2.0 x4, Micro SD card Socket; SIM card Socket.
<b>Power</b>	DC Jack 5.5/2.1mm or Type-c PD; DC Jack input: 12-19V; Type-C input: 15V DC
<b>Dimensions</b>	110x100mm (4.3x4.3 inches)
<b>OS</b>	Windows 10 and Linux
<b>Price</b>	\$238

### 5.5.2 Raspberry Pi 4

Raspberry Pi is a series of single-board computers made by the Raspberry Pi Foundation, a UK charity. It is a tiny and affordable computer that can be used to learn to program and build hardware projects, automation, edge computing, and industrial applications. It can run Linux and provides general-purpose input and output (GPIOs) pins that allow controlling electronic components such as sensors. The specific model discussed in the

section is the Raspberry Pi 4 Model B. A summary of specifications can be found in Table 10.

**Table 10:** Raspberry Pi 4 Specifications

Specifications	Description
<b>Name and Model</b>	Raspberry Pi 4 Model B
<b>CPU</b>	Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
<b>Memory RAM</b>	1GB, 2GB, 4GB, or 8GB LPDDR4 SDRAM (Depending on the model)
<b>Wireless</b>	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
<b>Networking</b>	Gigabit Ethernet
<b>Storage</b>	Micro-SD card slot for loading operating system and data storage
<b>USB</b>	2 USB 3.0 ports and 2 USB 2.0 ports
<b>Video Interfaces</b>	2 micro-HDMI ports (supports 4K), 2-lane MIPI DSI display port, 2-lane MIPI CSI camera port.
<b>Expansion Slots</b>	Raspberry Pi standard 40-pin GPIO header.
<b>Power</b>	5V DC via USB-C connector (minimum 3A) 5V DC via GPIO header (minimum 3A) Power over Ethernet (PoE) enabled (requires separate PoE HAT)
<b>Dimensions</b>	3.37 x 2.22 inches (56.5 x 85.6 mm)
<b>Others</b>	H.265 and H264 decode/encode. OpenGL ES 3.1, Vulkan 1.0
<b>Price</b>	1GB \$30, 2GB \$45, 4GB \$55, \$8GB \$75

### 5.5.3 UDOO X86 II

UDOO is a family of Open Source Arduino-powered Mini PC compatible with Windows, Android, and any Linux Distro. The UDOO X86 II series is a powerful x86 maker board with an Arduino Leonardo-compatible board embedded on the same board. The specifications of this board are shown in Table 11.

**Table 11:** Specifications for UDOO X86 II

Specifications	Description
<b>Name and Model</b>	UDOO X86 II Advance Plus
<b>CPU</b>	Intel Celeron N3160 @ 1.60-2.24GHz, Quad-core
<b>Coprocessor</b>	Atmega32U microcontroller.
<b>Graphics</b>	Intel HD Graphics 400
<b>Memory RAM</b>	4GB DDR3L Dual Channel
<b>Wireless</b>	M.2 Key E slot for optional Wireless Modules
<b>Networking</b>	Gigabit Ethernet connector
<b>Storage</b>	32GB eMMC storage
<b>USB</b>	3x USB 3.0 Type-A ports
<b>Video Interfaces</b>	1xHDMI and 2x miniDP++ ports
<b>Expansion Slots</b>	SATA, M.2 Key B SSD slot, micro SD card slot.
<b>Power</b>	DC in Jack 12V 3A
<b>Dimensions</b>	4.72 inches x 3.35 inches (120 mm x 85 mm)
<b>Others</b>	HW Video decode H.265/HEVC, H264, MPEG2, MVC, VC-1, WMV9, JPEG, VP8; HW Video encode: H.264, MVC, JPEG 1 x UART, 1 x I2C, 1 x SPI 23 digital I/O (PWM) 12 analog inputs
<b>OS Support</b>	Windows 10, Linux, Android
<b>Price</b>	\$214

#### 5.5.4 Local Server Choice

All three boards are capable of being a suitable computer to work as the control unit. The most powerful one, the Odyssey board, comes with all the components needed to install Linux and start developing, except for the power supply. In contrast, the Raspberry Pi not only does not come with a power supply, but it does not provide onboard storage. Nonetheless, even after adding the price of an SD card to the cost of the raspberry pi, it

costs less overall than the Odyssey or the ODOO boards. Since the parking project demonstration was planned to have a couple of cameras and a couple of LED signs, then any of the three boards could do the job. However, the availability of the boards played an important role when selecting the computer. When building the prototype, the raspberry pi 4 was out of stock, with no option to backorder it. The ODDO board was also unavailable, and its website did not provide future availability. Although the Odyssey board was currently out of stock, their website allowed backorders, with an estimated shipping date of May 7th of, 2022. Although the team decided that the Raspberry Pi was the right choice to maintain a cost-effective approach, it was ultimately the availability of the boards that was the deciding factor.

Initially, the Intel-based Odyssey X86J4125864 board from Seed was picked as the server; however, it became unavailable for a long time and only became available as a backorder. Therefore, the team decided to try getting a Raspberry Pi 4 model B with 8GB of RAM, which was not a simple choice since it was out of stock everywhere. Still, it became available later as a kit from the Canakit website. By the time it was ordered and received, the software development for the parking system had already begun using an intel-based computer. Since the Raspberry Pi is based on the ARM architecture, the team predicted some incompatibilities. After a few days of unsuccessfully trying to run java version 17, MySQL database, and the parking system software, the Raspberry Pi was put aside in lieu of an intel-based computer. Finally, a mini pc capable of running Microsoft Windows, and thus intel-based, was ordered. The model Mini S from Beelink has an 11th Generation Intel Celeron processor, model N5095, which has four cores and runs up to 2.9GHz; it has 8GB DDR4, 128GB SSD, and Windows 11 Pro. After a quick preliminary test, it was determined that this minicomputer could run the software needed without any issues.

## **5.6 Ethernet PoE Switch and Local Network Internet Access**

### **5.6.1 PoE Switches**

Power over Ethernet (PoE) is a technology that describes the transmission of electricity along with data in a single ethernet cable. It is used to deploy powered devices to areas where electricity is unavailable or to minimize the expense of installing additional electric wires and outlets. Therefore, a PoE switch is a regular switch with many ethernet ports that provide power and data to devices. A parking system that entitles the entire garage will require many cameras; therefore, a large PoE switch would be required. However, a small 5-port switch will be enough for the project's proof of concept. The chosen camera in the research section of this document requires PoE 802.3af; this version of PoE provides 12.95 watts and a maximum current of 350mA. It is easy to match the PoE requirements of the camera to a PoE switch because as long as the switch supports PoE 802.3af or above, it will work.

There are several small switches readily available with the desired PoE feature. For example, NETGEAR carries a 5-port PoE switch, model GS305P-100NAS, from which four ports provide PoE+ (PoE = 802.3af and PoE+ = 802.3at, the next PoE generation that almost doubles the wattage). PoE+ devices are backward compatible with PoE

devices; therefore, this gigabit switch will work with the camera, and its cost is \$49.99. TP-Link carries a suitable switch for \$44.99; model TP-Link TL-SG1005P V2 includes four PoE ports. Another switch, model GPOE204 made by STEAMEMO, is a 5-port gigabit switch for \$33.99 that offers four PoE+ ports. It comes with additional features such as VLAN and 1 SPI uplink port; however, these features are not relevant to the parking system and will not be discussed. A summary of these switches can be found in Table 12.

**Table 12:** Summary of PoE Switches

Brand	Model	No. of PoE ports	Price
Netgear	GS305P-100NAS	4 PoE+ ports	\$49.99
TP-Link	TL-SG1005P V2	4 PoE ports	\$44.99
STEAMEMO	GPOE204	4 PoE+ ports	\$33.99

#### 5.6.1.1 PoE Switch Choice

The switch the team decided on was the STEAMEMO, model GPOE204. It offers enough ports for the parking system, so it is an excellent option if the team adds more than one camera and LED display. Also, with a price tag of \$33.99, this model is the most cost-efficient option.

#### 5.6.2 Local Network Internet Access

The local network needs internet access for the parking system to communicate with the database in the cloud to support mobile and web applications. Several options exist to provide internet access to the local network, such as using a wireless router to connect to UCF's network, a cellular modem, or a personal cell phone's hotspot feature. As mentioned before, the web application and mobile application features were not implemented, and therefore, none of the methods described in this section were needed as internet access was no longer required.

##### 5.6.2.1 Cellular Modem

Cellular modems add 2G/3G/4G cellular connectivity to laptops, desktops, and tablets; they come as USB dongles, PCI or PCIe express cards, or standalone modems [22]. USB dongles connect to a USB port and are usually available for Windows and Linux. For example, Hologram.io offers a 2G/3G Nova Global Cellular Modem for \$83, \$5 for the SIM card, and a monthly cost of \$0.70 plus \$0.08 per MB of data used [23][24]. This is a good option because it works on Windows, Linux, and Raspberry Pi single-board computers.

The PCI, PCIe, or M.2 cellular modems connect to an available slot on the motherboard of choice, and after installing the drivers and a SIM card, the server can connect to the internet and share its internet access with the local network via ethernet. For example,

Quectel, a global IoT solutions provider [25], offers the EM06-A LET Cat 6 cellular module for \$49.50, which was compatible with the single-board computer model Odyssey X86J4125864. This module plugs into the M.2 slot on the board, and it needs an antenna and a SIM card to function. The SIM card can be the same one provided by Hologram.io using the same monthly plan mentioned above; an antenna for this modem is available for about \$2 to \$4 each.

Standalone modems are also called hot spots and are sold by many cellphone companies; these devices connect to a cellular network and share their internet access via a Wi-Fi network to which several WiFi-enabled devices can connect. For example, Verizon offers these mobile hotspot modems for \$79.99, \$199.99, \$299.99, and \$399.99 depending on the selected speed (i.e., 2G, 3G, 4G, or 5G) and a one-time activation fee of \$35 [26].

#### **5.6.2.2 Smartphone's Hotspot and Tethering.**

Most modern smartphones can turn into Wi-Fi hotspots; they can share their data with 4G or 5G internet access via a Wi-Fi signal. The local server would have to have a Wi-Fi adapter to connect to the Wi-Fi signal. However, the server would still need to be connected to the local network via ethernet cable. Although a smartphone hotspot is not a permanent solution to a parking system, it can still work for the parking system proof of concept. As an alternative, smartphones can be connected to a computer via a USB cable to be used as a cellular modem to connect to the internet; This is called tethering.

#### **5.6.2.3 Wireless Router**

A wireless router can create a local network that serves as the internet gateway by connecting to UCF's existing network. The WAN port on the router can be connected to any UCF's provided ethernet port on-site. However, after talking to UCF's IT department, it was determined that UCF's network policy does not allow wireless routers to be connected to its network for security reasons. Therefore, connecting a wireless router to UCF's network is out of the question for our parking system proof of concept. This research found an alternative wireless router that allows personal mobile device tethering capabilities to be plugged into the router to connect to the internet. The router then shares its internet via ethernet ports and a Wi-Fi network. This Wi-Fi router model GL.iNet GL-AR750S-Ext costs \$72.90, and it comes with two-gigabit ports and supports both 2.4GHz and 5GHz Wi-Fi signals.

Table 13 is a collection of solutions and the breakdown of their pricing and monthly subscription rate for visual purposes.



**Table 13:** Summary of Internet Solutions

Description	Unit Price	Monthly cost
USB 2G/3G Nova Global Cellular Modem	\$83.00	\$0.70 + \$0.08 per MB
EM06-A LET Cat 6 Cellular Module	\$49.50	\$0.70 + \$0.08 per MB
Verizon Mobile Standalone Hotspots (2G/3G/4G)	\$79.99 - \$399.99	\$60.00
Smartphone's hotspot or Tethering (T-Mobile)	N/A	\$0 (included in phone's plan)
GL.iNet FL-AR750S-Ext (Using smartphone Tethering)	\$72.90	\$0

#### 5.6.2.4 Internet Access Choice

Many choices were found that can provide internet access to the parking system. The most expensive solutions were the Verizon hotspots, but one fantastic feature these units have is that they are battery-powered, an excellent option for testing purposes on-site where electrical outlets may not be available or close by. However, the team decided it was not a cost-effective solution due to the monthly commitment of \$60 for a data plan. The following considered units were the cellular modules. Each unit's price was not that bad; however, the EM06-A module is the cheaper option, and since both monthly commitments are the same, the EM06-a module would be a clear choice among the two.

On the other hand, the tethering option of smartphones connected directly to a USB port on the local server is the most cost-effective since any team member's smartphone can be used at no extra cost. Fortunately, both Windows and Ubuntu operating systems support this scenario and can share this internet access to the network via their ethernet port. However, to have a backup Wi-Fi network in case there is difficulty connecting either the camera or the LED displays to the local server via ethernet cable, the Wi-Fi router with the tethering feature was chosen instead. The team will not incur any monthly fees associated with cellular modem data plans with this tethering option. Since the proof of concept is a small scale of the entire parking system, the team feels this option would be the most appropriate.

## 5.7 Web Application Research

Some form of web outlet would be our main way of communication with the consumers/users. One of the main aspects of any technological project has come down to the UI/UX aspect of the application. The complete and robust implementation of WebApp not only makes the users want to utilize the app more but also aids in increasing the digital footprint of the App. We intended to have a user-friendly, highly accessible, completely responsive WebApp to support our SmartParking system and cater to our

users. The final implementation of our project, as mentioned before, was not implemented due to not having enough time.

### **5.7.1 Web Application Types**

With the increasing demand for various technologies, the need for web applications has also changed with time. Depending on the business needs and consumer requirements, there have been implementations of nine different web application formats.

#### **5.7.1.1 Static Web Applications**

The most primitive and original form of a web application is the static web application, and it implies exactly what it suggests. These websites are static, meaning there is no communication module between the server and the user. Most of these static websites are effortlessly simple with the only implementation of basic HTML and CSS language. There could also be GIFs, videos, or animated banners included on these websites to attract users. With mobile applications, these web application types do not integrate well due to the need to send and receive huge amounts of data, which often leads to poor application performance. Mostly for the Web-Based Portfolio of an individual or a corporation where no back-end interaction is needed, these types of applications are utilized.

#### **5.7.1.2 Dynamic Web Applications**

One of the most popular forms of web applications, the Dynamic Web App allows real-time communication. It allows data generation upon the user's request and backend server response. They allow direct connections and interactions on the client side of the web app. They are significantly more complex, technology-wise, than static web apps and have various interactive components for full functionality. Databases are used to store the data that need to be shown on the web app. Administrations and organization boards are utilized for the admins to alter and improve the contents within the application, mainly for the frontend and backend parts. A lot of different languages are utilized to implement such web applications. An example of such an application would be Netflix. Depending on what the user wants and searches, the web app returns the precise content or some extremely similar items if the exact content can not be found. This process requires real-time computation and analysis. This application allows users to read, write, refresh and even delete data.

#### **5.7.1.3 Single Page Web Applications**

These apps do not need any page reloading as it reflects exactly what their title says. These websites allow efficient communication between clients and websites with minimum discrepancy. Responses from the back end of the website and requirements are more agile due to the low quantity of data. The logic implementations are usually done on the browser rather than the backend server. The clients can interact with the totality of the website characteristics within a single page. However, SEO guidelines are not supported for SPA websites due to their Universal URL nature. Many popular social media platforms are currently utilizing SPA websites for their quick response times and logic

implementation on browser qualities. Websites like Google, Twitter, Google Maps, and Gmail - are all single-page web applications.

#### **5.7.1.4 Multi-Page Web Applications**

These applications are more complex and offer many more features. For these websites, there is more than one page, and each time a user would like to navigate to another page, the entire page has to be reloaded. Whenever a customer browses different website features, the data from the back server fills up and reloads the page with new data. So as can be expected, the logic implementation of the website takes place on the backend side, and the requirements from the client and server are reversed. The user interface gets affected by this long process of creating pages on the backend and presenting them to the browser for the user. So AJAX is utilized to tackle unexpected segment issues without reloading numerous times. Multi-page applications are widely used in the modern world, and often they also are supported by both mobile and web browsers if the front-end design of the application is made responsive. Some of the more popular languages used to implement Multi-Page Applications are HTML, CSS, JavaScript, JQuery, AJAX, etc. SEOs are better supported for these web applications since all the pages are optimized for keywords. Due to their scalability with page limitations, modern websites, portals, marketplaces, stores, and enterprise applications heavily utilize MPA websites. But the only constraint for these web apps is that they are complex and difficult to maintain. Websites like Amazon, eBay, and Trello are some examples of MPA.

#### **5.7.1.5 Animated Web Applications**

Closely correlated with modern Flash technology, Animated Web Applications help market or showcase content via numerous visual animation effects. Thus, SEO is unsuitable for these web apps because keywords are unclear for the web pages and can not be read accurately. These web apps largely imply the usage of the creative and design aspect of web development; thus, the UI and UX engineers have the freedom of being extremely creative and utilize elephants of web design that are not supported by other web applications. This helps enrich the user experience because the users are exposed to unique designs and catchy effects along with the necessary information that the website is initially programmed to convey. CSS3, JavaScript, and SVG are some of the more popular tools for implementing such websites. Squadeasy and Miki Mottos are some of the best Animated Web Applications. However, few companies rely on these websites to communicate with customers because of search engine optimization limitations.

#### **5.7.1.6 E-Commerce Web Applications**

E-Commerce is a big contributor to the modern-day economy. E-Commerce businesses help increase digital foot traffic to consumers as opposed to shopping in physical stores. Lately, e-commerce has gained popularity, and thus web applications that are utilized to port digital stores onto the web have gotten more important and are being optimized to aid in sales. Web applications offer a lot of complex features to support all the functionalities of a physical store. These include electronic payment, online transactions, customer service capabilities, online browsing of store content, online shopping cart option, adding new products or removing old products, and many more. However, with

all these complex and intricate features come the burden of increased complexities on the developer end and maintenance constraints. Thus usually, for these websites, dedicated developers are often hired to maintain the security, sustainability, and efficiency of the final app. Also, such websites need robust and user-friendly graphical interfaces to increase digital food traffic or scale up. As a result, User Interface and User Experience engineers are needed to ensure all aspects of an efficient graphics interface are covered. Along with the interface for the users, developers must also include an admin interface for the website so that the online store employees can add new products. Shopify is one of the biggest E-Commerce web apps that is widely utilized.

#### **5.7.1.7 Portal Web Applications**

These applications offer the peculiar feature of having different categories of items available on the home page. Such features include forums, chats, e-mail, search engines, browsers, blog posts, the latest content, register/sign-up options, and many more. Since these applications have high feature attributes, many enterprise companies often use these applications to customize and tailor the experiences of the specific targeted audience. Often portal web applications offer different views depending on regular user or admin privileges. Admin sometimes also can monitor the activities of the signed-in user to the app. It also allows the capability of restricting certain functionalities of the web app depending on the user or admin account. Many websites, such as Udemy and Coursera, are utilizing the portal web application format to uniquely design the content on their website to cater to specific users.

#### **5.7.1.8 Rich Internet Applications**

These web applications are fundamentally implemented to be integrated better with the browsers and aid in resolving the restrictions implied. They utilize client-level plugins, which include Silverlight, Shockwave, and Flash. These applications support numerous desktop applications functionalities but with the extra addition of improved, more agile, and engaging communication systems. Due to these applications relying on the plugins and tools that improve efficiency and user engagement, they are better suited to improve visual user experience than generic program applications. But complete reliance on such plugins and tools brings up constraints of unannounced vulnerabilities and security gaps. For instance, any outdated or faulty plugins or tools will result in the websites partially being inactive or certain major components/functionalities being unavailable. Websites like Google Gears, Adobe Flash, and Microsoft Silverlight utilize Rich Internet Applications to improve the visual aspects for the catered users.

#### **5.7.1.9 Progressive Web Applications**

With similarities in visual aspects of mobile applications, these web applications are one of the most advanced forms of web development. With the increase in digital footprint from mobile devices, the importance of websites supported by mobile browsers is gaining more and more popularity. With Progressive Web Applications, users have the luxury of accessing all the necessary data from the web application with vast features and increased performance from any mobile browser. Such functionalities allow users to enjoy the enhanced mobile web experience and improve service even with slow network

connections. However, the main objective for implementing such web applications was never to increase functionalities or new features within the architecture. Rather this particular web form was intended to optimize the agility and adaptability between web apps and mobile devices through poor network connections. Some major benefits of such applications include caching, home screen installation, efficient data transmission with HTTP, and many more. Many popular websites for renowned companies are implemented with Progressive Web Application features. Companies like OLX, Starbucks, Forbes, Pinterest, and Spotify are just some of them.

#### **5.7.1.10 Decided Web Application**

The “smart” aspect of the Smart Parking Project refers to using advanced technologies to enhance the current parking experience and solve issues relating to the topic. Thus usage of a website to enhance the user experience for the targeted users is crucial to the project implementation. The project is planned to have video cameras set to record vehicle movements and presence within a certain amount of spaces and utilize machine learning algorithms to make aid in parking decisions for the intended users. Thus being able to visualize the different metrics, such as open parking spots and available spots, is crucial. Showcasing video camera recording in real-time would also aid users in navigating and making smart parking decisions. However, other than these features, for Smart Parking, no more complex logic implementations or real-time processing would be needed. Considering all aspects and the necessary complexities of the project, it can be stated that utilizing a Dynamic Web Application to support server-side and user-side real-time communications would fit in and support all the major requirements.

### **5.7.2 Web Development Stacks**

Web development stacks mainly refer to the collection of different software set up together to implement and support the total functionality of websites. Different software is usually utilized to support different sides, such as the user, server, and database side of the web applications. In this retrospective, the software is stacked to support the website. With the raging technological improvements, the web development world has adapted many different web stacks for their specific needs and advantages. Selecting a correct stack to support the functionalities of the Smart Parking Web Application is a critical element of the project.

#### **5.7.2.1 LAMP Stack**

The LAMP stack is considered the most mature web development. It is one of the first open-source technology stacks. LAMP stack is the compilation of Linux, Apache, MySQL, and PHP. Each of these technologies has its purpose. MySQL is the data server for the stack with the utilization of SQL language. Apache is the HTTP Web Server to support the communications done throughout the websites. PHP is the backend side of the web stack. Linux is utilized as the operating system for this particular stack. This stack provides enormous performance capabilities and flexibilities reflecting the needs through customized modifications. It is the industry standard with optimized efficiency and peak performance. LAMP stacks also support FrontEnd or the user side of the web applications with low-level JavaScript, HTML, and CSS for implementing necessary

graphics interfaces for the users. Since the operating system can be changed for any tech stack, it is easy to change the Linux-based operating system to a Microsoft Windows system which would create WAMP, or even a MAC OS-enabled stack which results in a MAMP stack.

### **5.7.2.2 MEAN Stack**

This stack is one of the newer technology stacks of the modern world. It consists of Express.JS, Angular.JS, Node.JS, and MongoDB. Express, Angular, and Node are all languages utilizing JavaScript. Thus MERN can be considered through and through JavaScript tag. This feature aids in the learning and implementation organization. The entire web stack utilizes JavaScript, so the structure is more optimized and simplifies the learning curve and usage, aiding in seamless integration within the stack's components. Express.JS supports the server-side, back-end portion of the stack. Angular.JS implements the front end or user end of the web stack. Node.JS is a runtime environment, which could be considered the web application server. MongoDB is a NoSQL database to store information from the client and server sides. With the usage of a single language, the MEAN stack also provides the ability of code reusability across the platform to decrease redundancy. The technologies within the MEAN stack are all open source and free, which enables an ample amount of resource sharing for the web developer community. JavaScript is gaining immense popularity throughout different industries; thus, complete utilization of Javascript across both servers and client-side makes it stay with the trend and simple execution for the developers. MEAN stack offers scalability and flexibility in cloud hosting, and with the included web server, the web deployment is also super simple. Different components within the stack communicate in JSON data transmission, which optimizes communication efficiency. So, the MEAN stack allows the implementation of agile and immensely efficient apps.

### **5.7.2.3 MERN Stack**

MERN stack is extremely similar and shares many of the same technologies as the previously discussed MEAN stack. The new addition to the MERN stack is implementing the front-end user side of the web application with ReactJS. ReactJS has a lot of benefits of its own. With the utilization of the Virtual DOM embedded within, React.JS makes any changes made to the technology code - seamless. One of the advanced forms of modern JavaScript, JSX, is used within React.JS which provides harmonious component support and communication. React allows simultaneous code usage between servers and browsers with a powerful built-in library. For a lot of off-the-shelf high-performance web applications, the MERN stack is extensively implemented. React.JS requires no templates. This allows the reduction of repetitive DOM and HTML elements. The isometric nature of React allows it to run on both the user side and the client side of the web application. This aids in Search Engine Optimization purposes as pages can be created on the server. Thus, using ReactJS, code development is more efficient and faster. It is great at supporting applications with low-level complexities.

#### 5.7.2.4 MEVN Stack

Very similar to the previous two discussed stacks, the MEVN stack is a combination of MongoDB, NodeJS, and ExpressJS. However, instead of the former ReachJS or AngularJS, this particular stack utilizes another JavaScript-based front-end framework called VueJS. The benefits of VueJS include a rich set of development tools and lightweight out-of-the-box functionalities, which also extends to third-party services. VueJS represents a framework, whereas ReactJS is a library. VueJS also utilizes HTML instead of JSX, which is used by ReactJS. MEVN stack introduces the application of the newest technologies out of the other JavaScript stacks like MEAN and MERN. However, this also poses a constraint towards the MEVN stack due to the recent development of VueJS and not a lot of resources being developed compared to the other front-end frameworks.

#### 5.7.2.5 Python - Django Stack

Different from the stacks utilized in the discussions above, the Python-Django stack uses one of the most popular and fast-growing languages, Python. For the web server, the Python-Django stack has Apache for support. MySQL is used as the database for web applications. Also, due to the machine learning and data science capabilities of Python language, this stack has the potential to take web development to the next stage. Django framework is utilized for the server-side support of web applications, which is entirely written in Python. JavaScript is utilized for the frontend aspect of the web stack. This stack also provides a rich collection of third-party packages to enrich the developing experience.

#### 5.7.2.6 Ruby on Rails Stack

The dynamic programming language, Ruby, is used on the Ruby on Rails web stack. The server-side backend aspect of the web stack utilizes Ruby, which supports default structures and database management. Sinatra, Hanami, and Padrino are some of the more popular frameworks for web development with the Ruby on Rails stack. Ruby language is open-source; thus, a large community of developers is experienced, and rich resources are available on the world wide web. A strong infrastructure along with test systems and database integration support Ruby on Rails which further enhances the developing capabilities utilizing the stack. Ruby on Rails also allows flexibility. One of the biggest fortune 500 companies, Shopify, utilizes Ruby on Rails which is also known for handling almost 80,000 requests per second. Websites like GitHub and AirBnB implemented their website with the Ruby on Rails stack.

#### 5.7.2.7 Decided Web Stack

Choosing the correct web stack is realistically the single most important step at the start of web development. Out of the many web stacks discussed and many more technologies that have not been emphasized, there are many options with their upsides and distinct downsides. With the ever-changing flow of technological advancements, web applications must correlate and support web stacks that change and keeps up with modern-day technologies. This would ensure new documents being produced, advanced resources being developed, and bigger communities supporting those technologies being

formed. However, that does not mean that the older forms of technology are not efficient and should not be used. Stacks like LAMP stack and Ruby on Rails - despite being one of the earlier web development technologies, to this day are practiced and being updated. LAMP stack was developed in 1998, but websites like WordPress, Tumblr, and Wikipedia are still built on the LAMP stack. On the other hand, popular websites like Hulu, Twitter, Github, and Shopify are still utilizing and supporting Ruby on Rails technology. However, the recent technologies and the oncoming technologies like Python - Django stack and the MERN, MEAN, and MEVN stacks are being optimized religiously due to their popular usage and easy-to-implement capabilities. More tools are being generated to support the framework, and rich libraries are introduced to enhance the developing experience. With the promising future of accommodating the unlimited capabilities of machine learning and enhancing data analysis algorithms, Python-Django can revolutionize the web development era. However, with the same language, JavaScript, implemented on the Front-End side, Back-End, and server side, stacks like MERN, MEAN, and MEVN are making it simple for developers to learn and maintain and implement more robust and optimized web applications.

For the Smart Parking system, a dynamic and visually pleasing web application is needed to cater to the targeted audience. The web applications need to support real-time capabilities, emphasizing frequent yet fluent communication between the server-side and the user end. The time frame and deadline of the project would also contribute to the web development planning since learning and implementing various technologies to support an efficient web application can pose many challenges. Thus, to eradicate the time constraints for the project, choosing a stack that implements similar technologies that are easy to implement and have redundancy across platforms between components could be one of the possible solutions. MEAN, MERN, or MEVN utilization would make the prospect to be implemented in real life. All three of these web stacks implement some variant of JavaScript frameworks across the front-end, back-end, and server-side aspects of the web application. This allows code re-usage and simplifies the learning curve for the developing stage of web applications. However, the MEVN stack has its constraints for being a moderately new technology. There are fewer resources for VueJS developed, and the community for this open-source language is not yet matured as much as some of the other front-end frameworks available. MERN stack is a compilation technology that allows code reusability along with the implementation of a really popular ReactJS front-end framework. ReactJS uses libraries like MaterialUI to improve user experience and web-to-mobile responsiveness. Using the UI libraries ReactJS allows the development of rich user interfaces. Not only for web applications but also for future implementation of mobile app features, ReactJS also has an extension developed by Facebook, which is a front-end framework for mobile devices called React native. The utilization of JSX implies the ability to write custom components catering to audiences. React also allows code component reusability, which allows the redeployment of digital objects. For marketing purposes, React also enhances Search Engine Optimization for different keywords across the web application.

Thus considering the immense benefits of utilizing the MERN stack to deploy a Dynamic web application would be the ideal choice for the Smart Parking App. This will lessen the



learning curve and lighten the workload through code reusability and with different library implementations to enrich the User Experience with unique graphical interface features.

## **5.8 Mobile Application Research**

For our project, a mobile app that shows parking availability was desired. The app can inform users of the number of spaces available per garage, level, and area in real time. The two leading mobile OS platforms for which a mobile app can be developed are Apple's iOS and Google's Android. However, a mobile app was not implemented on the final project due to time constraints.

### **5.8.1 Mobile Application Types**

There are three major types of mobile apps to choose from Native apps, Web-Based Apps, and Hybrid apps.

#### **5.8.1.1 Native Apps**

These are apps developed for use on a particular operating system, which means the app only works for that specific device. The advantages of native apps are highly customizable options, fast performance, and access to all the device features and functionalities. Among the disadvantages are the need to learn specific languages and IDE for their development, and the time it requires to build the app doubles as the same app needs to be written on each platform. For example, creating native apps for Androids requires Java or Kotlin, and for iOS, languages such as Swift and Objective-C are used.

#### **5.8.1.2 Web-Based Apps**

Web-based apps are websites made using responsive web design in which the website reacts to the user's environment based on platform, screen size, and orientation. One advantage of this approach is that the app does not need to be installed onto the mobile device because the app is accessed through the already installed web browser. Another advantage is that the code is written once. Since the same web app can be accessed from either mobile operating system, writing apps for each platform is unnecessary. One of the disadvantages is that since web apps require a browser to run, web-based apps are slower than native apps because browsers typically do not work at the same speed as native apps. Another disadvantage is that all device functionality is not fully supported.

#### **5.8.1.3 Hybrid Apps**

Hybrid apps are a combination of the two previous types. These apps are cross-platform compatible, meaning they can be installed onto iOS and Android, but their functionality is similar to native apps. Therefore, these apps do not require a web browser. The advantages are that the codebase is written once (and thus takes less time to develop), and they can access the phone's features and functionalities. One of the disadvantages of hybrid apps is that since it uses web components, everything from buttons, navigations, and transitions looks and feels different. Another disadvantage is that hybrid apps do not run as fast as native apps because of the introduction of a middle layer that the app must go through to translate web functions and elements to the native device.

#### **5.8.1.4 Mobile App Choice**

After analyzing the pros and cons of each type of mobile application, the team decided to go with the Hybrid application for the following reasons. This project's mobile app does not require any of the phone's features and functionality beyond the display, touchscreen capabilities, and internet access; therefore, the app does not need to be native. Also, writing two native apps to support iOS and Android devices would take too much time. Although the web-based app allows writing the code once, it does not allow it to be installed. Therefore, the hybrid application provides the best of both worlds. It will enable writing a single codebase for both platforms and allow the app to be installed on either platform (cross-platform). Even though a web app may not perform as well as a native app, the team believes a hybrid app will perform great for this project's objectives.

#### **5.8.2 Cross-Platform (Android/iOS) App Development Framework Options**

A cross-platform development framework is needed for this project, and there are plenty of options to choose from, such as React Native, Flutter, Xamarin, Ionic, Cordova (formerly PhoneGap), JQuery Mobile, NativeScript, Swiftic, Corona SDK, Appcelerator Titanium, and Sencha. Due to the vast number of options, this research will focus on these four popular frameworks: Xamarin, React Native, Flutter, and Ionic.

##### **5.8.2.1 Ionic**

Ionic is an open-source software development kit (SDK) for hybrid mobile app development, and it was built on top of Apache Cordova and Angular JS. It was later rebuilt as a set of web components to allow developers to choose from different user interface components from Angular, React, or Vue JS. Besides being free and open-source, one of the main advantages is its extensive choice of user interface (UI) and quick prototyping; these ready-made elements speed up the construction of a graphic user interface (GUI) [4]. Another advantage is its documentation; with more than five million mobile apps built with Ionic, their exhaustive documentation is available on its website [5]. In addition, a strong online community of more than five million developers in constant activity on the forum makes it easier to find help. One of the disadvantages is the absence of hot reloading, which would allow changes to an app to be applied without reloading the whole app. Instead, the app refreshes the entire app to make changes active [4].

##### **5.8.2.2 Flutter**

Google created Flutter as an open and free framework to develop native Android and iOS applications with a single codebase. Unlike Ionic, which uses HTML, CSS, and Javascript, mobile apps in Flutter are written using Dart (Google's programming language) [6]. One of the advantages of Flutter is that it does hot reloading, which allows for changes to the app to be seen immediately [6]. It also has full customization and fast rendering, which permits graphics, video, and text animation without limits. Another advantage is that it has an app builder; this tool allows one to write code using building blocks that can be mixed and matched to suit any needs [7]. One of the disadvantages is that Flutter is still a relatively new framework because it has not been in the market for as long as other frameworks; this translates into a lack of some advanced features and a vast

resource base. Another disadvantage is learning a new programming language, Dart, to write mobile apps in Flutter. Since Dart is not a popular language, it could be a disadvantage since the online resources are not as extensive as other programming languages.

### **5.8.2.3 React Native**

Meta Platforms, then Facebook, created React Native as an open and free framework to develop Android, Android TV, iOS, macOS, tvOS, Web, and Windows applications. React Native applications are written in React, which is coded in Javascript. One of the advantages of React Native is its countless ready-made solutions and libraries; this enhances the development process for quick prototyping [8]. Another advantage is that the developer community and the online support are large, which allows for getting help quickly. In addition, the use of Javascript opens up a vast online community that also offers support to React Native developers. One of the disadvantages is that it can be hard to debug, and developers need knowledge of the native language of each platform [8]. One more disadvantage is that 90% of the code can be applied to iOS and Android platforms; therefore, depending on the native elements used, it may require separate optimizations for each platform.

### **5.8.2.4 Xamarin**

Owned by Microsoft, Xamarin is a free cross-platform and open-source app platform for building Android and iOS apps with .NET and C#. .NET is a developer platform made up of tools, programming languages, and libraries for building many different types of applications and Xamarin extends the .NET developer platform with more tools and libraries [10]. One of the advantages of Xamarin is its single tech stack. Due to its compatibility with the .NET framework, .NET is used to develop the app, but it is also used to develop its backend server; this advantage could potentially cut development time since the same .NET framework is used for both[9]. Another advantage is the close-to-native performance of the apps. Since Xamarin apps are compiled for native performance, the app can access all the features and functionality of the platform and device and platform-specific hardware acceleration. On the other hand, one of the disadvantages of Xamarin is that it is not suitable for heavy-graphic apps. If the app requires intense user interaction or relies a lot on appearance, then the app will require considerably more time to develop as this requires advanced knowledge of iOS and Android native technologies.

### **5.8.2.5 Cross-Platform App Development Framework Choice**

After analyzing the pros and cons of each development framework, the team decided to go with React Native for the following reasons. The main reason is that React Native uses ReactJS at its core, and two team members have experience writing web applications in ReactJS; this dramatically reduces the learning curve. Although Flutter is more than capable of handling the project's mobile app requirements, learning Dart as a new programming language would considerably increase the time to develop the mobile app. Xamarin is a great choice, but since none of the team members have worked with C# or .NET, additional time would be spent learning the language. Ionic could have been chosen; however, to match the team's decision to use the MERN stack, React Native was

a clear choice. In addition, since the web application will use the MERN stack, the same web server can accommodate the APIs needed to serve the React Native mobile application and the Web Application.

## 5.9 Web Server Research

The parking garage system needs a place to store the information it receives from the various video cameras throughout the parking garage from which web app and mobile app clients can get their data. A database is the best solution to store this type of data, and a web server is the best place to host the web app and the mobile app files, which facilitates communication with the database.

Since the MERN stack was selected as the software development technology for this parking garage system, as discussed in the web development stack section, what is left to discuss is where to host the database and the web server. There are two main options for hosting the backend and frontend of the app; it could be hosted in a local server or the cloud through a service provider. However, to maintain a cost-efficient solution, choosing a cloud service provider would be the best option, financially speaking. A local server capable of supporting hundreds of connections from students' phones could quickly escalate to thousands of dollars in upfront costs associated with buying all the parts needed to build a server, as opposed to a cloud service provider that offers to host plans of less than a hundred dollars per month. In addition, when the web server reaches its capacity, a cloud webserver can easily be upgraded with a few clicks.

As mentioned before, the web app was not implemented in the final prototype; therefore, there was no need for a web server.

### 5.9.1 PaaS vs. IaaS

Platform as a service (PaaS) and Infrastructure as a service (IaaS) are two options to choose from when selecting a service provider. The infrastructure as a service (IaaS) provides access to servers on the cloud; these physical or virtual servers are fully managed and maintained by the app developers or by IT staff. Therefore, knowledge of server operating systems, installation and maintenance, storage capacity and speeds, and CPU speeds maybe be needed when choosing the right plan. For example, among the features to choose from are storage devices such as SSD or HDD, amount of RAM such as 1GB or 8GB, Linux distribution such as Ubuntu or Fedora, and shared or dedicated CPUs. One of the benefits of IaaS is that businesses can purchase resources as needed instead of buying hardware upfront [11].

On the other hand, PaaS provides the necessary software components to deploy an app without managing the servers where apps run [11]. The server provider takes care of the virtualization of servers, so resources are easily scaled up from the developer's point of view. Therefore, it is an excellent option for developers that do not have server knowledge or do not want to spend time managing the servers.

For the parking garage app, either of these two options would work because, fortunately, one of the team members has IT experience and can manage an IaaS plan if chosen as a

cost-effective solution. In addition, most team members have some Linux installation experience in virtual environments, as some of our courses required this knowledge to complete a few assignments.

## **5.9.2 Web Server and Database Hosting Providers**

Many cloud service providers offer Web server hosting plans; some popular ones include Digital Ocean, Heroku, Microsoft Azure, Google Cloud, Amazon web services, and MongoDB Atlas. Each service provider is explained in the following sections, and its pros and cons are analyzed.

### **5.9.2.1 Digital Ocean**

Digital Ocean is a simple and scalable cloud platform for all developer needs. It provides services such as infrastructure as a Service (IaaS), Cloud-Native (managed Kubernetes), and Platform as a Service (PaaS) for all computing, networking, storage, and database needs. In their IaaS plans, Digital Ocean provides one-click virtual machines called droplets. A droplet is a virtual server with a Linux-based distribution already installed and configured; this is a good option because it reduces the time for the OS installation. Their plans include a shared CPU server or a dedicated CPU server. The shared CPU is where one physical server contains many virtual servers, each hosting other people's backend and frontend applications; thus, one CPU is shared among many virtual servers. The dedicated CPU plan means a physical server is dedicated to hosting only one backend/frontend application. The shared CPU plan is their more cost-effective option, ranging from \$5 per month to \$96 per month, while the dedicated CPU cost ranges from \$60 per month to over \$2000 per month. Digital Ocean offers database hosting that includes MongoDB (part of the MERN stack) starting at \$15 per month, but it also offers the option to connect to a MongoDB database hosted somewhere else.

In their PaaS plans, Digital Ocean offers a basic plan for \$5/month and a professional plan for \$12/month. The basic plan is recommended for prototyping only, and the professional plan is for the production stage.

### **5.9.2.2 Heroku**

Heroku is a cloud platform as a service (PaaS) supporting several programming languages such as Node.js, Ruby, PHP, and more, allowing companies to build, deliver, monitor, and scale apps. Apps in Heroku run inside smart containers, called dynos, in a fully managed runtime environment; Heroku handles these containers, so everything from configuration, orchestration, load balancing, failovers, logging, security, and more, is handled by Heroku. Therefore, app developers can entirely focus on developing the app frontend and backend. One advantage of Heroku is that it has GitHub integration, which means that a repository can be set up to auto-deploy with every push to a branch, effectively reducing the deployment time of every new code change. Heroku offers a free plan to try Heroku with no commitment; in this plan, the container sleeps after 30 minutes of inactivity. The Hobby plan for small non-commercial apps costs \$7 per month and includes an always-on container, a free secure socket layer (SSL), and automated certificate management. The standard plans range from \$25 to \$50 per month, including threshold alerting. The Performance plan for high-traffic applications ranges from \$250

to \$500 a month, including deploying the app in different regions globally. Although Heroku offers database hosting, they do not offer MongoDB hosting; instead, they make it easy to connect the app to an existing MongoDB hosted elsewhere.

### 5.9.2.3 MongoDB Atlas

MongoDB Atlas, a database-as-a-service (DBaaS), is a cloud computing service. The database is stored in AWS, Microsoft Azure, or Google cloud platform, but Atlas provides the management tool in a centralized and convenient dashboard in a web app. Users do not have to handle the setup of hardware or software installation; the service provider handles everything related to managing the database. One of the advantages of MongoDB Atlas is that it allows accessing and manipulating the data programmatically; therefore, the data can be edited by the backend server hosted somewhere else. They have essentially main plans. Their share plan offers a free option, a \$9 per month option, and a \$25 per month one. Their dedicated plans are charged by the hour and vary depending on where the files will be hosted (i.e., AWS, Azure, or GCP), but they range from \$0.08 per hour to \$33.26 per hour.

### 5.9.2.4 Microsoft Azure App Services

Azure app services is a platform-as-a-service product to host web and mobile apps' frontend and backend code. It works with .NET, .NET Core, Node.js, Java, Python, or PHP programming languages. It offers Azure Cosmos DB API for MongoDB, making it easy to leverage Mongo databases by pointing the application to the API for the MongoDB account's connection string. One advantage of using Azure is that it offers continuous integration and deployment, where the app is automatically updated and redeployed whenever there is a change in the code repository, such as GitHub. Azure App Services offers from free to isolated plans (i.e., dedicated servers). The app sits on a shared server, with 1GB of storage in the free plan, but it does not support a custom domain. The next plan is the Basic; it includes a dedicated server created for development and testing. This plan supports custom domains and costs \$12.41 to \$48.91 per month. The production plan is the Standard service plan and goes from \$69.35 to \$277.40 per month.

### 5.9.2.5 Google Cloud Platform

Google offers two options for app deployments. One of them is Cloud Run, a managed computing platform to run containers invocable via requests or events, and it supports many programming languages such as Go, Python, Java, .Net, Node.js, and more. It is serverless, which abstracts away all infrastructure management so that developers can focus on the app itself. Cloud Run has a pay-as-you-go where it charges for CPU, about \$6.75 per month, memory, about \$6.75 per month, and request usage, about \$0.40 per million requests. It must be noted that the first 50 hours of CPU, the first 100 hours of memory usage, and the first 2 million requests are always free. In addition, new customers get \$300 in free credits to spend on Google Cloud during the first 90 days.

Another option offered by Google is the Google App Engine (GAE), a platform-as-a-service product that provides web app developers access to Google's scalable hosting in Google-managed data centers. It is similar to Cloud Run, but it is a bit

easier to deploy an app in App Engine than in Cloud Run. Cloud run provides more options that can easily complicate an app deployment for the inexperienced developer. The App Engine pricing includes charges of \$0.05 to \$0.30 per hour per one container instance; there are charges of \$0.12 per gigabyte of outgoing network traffic.

#### **5.9.2.6 Amazon Web Services (AWS)**

AWS is a subsidiary of Amazon that provides on-demand cloud computing platforms. It offers PaaS and IaaS solutions suitable for the parking garage system. As part of the IaaS, Amazon Elastic Compute Cloud (Amazon EC2) offers virtual servers. Amazon offers AWS Elastic Beanstalk for the PaaS, an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on servers such as Apache, Nginx, Passenger, and IIS.

The Amazon EC2 offers many options, including a web server but not a database. The on-demand plan price is computed by the hour or the second; for example, the most basic option starts at a \$0.052 hourly rate, which translates to \$37.44 a month if the server is used 24/7. The spot instance plan, which allows the use of spare computing capacity for up to 90% off the On-Demand price, was designed for flexible start and end times applications. Unfortunately, it will not work for this project since the app must be available 24/7. The savings plans, which allow discounts based on a 1 to 3-year commitment, will not work for this project because the service will be used for a few months rather than a few years. The dedicated plans offer a physical server only for an app, which means no other apps are running on the same server (not shared). The pricing varies depending on the server's performance, but it can range from \$0.45 to \$67.00 per hour.

The AWS Elastic Beanstalk has no additional charge for its usage. The cost is associated with the AWS resources needed to store and run the application. For example, the cost of Amazon EC2 instances or the cost of S3 buckets for storage used for the app would be the only cost for the hosting of the webserver. Therefore, AWS Elastic Beanstalk is a simple way to get web applications up and running on AWS.

AWS services offer one more option: the free tier designed to gain free, hands-on experience with the AWS platform, products, and services. It includes 750 hours per month of Amazon EC2 for 12 months, among many more perks. The 750 hours is equivalent to 31.25 days per month, so an EC2 instance can be left running 24/7 for a year before incurring any cost.

#### **5.9.2.7 Web Server and Database Hosting Provider Summary**

A summary of the cost associated with the analyzed service providers relevant to this project is provided in table 14 below.

**Table 14:** Web Server and Database hosting plan and Pricing

<b>Provider</b>	<b>Web server hosting (Monthly)</b>		<b>Database Hosting (Monthly)</b>	<b>Total cost (Monthly)</b>
<b>DigitalOcean (IaaS)</b>	Basic: \$5	Next: \$10	n/a	\$5 to \$10
<b>DigitalOcean (PaaS)</b>	Basic: \$5	Pro: \$12	n/a	\$20 to \$27
<b>DigitalOcean Database</b>	n/a	n/a	Basic: \$15	\$15
<b>Heroku (PaaS)</b>	Basic plan: \$0 to \$7	Standard: \$25 to \$50	---	\$0 - \$50
<b>MongoDB Atlas</b>	n/a	n/a	Shared plan \$0 to \$25	\$0 to \$25
<b>Amazon EC2 (IaaS)</b>	Free tier: \$0	On-Demand: \$37 to \$ 158	n/a	\$37-\$158
<b>Google Cloud Run (PaaS)</b>	Free tier: \$0	After free tier: starting at \$15	n/a	\$0 - \$15
<b>Microsoft Azure App Services (PaaS)</b>	Free tier: \$0	Standard \$69 to \$277	n/a	\$0 to \$277

#### 5.9.2.8 Web Server Hosting Provider Selection

After a team discussion about all the analyzed providers, the team arrived at the following conclusion. Microsoft Azure App Services, Google Cloud Platform, and Amazon Web Services seem somewhat more complicated to deploy apps. Due to their excellent scalability and vast additional options would be great for app deployment that requires a more robust solution that can be spread out in multiple regions and guarantee 99.999% uptime reliability. Although all three options offer free tiers, their pricing seems more complicated. In addition, none of the team members have experience deploying or using any of the three services; this can increase the learning curve for deploying the web and mobile app. Digital Ocean seems to be a great option; their one-click deployment of virtual servers with the Linux-based OS already installed makes app deployment simple. In addition, one of the team members has previous experience deploying this kind of virtual server using Digital Ocean droplets.

On the other hand, Heroku provides an excellent, easy-to-use app deployment that includes the automatic update and redeployment of the app with every push to the app's code repository. Also, the free tier or the \$7 per month cost makes it straightforward. Some team members also have experience using Heroku in previous university projects, which reduces the learning curve time. For these reasons, the team decided to use the Heroku platform to host the frontend and backend of the parking app and MongoDB Atlas as the database hosting provider. However, at the end of the project, the web app



feature of the parking system was not implemented and the selection above was not disregarded.

## 6.0 Related Standards

Standards are documented to provide consistency and compatibility for products and efficient production processes as well as for safety for consumers. Such standards act as building ground or base guidelines for product development. But with these advantages, standards can also limit development options. These standards could be implemented by the government or any other companies that produce the initial products. Following certain standards often are mandatory, and there are also soft standards that are optional. However, considering all the standards in the development process can ensure safety in the developmental process.

### 6.1 OSHA Standards

Occupational Safety and Health Administration is responsible for creating workplace standards for development safety. They document procedures for workplace safety, from construction industries to electrical regulations. In the smart parking project, there would be advanced work in the mounting and installation process and electrical work for the PCB boards and the main control unit. Following the OSHA standards would ensure safety and decrease the risk of development accidents. Some of the instructions are:

- Inspect the electrical equipment before use and locate equipment that is not in the most optimum condition.
- Make sure all the electrical equipment is connected with a ground wire before being used.
- Avoid wet areas and keep the electrical equipment from any liquid spill.
- All the exposed metal parts need to be grounded before operation.

Exposure to current flows directly could have mild to severe reactions while in contact. This depends on the duration of contact and the electrical flow. Electric flows up to 5 milliamps can have a faint reaction in the human body. Up to 30 milliamps of electrical flow contact can cause painful shock in the human body. Coming directly in contact with up to 150 milliamps of electricity can cause severe pain and sometimes, depending on the contract duration, can also cause respiratory arrest. From 1 amp to 4.5amps current can cause the heart to cease and serious nerve damage. Electric flows of more than ten amps can cause cardiac arrest and, in the worst cases, death.

Exposure to direct current flow can cause severe consequences. Thus following the standards of OSHA is crucial for the smart park project. Working with the PCB or the control unit would make the team come in contact with metal - electricity-driven devices. So following OSHA standards would allow the creation of safety barriers and working habits during the development process of the system. While handling or working with high current flows, following these standards can prevent any possibilities of major accidents during the senior design one and two durations. It is also advised by the authors

of the standards that while working with equipment accessing high voltage of current flows, to work in groups. This would enable quick access to help in case of emergencies. Make sure the ground connections for the electrical devices are steady through the end of the connection. This would help ensure the ground connection to avoid short-circuit incidents.

## 6.2 Data Communication Standards

IEEE standards for data transmission and communication can be used as a base guideline for the smart park project while establishing a communication network between the local server and the cameras, and the web application. IEEE 802.11 standards establish communication standards for WLAN architecture and specifications.

In [21], the IEEE documentation can be found, and the depiction of the standard max data transmission rate is added in the figure below,

**Table 15:** Data rate based on IEEE 802.11

Key Standards	Max Rate	Spectrum (U.S.)	Year
802.11	2 Mbps	2.4 GHz	1997
802.11a	54 Mbps	5 GHz	1999
802.11b	11 Mbps	2.4 GHz	1999
802.11g	54 Mbps	2.4 GHz	2003

This standard documentation 2 Mbps speed with 2.4GHz ISM bands. As a replacement for a wired network, this implementation could be the best alternative. When implemented along with TCP/IP, this network system works extremely well in the mid-range with the most efficient and optimized transmission speed.

### 6.2.1 Ethernet Standards

For computer systems to communicate over a network, they follow a conceptual model known as the Open Systems Interconnection (OSI). The OSI model outlines seven different layers in the connection between two computer systems. These layers are physical, data link, network, transport, session, presentation, and application.

The ethernet connection standards addressed by IEEE's 802.3 documentation define how wired ethernet pertains to two of these layers the physical layer and the data link layer's media access control (MAC) of wired Ethernet. The physical layer specifies electrical signals, signaling speeds, media and connector types, and network topologies. This layer also implements the ethernet physical layer portion of data transmission speeds, including 1000BASE-T (1000 Mbps), 100BASE-T (100 Mbps), and 10BASE-T (10 Mbps). The data link layer specifies how communications occur over the media and the frame structure of messages transmitted and received. In other words, this layer dictates how the

bits come off the wire, and the arrangement that they will have such data can be extracted from the bitstream. In ethernet applications, this is called media access control.

The connection between ethernet components and a media access controller can be accomplished through the media-independent interface (MII). The MII is standardized by IEEE's 802.3u and allows for different types of ethernet PHYs to be connected to any MAC. There are many variations of the MII, including the Reduced MII, Gigabit MII, Reduced Gigabit MII, and the Ten Gigabit MII; however, we will be using the reduced MII as this is the only interface supported by our MAC block on our MCU. The signals used to accomplish this interface are detailed in section 8.6. In the table added (table 16), the different standards for ethernet wires have been pointed out for convenience.

All Ethernet cables use the same standardized RJ45 connectors; however, the wiring configuration inside the cable can differ from cable to cable. There are two common standards for ethernet wiring called T568A and T568B, as shown in Table 16. There are four pairs of wires in each standard, commonly referred to as twisted pairs. These standards are similar, as the only difference between them is the green and orange pin assignments are swapped. With this said, it does not matter which of these is used in our system as long as the wiring standard is the same on both ends of the cable.

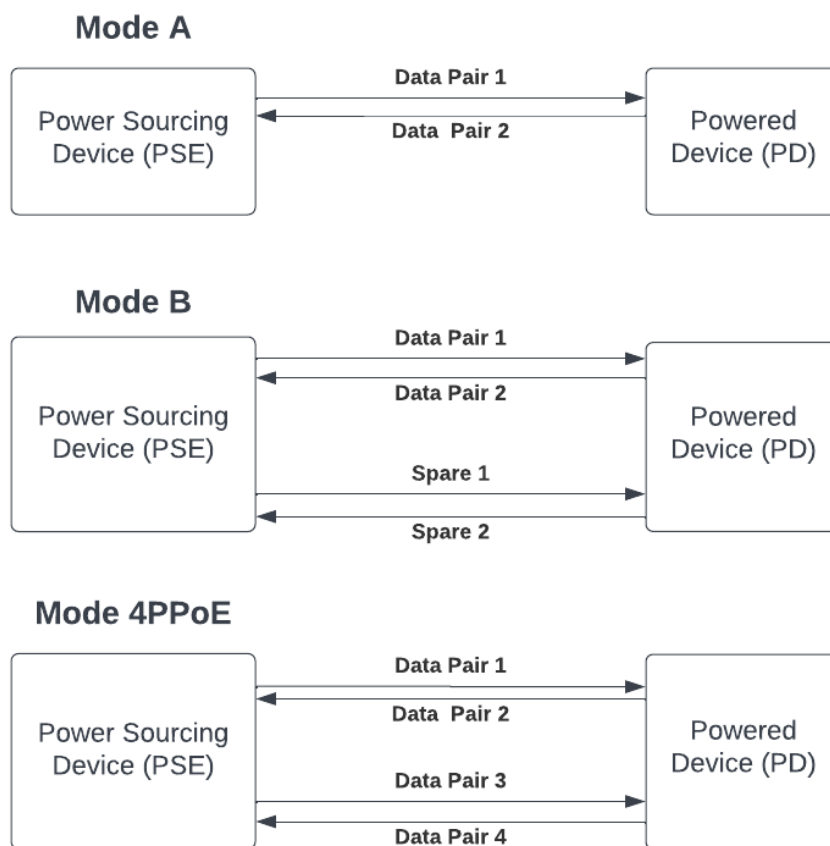
**Table 16:** Ethernet Wiring Standards

	T568A Wiring			T568B Wiring		
Pin	Pair	Wire	Color	Pair	Wire	Color
1	3	1	White/Green	2	1	White/ Orange
2	3	2	Green	2	2	Orange
3	2	1	White/Orange	3	1	White/ Green
4	1	2	Blue	1	2	Blue
5	1	1	White/Blue	1	1	White/ Blue
6	2	2	Orange	3	2	Green
7	4	1	White/Brown	4	1	White/ Brown
8	4	2	Brown	4	2	Brown

The standard that defines how a single cable can provide both data and electric power is known as IEEE 802.3af. Any port designed for Power over Ethernet applications should provide up to 15.4 W of DC power (minimum 44 V DC and 350 mA). Three techniques are used to accomplish this, as shown in Figure 14. The first set of blocks in the figure is known as Mode A, and this mode transports power on the same wires that are

transporting data. This is made possible by applying a common voltage to each pair. Since the twisted-pair ethernet wires use different signaling, applying this voltage does not interfere with data transmission. Using this method only allows for 10BASE-T and 100BASE-T speeds; however, this will not be an issue for our system.

The second set of blocks is known as Mode B, and this mode uses the extra two twisted pairs to transport power, while the first two pairs are only responsible for data. This is effective as two twisted pairs can be treated simply as V+ and V-. The final set of blocks is known as Mode 4PPoE, and this mode uses all four twisted pairs to transport power and data. This mode is not commonly used as it only applies to high-power applications. A summary of the PoE modes is shown in Figure 14.



**Figure 14:** PoE Connection Standards

### 6.3 Programming Standards

Despite not affecting the actual functionality of the code or the functionality of the system in general, following programming standards can help developers stay organized. The standards include naming standards, bracing standards, formatting, and syntax standards. These standards help the codebase stay organized across platforms. This also improves the readability of the code base, which helps anyone outside the developer team

to glance at the code base and understand the business logic of the code. The object-oriented implementation of the code is also enhanced as these standards bring portability to the codebase. The maintainability is optimized; thus, it helps the developers with the debugging process. However, an even bigger advantage is adding new features or advancements to the overall system; following a standard code system can make the integration process extremely smooth.

### 6.3.1 Programming Naming Standards

Naming standard is one of the biggest advantages of the programming standard process. These processes include camel case methodologies. There are three types, and they are the initial uppercase, the lowercase, and the consistent case. These naming conditions simplify the naming process across the code base and define the purpose of numerous variables defined. To understand certain parts of the code, consistent variable naming allows developers to keep track of multiple usages. Some examples of these naming standards are listed below,

Consistent Case - `CONSISTENT_CASE`

Upper Camel Case - `UpperCamelCase`

Lower Camel Case - `lowerCamelcase`

### 6.3.2 Programming Syntax Standards

Syntax standards include consistency in white spaces, comments, line length, etc. This allows for improved readability and reusability in the code. A line worth of space between different modules within the code would increase the portability of the codebase. Between multiple business logic codes, vertical single-line spaces can help developers distinguish different parts of the code easily.

Comments are one of the main indicators to communicate the purpose of the code to different developers or even to skim through the code base with speed. Commented each code block would increase general context understanding for the readers who are not part of the code developing team. However, following a specific standard while commenting is important. The standards could include the usage of a single “//” character at the end. Also, starting each code block with a single line comment of 50 characters max can allow code blocks to be structured. Avoiding excessive white spaces while commenting should also be included within the standards.

Having a set max length along the codebase allows the integration of smooth structure across. This standard allows no logic to exceed the width of normal devices. Developers will have efficiency while browsing through or debugging code with these implementations. This standard makes it possible by wrapping the line after a set amount of characters. A generalized example would be wrapping a line of code to another line after it exceeds a maximum of 100 character count.

### 6.3.3 Programming Indentation and Bracing Standards

Indentations and bracings are two of the most vital components of keeping the codebase neat and organized. Braces are used frequently in code statements for logic implementations. However, even though, in some cases, braces are mandatory, some “if” statements do not require braces. However, as standards, for any type of business logic or driver-level logic, braces covering code blocks should be used. It helps developers keep track of the start and end of existing code blocks.

Indentation is another important concept for organizational purposes. After an if-statement or any loops within them, the code needs to be indented by one tab worth of space. Every time there are embedded loops within, the spacing for indentation should be consistent and increase by multiples of 1 depending on the level of depth within the logic. Without the consistent standard implementation of indentation, the code blocks would become difficult to read.

### 6.4 Ingress Protection Code (IP Rating)

The Ingress Protection or IP rating code classifies the protection provided by an enclosure for electrical equipment against intrusion, dust, contact, and water [27]. IP ratings are defined in the International Electrotechnical Commission (IEC) number 60529 (also published by the European Union by CENELEC as EN60529) [27]. The two numbers that come after IP have two meanings (e.g., IP65). The first digit refers to the solid particle intrusion protection. This number is required and indicates the level of protection of persons against access to hazardous parts inside the enclosure and protection of the equipment inside the enclosure against the ingress of solid foreign objects. The meaning of the different options as the first digit is summarized in Table #. The second digit is optional and indicates the level of protection against water ingress, and each level is outlined in Table 17 and 18. The IP rating for the parking system was chosen to be at least IP65 which provides dust-tight protection and protection against water jets.

**Table 17: IP Code First Digit Meaning**

Level	Effective Against
X	X means no data available
0	No protection
1	Large surface of the body
2	Fingers or similar objects
3	Tools, thick wires, etc.
4	Most wires, screws, large ants, etc.
5	Dust protected
6	Dust-tight

**Table 18: IP Code Second Digit Meaning.**

Level	Effective Against
X	X means no data available
0	No protection
1	Vertically dripping water
2	Dripping water when the enclosure is tilted 15 degrees
3	Spraying water
4	Splashing of water from any direction
5	Water jets (12.5 liters/min at 4.4 psi)
6	Powerful water jets (100 liters/min at 15 psi)
6K	Powerful water jets at higher pressure (75 liter/min at 150 psi)
7	Immersion (up to a meter)
8	Immersion (more than a meter)
9k	Powerful high-temperature water jets

## 6.5 Voltage and Power testing Standards

Safety is one of the major components to consider while working with electrical tools with higher voltage and power. Most corporate companies who work with PCB designs and microcontroller manipulation often have their own set of rules for a safe environment for working with electronics. In [31], the IEEE Std 510-1983 has focused on the guidance for working in a safe environment while working with electrical equipment. The paper then goes in-depth with the definitions and usages of rubber protective tools, safe work practices while dealing with electrical tools, field tests, special concerns working with current flow, safe electrical equipment, and many more.

As far as protective equipment goes, insulated rubber equipment can provide a lot of core safety while the machine is energized. During the process of switching equipment, one must use insulated equipment. Because the machines may still be energized at the time of the switch. Also, while the equipment is being turned off or de-energizing, it should be considered energized, and caution must be taken. National Consensus Standards NCS is mainly responsible for employing and publishing safety protocols for Rubber - Protective types of equipment. All the rubber-insulated equipment must be National Consensus

Standards approved. Such equipment could include rubber-insulating gloves, matting, blankets, hoods, line hoses, and sleeves.

During laboratory or research implementation work - multiple safety protocols are mentioned to be followed. Having permanent and temporary test areas, interlock or fail-safe signaling systems, grounding protocols, and power supply safety are just some of them. The main or permanent test area should be enclosed with some sort of barrier with doors or some type of blocker to restrict public access due to potential power hazards. At the same time, caution boards must be utilized to warn individuals working closely. As far as temporary testing areas go, they do not need permanent barriers. However, there should be some sort of temporary separators like portable fences. Safety tapes easily visible and at least waist high should enclose the parameter. Only once the equipment is turned off and fully de-energized can the tape be removed for others to enter the area.

Access to a loudspeaker will aid in communicating with many individuals simultaneously. For fail-safe purposes, some sort of signal or interlock system can also be put in place. Any type of audible sound or gesture can help enhance the safety net. For the main and temporary test areas, a systematic procedure should be put in place. This will ensure that the equipment is not energized while someone else is nearby. In critical cases, companies have been known to select observers to ensure the test areas are safe for other individuals to engage in.

With the implementation of workspace and workbench safety comes the concept of work equipment safety. Without the equipment thoroughly checked for safety measures, any type of workspace area precaution would be invalid. These equipment safety measures can include special and basic grounding practices, safety in power supplies, being cautious of any physical hazards, and many more. Before working on any test object, it should be a mandatory requirement to ground any type of high-voltage circuit. This can save any type of personnel hazards. For extremely high-voltage equipment, operators usually attach a ground to the high-voltage terminal using an insulating material. All the exposed intermediate terminals with running electricity should be grounded. Also, grounding instruments should be checked for efficiency and safety, and this procedure should have more precedence over proper signal grounding.

For the direct voltage power supply, all the individuals working nearby need to be notified as there could be scenarios of random discharge of electricity in underground metallic objects and capacitive elements. The equipment nearby also should be grounded and short-circuited before initializing the direct voltage source. Also, after usage, the equipment should be allowed some time for the test voltage to decay to a low enough value so that other individuals can safely access the area. With these, caution should also be followed for airborne porcelain or any other material in case of equipment failure. Having any sort of open container should also be prohibited as it may cause physical injuries in case of spillage.



The Smart Parking system was designed to have the PCB set to support the control unit and LED signs. Thus various electrical equipment was needed for the proper implementation of the system. To ensure a safe environment during the implementation and the test procedure, the safety precautions mentioned in IEEE's Std 510-1983 standard can indeed provide a multitude of guidelines. Following these strategies, safe execution of the electrical component testing, as well as the application of the Smart Parking System, was achieved.

## **6.6 NEMA Ratings for Enclosure Standards**

National Electrical Manufacturer Association (NEMA) signifies and enforces protocols on the environment where the electrical component resides (fixed enclosures) and their stability. The association standardizes the quality of the fixed enclosures used for storing electrical components and tests the ability to withstand certain environmental calamities. NEMA not only helps the end-users to achieve protection of a safety net to keep the electrical components enclosed, but they also ensure the economic aspects of the enclosing are subpar. They aid in improving communication safety and economics between the end user(purchaser) and the seller (manufacturer).

NEMA standardizes and supplies an ample collection of enclosures. IP Enclosures, ICEx Enclosures, ATEX Enclosures, UL Listed Enclosures, C - UL Enclosures, and many more. The enclosures are all powder-coated carbon steel, 304 stainless steel, 316 stainless steel, or the aluminum enclosures of NEMA are all supported and verified by the UL or IP ratings. There is a selection of NEMA standards that ensure all the necessary capabilities for the safety of the enclosed electrical components.

NEMA Type 1 standard ensures protection of the electrical components against solid falling dirt. The Type 2 standard enhances the safety of minor water splashing and dripping as well. Type 3 of the standard protects the enclosures from windblown dirt, rain, and snow. This standard specifies the safety features for the components even against the external formation of ice. The Type 3X standard allows standardizing the enclosing equipment to be sturdy enough to fight against corrosion and stay undamaged by any external formation of ice. Standard Type 4 enforces stronger safety nets by adding security to the enclosure products to keep the electrical components operable even against external hose-directed water at a strong force. Type 4X improves the Type 4 standard with the capability of working against corrosion. Type 5 standard allows protection against more fine-grained objects that may affect the components. Such objects include dust, lint, fibers, and other flying debris. Type 6 standard states that the enclosures must provide safety of the components from external submersion for a limited time at a limited depth. In the Type 6P standard, this safety net gets improved with safety against prolonged external submersion of the enclosure.

NEMA Type 7 and the later standards also focus on safety outside and inside the enclosures. Standard Type 7 states that the enclosures must contain any internal explosions without causing any external hazards. Type 8 of the standard is stated to protect the components by preventing any possibilities of combustion through the use of oil-immersed equipment. Type 9 standards prevent the possibility of combustion through

airborne combustible dust. Type 12 standards and the Type 12K standards are specified to provide the safety features mentioned above, however, with the additional emphasis on being constructed without a knockout. Type 13, the latest standard of NEMA, is introduced to add an extra layer of protection for electrical components. The standard protects against dirt, circulating dust, and any airborne particles like dust, lint, and fiber. As well as water dripping, splashing, and spraying. However, it adds a layer of security by adding security against oil and non-corrosive coolants as well.

NEMA standards provide safety measures and standards to shield electrical components and systems from any environmental (external) calamities, as well as offer protection from external hazards from internal combustion and explosion. The Smart Park system has multiple crucial electrical components, such as the POE cameras, PCB boards, and the Control Unit. These components are a vital part, in fact, the heart and soul of the system. Thus protective measures must be taken to ensure the safety of these components. Following the NEMA guidelines and standards can allow the developers to achieve safety and maintenance for the electrical components.

## **6.7 CPSC Standards**

Consumer Product and Safety Commission is a government agency that ensures safety in product manufacturing. The commission is responsible for issuing safety standards for all merchandising products. From children's safety for walkers and cribs to safety standards for writing instruments and other materials - all are standardized by this corporation. These standards do not specifically enhance safety in the workspace or the workbench. However, these government-regulated standards act as primary guidelines to help consumers with the safe handling of all the necessary products.

Poison Prevention Packaging Act enacts upon a set of standards and regulations that tread closely with safety in packaging for the end users (buyers) from different materials. While product parts for any project are delivered at home, the packaging materials must be safely maneuverable. Some health constraints may come from the materials of the packaging substances. The standard paper also focuses on the safety of children from the packaging materials. With an emphasis on child-resistant packaging, this standard ensures that any of the packaging materials are manufactured, keeping consideration of physical hazards. Ensuring all the components of the electrical system are CPSC-rated is an important step to ensure safety in product safety and packaging. This way, the developers can safely handle the delivered packages at home, and even consumers can get a sense of security while unboxing and utilizing the devices. Batteries are one of the more common components that the paper emphasizes.

CPSC also implies electronic products being shipped packages approved by its 1700.15 section, which standardizes poison prevention packaging standards. Since in batteries, low-viscosity hydrocarbons can be found, they must abide by the 1700.15 a and b packaging section regulation to protect the consumers from any type of physical hazard during or after the delivery process. The section also mentions the packaging going through Younger - Adult tests and Senior - Adult tests for scoping purposes. In the Younger-Adult test, the packaging is tested for child resistance with at least 80 percent

effectiveness. Whereas in the Senior-Adult testing, the packaging goes through easy opening methods with 90 percent senior adult panel test.

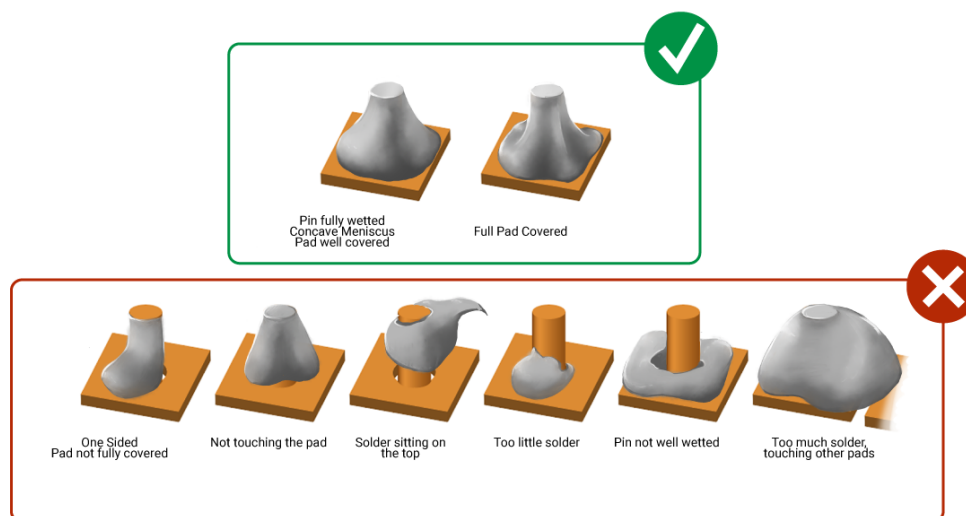
For Smart Park System, all the parts were ordered online, and thus, making sure that the products, as well as the parts, were CSCPSC PC rated was crucial. Since the products were delivered directly at home, the packaging safety measures were monitored closely for the safety of family members and nearby individuals. The CPSC regulations can provide enough context in safety for product and packaging safety.

## **6.8 Soldering Standards**

Joint Industry Standard (J-STD-001) is the current state-of-the-art industrial specification for the grouped assembly of electronics and electrical product classes. Institute of Printed Circuits regulates the usage of J-STD-001 standards for the soldering process of manufacturing PCBs. There are a lot of training programs to instantiate the J-STD-001 soldering and manufacturing process. Released in 1992, the J-STD-001 A version was the pioneer of global soldering regulation, and now the latest standard document is at J-STD-001 H. This standard initializes and defines different materials and production processes of soldering for stability, reliability, and quality in solder joints and assembly. The standard initializes material requirements, soldering requirements, component placing guidelines, rework conditions, assembly inspection, connector details, and many more.

The general provisions of soldering of J-STD-001 are the first steps towards a proper execution of the soldering process. To prevent contamination of materials and surfaces, J-STD-001 emphasizes the cleanliness of the workbench and the tools. These inspections for cleanliness need to be done even before the conformal coating and stacking applications. It also specifies abiding by the manufacturer's rules for the environmental heating or cooling rates. The stacked and multi-layer chip capacitors are treated as thermal shock-sensitive. This allows protection against thermal excursions. The soldiers also must wet the tinned areas of the wire. This aids in the strands of the wire not being damaged. It also emphasizes the defect during the soldering process and differentiates the acceptable mistakes from those that must go through a rework process.



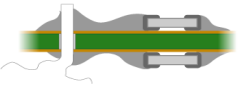

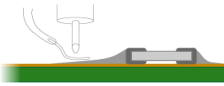
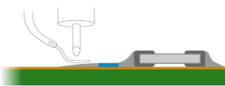
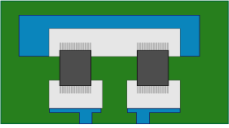
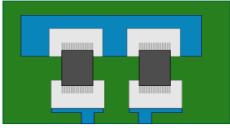
The proper welding demonstration is shown in Figure 15 [32]. Visually noticing the soldering processes can often aid in determining the severity of the rework needed for any soldering job. A snapshot of the article portrait can be found below,



**Figure 15:** Acceptable vs. Rework Needed Soldering Process.

In the standards, there have been many mentions of process requirements that aid in space addendum applications. One of them is the prevention of corrosion. User-approved red plague plan is kept in consideration for silver-coated copper conductors. This aids in lessening the growth of cupric oxide and latent damage due to various environmental conditions. The main constituents of the soldering process are solder paste, cleaning media, soldering system, flux, etc. When these or any of these are changed, proper tests should be run for validation, and the changes should be documented in a changelog. Sn60Pb40, Sn62Pb36Ag2, Sn63Pb37, or Sn96.3Ag3.7 are different types of soldering and are considered ideal per standard. Making sure to choose the correct alloy with good service life, performance and reliability are important. Then the Flux considerations also emphasized these sets of standards and requirements. RO (rosin) and RE (resin) are two of the J-STD-001ES flux categories. These also have activity stats of L0 and L1. The compatibility of these different Fluxes and their activation level must be compatible with the process and thus need to be tested. This goes hand in hand with Solder Paste Testing to check the paste spreads. At the same time, solder balls caused by oxidation should also be checked.

While considering chemical strippers, tests should be run to check for degradation or damage. Chemical strippers are flux removers to clean up excess flux after the soldering process. For thermal protection, heat sinks also should be utilized. Soldering or reworking during the soldering process, the components are at risk of heat and thermal shocks. Heat sinks or thermal shunts give an extra layer of protection for the electrical components. The components also should be considered to have a higher thermal threshold to hold up against thermal shocks and excessive heat during the soldering process. During PTH soldering connections, the solder must fill up the PTH. This allows the top and bottom barrels, better leads, and wetting of lands. The leads could be Flat Leads, Coined Leads, or round Leads. Figure 16 shows a comparison between what is acceptable and not acceptable soldering

Item	Poor Example	Recommended example /Separated by solder resist
Multiple parts mount		
Mount with leaded parts		
Wire soldering after mounting		
Over view		

**Figure 16:** Acceptable vs. Not Acceptable Part Mount

In Figure 16, the different mounting processes and standard mountain procedures can be witnessed. While mounting, there should be enough space between different components for cleaning. An exposed base metal must not prevent a solder connection from forming. Inhibition of formations of the solder joints with OSP must not be permitted. Wire ends and lead ends can not exceed the terminal by more than one lead worth of diameter. Leads should be formed following the system design even before assembly because reworking leads is not recommended. Just bending angles and minor lead adjustments can be acceptable at best. Lead seals, welds, or connections inside components, must not be interfered with or damaged by lead forming procedures.

For cleaning, there are also mentions of ultrasonic cleaning, and emphasis has been put on particulate assembly mattes being cleaned. Transmitting high-frequency sound waves through the liquid enables the Ultrasonic cleaning process. In the scenario where nothing but terminals or connectors are present on the bare boards - this is used. Caution must be abided, as ultrasonic cleaning should only be used if the manufacturer's documentation specifies that the cleaning would not affect or interfere with the electrical performance of the system assembly. Debris, solder splash, wire clips, and solder balls should be cleaned from the particulate matter assemblies.

The Smart Parking System incorporates a PCB design to support the LED and Control Unit Communications. To customize the connection on the PCB board and to enable ethernet communication, soldering was done between different components. The universal standard of soldering, J-STD-001, guided the soldering process of smart parking systems. It offers recommendations to follow the optimum environment built with the correct materials and placements.

## **7.0 Design Constraints**

The conditions that limit the choices that can be made during the design process of a system are called constraints. Constraints force developers to consider edge cases of failure and ensure a safe and reliable execution of the desired system. Constraints include the economic aspects of the project, environmental effects, safety standards, ethical considerations, and time limitations, just to name a few. While developing the design of our smart parking system, there were a variety of constraints that we needed to consider and adhere to. This section will discuss those constraints and address how they will impact the design of our system.

### **7.1 Economic Constraints**

One of the biggest constraints we must adhere to during the design process of our system is the total cost of all the components. Since our project was not sponsored by an external company and we are instead splitting the total cost of our system evenly between all group members, we are limited in some of the design decisions we can make. To counteract this, we cut costs by using components that we already had on our hands, such as ethernet cables. Additionally, we did extensive research to ensure that the components we used were of high quality but affordable. Lastly, before we purchased any components, we talked to all the suppliers to check if there was any student discount available.

It is also important to mention that this constraint explains why we did not design a parking management system for an entire garage. Every garage on UCF's campus has over 1000 spots that could be monitored, and we do not have the finances to design a system at that scale. Instead, we will be designing a system that is just a proof of concept and can be scaled to fit the needs of an entire garage. The final budget for our project can be seen in section 12.

### **7.2 Environmental Constraints**

The parking garages at UCF are exposed to all the elements that Florida's weather presents; we needed to be sure that the electrical components in our project were protected from these elements. For this reason, we chose to use at least an IP65-rated enclosure for all of the electrical components in our project. This ensured that they were protected from any moisture and dust that could be present in the air. Additionally, with the high temperatures that Florida experiences during the summertime, we needed to make sure we were picking components with an appropriate ambient temperature rating.

Another constraint that needed to be considered related to the environment in which our smart parking system was going to function had to do with the garage itself. Each level in the garage except for the top level had a ceiling that could present quite a few issues. The first of these issues was the maximum height our camera could be placed. Ideally, we wanted to place our camera at a high enough level such that the best view, but the ceiling hindered us from doing this. The second issue was the potential lighting issue. Since there were many shaded areas throughout the lower levels of the garage, we feared that it

would have a negative impact on the detection of vehicles. However, after preliminary camera tests, the lighting conditions and the ceiling height were not an issue at all.

### **7.3 Social Constraints**

One of the major constraints that could have affected the design of our system was social constraints. The fact that people often walk through the parking garage or use their car to take up multiple spots instead of just one, could create some issues with the reliability and accuracy of our system. The students and faculty at UCF will likely not have the smart parking system in mind when they are navigating the garage, and we needed to design our system in a way that responds accordingly. For example, we needed to place the LED sign in a location where the view of it had no chance of being obstructed by the way someone decided to park their car. Additionally, we needed to design the computer vision system in a way that it would not mistakenly count people walking by as cars. We also needed to think of the ideal camera placement to deter from being tampered with.

### **7.4 Political Constraints**

This constraint did not have an impact on the design of our system since we were not designing our system for government use.

### **7.5 Ethical Constraints**

With the goal of designing and building a successful smart parking system, ethical constraints are something that we had to keep in mind. IEEE defines its code of ethics in [19] and emphasizes the importance of their technologies having a positive impact on the quality of life, accepting a personal obligation to their profession, their members, and the communities they serve, and committing themselves to the highest ethical and professional conduct. Therefore, this code of ethics served as a good set of morals for senior designs one and two. After reading IEEE's code of ethics, as a team, we understood that we must do the following:

- Uphold the highest standards of integrity, responsible behavior, and ethical conduct in our activities.
- Treat all group members fairly and with respect, not engage in harassment or discrimination, and avoid injuring others.
- Strive to ensure this code is upheld by all group members.

### **7.6 Health and Safety Constraints**

For our project to be considered successful, it had to operate without compromising the health or safety of students and faculty navigating garages. We used tripods to lift our camera to its required height during our testing and presentation, however, in a real implementation of our system, the cameras and LED signs would be mounted on a ceiling or wall. The vibrations that go through the garage while people are driving through them could loosen the screws used to mount the cameras and LED signs. This creates a risk of one of these components possibly falling onto a car or person, which

could potentially cause damage or injury. For this reason, if this system were to be implemented on a larger scale, we suggest that the mounts for all of the cameras and LED signs be checked routinely.

With the fact that we used a camera embedded with computer vision capability and data would be transferred through a LAN within the garage, privacy is another constraint relating to safety that must be considered. Using computer vision gives the programmer many capabilities that could potentially be considered an invasion of privacy against people navigating the garage; therefore, our group pledged that we will not use these cameras for any purpose other than keeping track of open parking spots in the garage.

## **7.7 Manufacturability Constraints**

One of the major constraints that could affect the design of our system in a variety of ways is manufacturability constraints. When we are picking components to be used in our system, we always needed to be aware of if that part was in stock or not. We also needed to be wary of if a component to be used in our design was only sold in bulk quantities or if we could purchase just one. Additionally, we needed to check how long it was going to take us to receive any component after purchasing them.

This constraint was major consideration throughout our design process as it could have had a negative impact on other constraints, including our time constraints and economic constraints. If we did not verify that a company is trustworthy before deciding to purchase a component from them, we could have run the risk of not receiving that component by the promised time or not receiving it at all. This could have caused both a waste of time and money which could have been detrimental to the final design of our project. To counteract this, we started ordering components towards the end of senior design one and over the summer such that they were already on hand at the start of senior design two. After components were ordered, we created a table listing each of them with the expected delivery times to track them easily.

## **7.8 Sustainability Constraints**

We wanted to design a reliable system able to function for 5+ years before needing any maintenance requirements; therefore, we needed to design our system in a way such that this was possible. To do this, we needed to consider the environment that our components were going to be exposed to and be aware of the impacts it could have on our system.

One of the ways we accomplished a reliable system was to work with PoE connections throughout our design rather than wireless connections. This rode our system of batteries which would have needed to be replaced on a routine basis. In a clean environment with no exposure to elements, ethernet cables should last for at least 20 years. Since our system was going to be functioning in an outdoor environment, we expected they should last for about 5-10 years before needing to be replaced.

In addition to working with PoE connections, we also needed to pay attention to the operating temperature of the components we chose. Florida can experience some very hot



weather, upwards of 90 degrees Fahrenheit, and we needed to be sure that our components were rated to handle an ambient temperature in this range and would not malfunction. We also planned to enclose all electronic components in an enclosure rated for at least IP65, which would protect them from both moisture and dust.

## **7.9 Time Constraints**

One of the major constraints we needed to adhere to while designing our system was the amount of time we had. We only had two semesters to design and build a functioning project, and a final product for our smart parking system needed to be delivered by the end of senior design two. To ensure we were staying on track and not falling behind, we created two tables consisting of strict deadlines that needed to be met, which can be seen in section 13. The first table was for senior design one, and the second was for senior design two.

With only two semesters to complete all research, design, testing, prototyping, debugging, and deliverance of a final product, we were prepared to make any major design changes to ensure we meet these deadlines. For example, the web and mobile applications would have been a great addition to the project, and they would have taken our system to another level; but they were scrapped due to not having enough time to complete them.

## **7.10 Testing and Presentation Constraints**

One of the major constraints that came into play during senior design two was how we tested and presented our system. Since we had decided to use ethernet connections between the components within our system rather than wireless connections, we always needed a power source to be available to us at whatever garage we were working at. To deal with this, we used a battery backup or UPS that provided enough electricity, and flexibility to run the tests and demonstrations. Additionally, to present our project, we thought about potentially building a miniature mock garage to show our proof of concept. This could have eased the process of showing how our system works since we did have to set it up at an exterior garage.

## 8.0 System Design

In previous sections, we researched the different technologies implemented in our project, the standards we needed to follow while developing our design, and the constraints that needed to be considered. Now, here is an overview of how the major components of our smart parking system work. This section is separated into five parts, the computer vision system design, LED display system design, mobile application design, web app design, and local server design. Additionally, the final section in 8.0 addresses the components we used on our PCB.

### 8.1 Computer Vision System Design

This section gives a detailed discussion of the hardware and software design implemented in the computer vision system.

#### 8.1.1 Computer Vision System Overview

The computer vision system takes a live video feed of a parking area, detects cars, determines whether a car is entering or leaving the parking row, and transmits this data over ethernet to the database running on the local server. This system utilizes the OAK-1 PoE to satisfy video capture and computing requirements. This camera employs the Movidius Myriad X VPU, which allows us to use advanced computer vision techniques while limiting the amount of data the camera is required to transmit, as explained in section 5.2.

#### 8.1.2 Software Tools

The table below shows the software tools used to develop the computer vision system. Each of these tools is further described in sections 5.1 and 5.2.

**Table 19:** Software Development Tools

Software	Description
<b>DepthAI SDK</b>	A python package containing convenience classes and functions that helped complete the most common tasks using the DepthAI API.
<b>DepthAI API</b>	The API we used to connect to, configure, and communicate with the OAK-1 PoE camera. Supports both Python and C++ APIs.
<b>Python</b>	A high-level, general-purpose programming language. This is the language we used to complete computer vision tasks.
<b>OpenCV</b>	An open-source computer vision library containing a variety of programming functions aimed at real-time computer vision. This is the underline computer vision technology used by DepathAI.

### 8.1.3 Software Design

The computer vision system used a combination of the software tools described above to satisfy the requirements described in section 8.1.1. Upon bootup, the camera immediately starts doing the tasks possible with the PipelineManager class from the DepthAI SDK. The Pipeline Manager allowed us to create a pipeline that defined the workflow processes of the computer vision system before loading it to the camera.

To define these processes, we used DepthAI API installed on the server to populate the pipeline with the nodes responsible for accomplishing each task. The pipeline for the computer vision system consists of four main processes:

- Obtain a live video feed of the parking spots.
- Apply image manipulation techniques (edge detector, hough transform, and regions of interest).
- Apply vehicle detection techniques (Cascade Classifiers, YOLO, or Tiny-YOLO).
- Send this data to the local server and LED display system via ethernet.

The DepthAI API offers predefined nodes we used to accomplish some of these processes. However, we also needed to use the script node. In the script node, we wrote customized Python code that allowed us to determine whether a car was entering or leaving the parking area monitored by the camera.

For our system to work properly, we must begin with a set of empty parking spaces for our system to work properly. The camera will have an aerial view of the parking spaces so that as many white lines that define a parking space are shown as possible. Once the OAK-1 PoE is plugged into the PoE switch and the camera is booted up, it will take a picture of the parking spots to be observed and use the image manipulation techniques described in section 5.1.2 to define the regions of interest, i.e., each parking spot. These regions of interest are then used as a mask over the live video, and a rectangle is drawn for each parking spot. From here, we will use the chosen object detection technique described in section 5.1.2.4 to see if any cars are occupying the regions of interest. If a car enters a region of interest, that region of interest will go from an unoccupied state to an occupied state and vice versa.

While the computer vision system will immediately react to a car entering or exiting a region of interest, it will only transmit this data via ethernet to the local server at a certain time interval, depending on the time of day and how busy the parking garage is. This can be accomplished via the XLinkOut node on the DepthAI API, which allows data to be sent from the OAK device to the host. In addition to the updates, the computer vision system can also transmit a live video feed with an overlay of the computer vision techniques that have been applied.

We would like the final design for our computer vision system to require no human input and work autonomously upon bootup. Applying such advanced computer vision techniques will require extensive testing as there is much room for error. The methods we

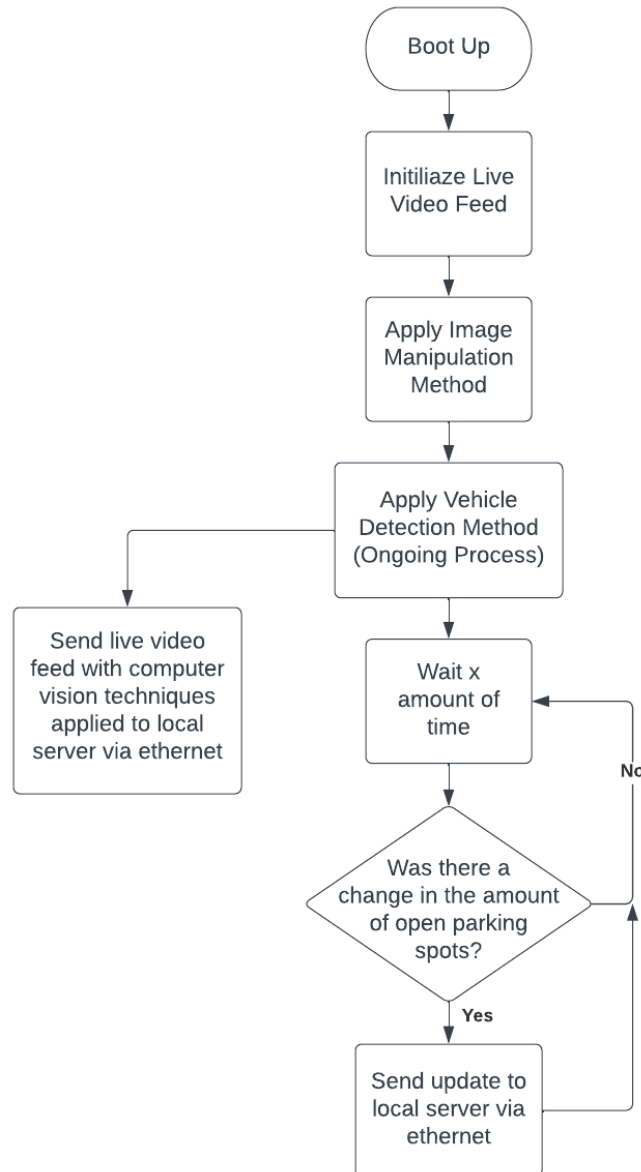
will follow for testing, as well as the actions we will take relating to the design of the computer vision system, depending on our results, are further described in section 10.

For the final implementation, the sample python program, gen2-people-tracker, from the DepthAI experiments repository [36] was utilized as a starting point for the camera computer vision code. As the name implies, gen2-people-tracker was written to track walking people using a people-tracking model. However, for vehicle detection, a vehicle detection model from OpenVino called vehicle-detection-0202 [37] was imported to replace the people-tracking model. The vehicle model conveniently returns the car ID, label, confidence percentage, the x and y coordinates of the upper left corner, and the bottom right corner of each vehicle detected. Determining the direction of a vehicle was accomplished by using the coordinates returned by the vehicle tracking model. A midpoint (i.e., an x and y pair) is computed using the returned coordinates when a vehicle is detected for the first time and when it is last seen before it leaves the camera's field of view. Thus, providing two pairs, (x1, y1) and (x2, y2), where 1 indicates the initial location and 2 is the final location of the vehicle. The origin is the upper left corner of the captured frame. Then, the pair is subtracted from the other; if the difference is a negative number, it means the car was moving right (entering the premises), and if the value is positive, it depicts the car was moving left (leaving the premises). This computation is done with every car entering and leaving the camera's field of view.

A remote database was utilized to send the necessary information to the control unit. Once the vehicle is detected either entering or leaving, the program updates a variable to keep track of the entering or leaving state of the vehicle. If the camera detects a vehicle entering the parking space, it updates the value with a '-1', depicting one less parking spot available. The leaving state of the detected vehicle updates the variable with a '+1', essentially indicating one additional parking spot available within the premises. In the end, the python code sends the value of the mentioned variable to the camera\_log table in the parking system database.

#### **8.1.4 Software Flowchart**

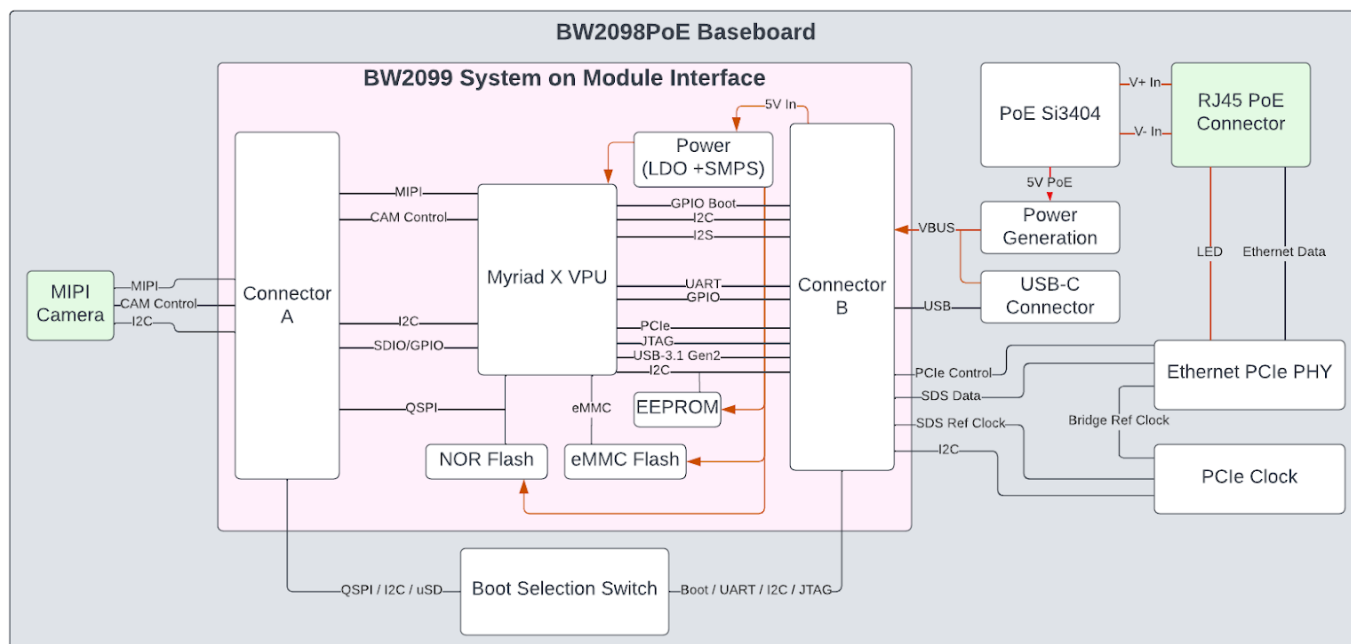
Figure 17 presents a flowchart diagram showing the workflow processes of the computer vision software. It can be seen that the primary responsibilities of the computer vision system are to detect how many parking spaces are available, transmit this information to the local server, and send a live video feed with an overlay of the computer vision techniques.



**Figure 17:** Flowchart of the Computer Vision System Workflow

### 8.1.5 Hardware Design

Referring to Figure 18 below, a basic block diagram of the hardware within the OAK-1 PoE is shown. This figure is only meant to provide a general idea of the primary connections between the components on the circuit board within the camera. It does not show every single connection between the components.



**Figure 18:** Basic Block Diagram of the OAK-1 PoE Hardware

This camera employs two circuit boards known as the BW2098PoE, which is the primary baseboard, and the BW2099 SoM, also known as the OAK-SoM-Pro. The baseboard includes the RJ45 PoE Connector, which receives an external 1000BASE-T ethernet cable that handles data and power transmission. This connector is wired to the PoE Si3404, which converts the high-voltage ethernet connection into a regulated, low-voltage output supply suitable for the rest of the circuit board. These boards are interfaced through two 100-pin DF40C-100DP-0.4V(51) connectors which carry all input and output signals and the 5V input. The power system onboard the BW2099 employs a switched-mode power supply (SMPS) which regulates the 5V input to provide all the necessary digital and analog power for the electronics on the OAK-SoM-Pro.

The primary electronics on the OAK-SoM-Pro include the Movidius Myriad X VPU, a 16GB eMMC 5.1 flash device, a 128MB QSPI NOR flash, and a 32kB EEPROM. USB 3.1 Gen2, QSPI, UART, I2C, 1-lane PCIe, and SDIO are included on the BW2098POE baseboard and are routed to the OAK-SoM-Pro through the connectors. The OAK-SoM-Pro exposes two 2-lane MIPI CSI-2 D-PHY channels and two 4-lane MIPI CSI-2 D-PHY channels, allowing for multiple camera inputs. This system also employs an I2S interface which gives us the ability to connect microphones and an external audio device to the camera; however, this is a functionality we will probably not use for our application.

The baseboard utilizes a boot selection switch which allows the Movidius Myriad X VPU to be booted in a number of ways, including USB-C, EEPROM, NOR flash, eMMC, SPI, and ethernet. This is part of what makes this camera so effective since there is a lot of flexibility in the ways it can be booted.

## 8.2 LED Display System Design

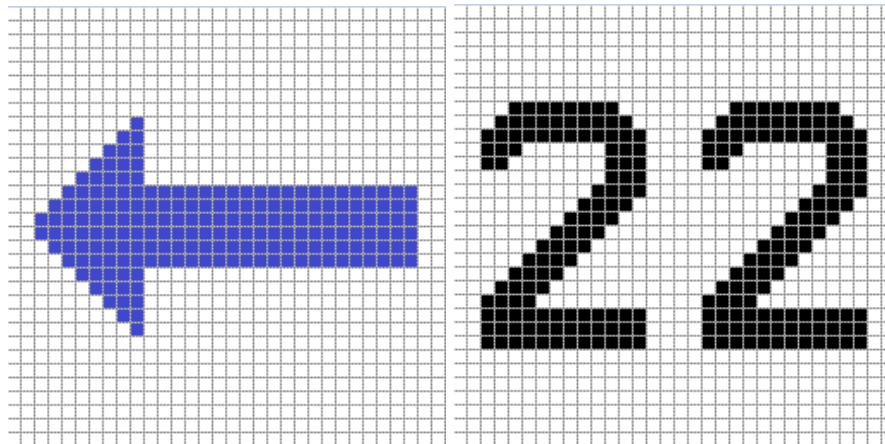
Major items to cover when describing the LED display system are the images to be displayed, as well as the software methods on the microcontroller to display the necessary images. This solution includes having two displays daisy-chained to each other back to back with the displays facing outward from each other. This allows for the same messages to be viewed from either side but also results in the MCU displaying an image as though it is on a 32 x 128 display rather than a single 32 x 64 display.

### 8.2.1 Display Images

The display had to indicate how many open spots were down an adjacent “corridor” and in some cases, it would display data on two different corridors if the display was situated so that it was both adjacent to one corridor and adjacent to an end corridor of the garage. If there was only one corridor being displayed for, then all that needed to be displayed was an arrow pointing in the direction of indication as well as a total free space count. The open spot value should be updated every time the microcontroller receives new data.

For simplification, images can be shown in 32 x 32 blocks. This would result in one display showing two 32 x 32 images. With the two daisy-chained panels, there would be four separate 32 x 32 image blocks concurrently displaying. Most often, the second panel will be mirroring what is displayed on the other side, except for the numbers being properly flipped. Figure 19 shows an example of a 32 x 32-pixel image that gives a rough depiction of what the display will look like.

(a)

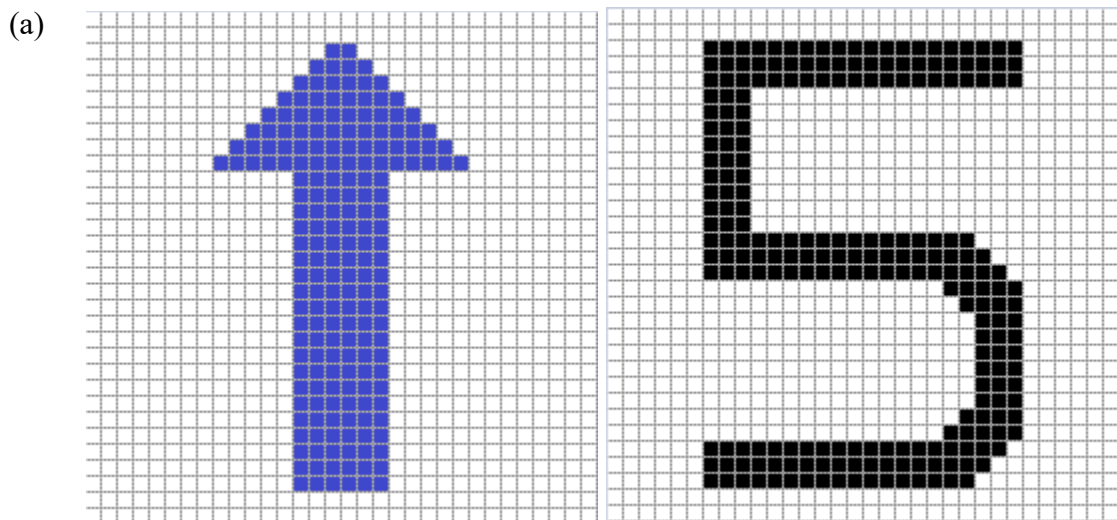




**Figure 19:** LED Example Images (Double Digits) (a) Initial Design (b) Final Design

In reality, the LEDs would have a small bit of margin between them, unlike these pixel images which have zero margins. However, the actual images would have the same resolution, which is represented through the gridlines. In the final implementation, the colors picked would have a high degree of contrast and visibility for the drivers. If the previously mentioned case of the display being situated around a corner corridor was in effect, then the display would be periodically switching from one sign to another. A time of 5 seconds between display images would be used.

For example, if the adjacent corridor had 22 open spots (as shown in Figure 19 above), and the next corner corridor had 5 open spots, then 5 would be displayed, as shown in Figure 20 below.







**Figure 20:** LED Example Images (Single Digit) (a) Initial Design (b) Final Design

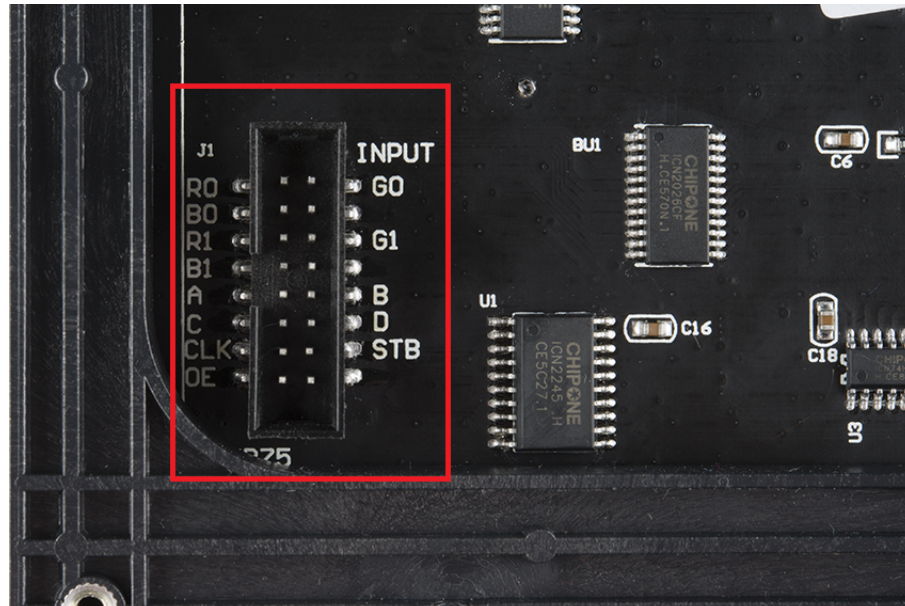
When a driver saw the sign, they would first see that there were 22 open spots to their left. After a few seconds, the sign would change to reflect that there were 5 open spots at the next corner corridor if they continued ahead. Since the signs were double-sided, these corner cases would have to be treated uniquely since the side of the sign facing the corner corridor could not show data on that corner corridor since cars coming from that direction had already been there. Instead, only the adjacent corridor data would be displayed. This resulted in one side of the display cycling between images while the other side constantly displayed one single image.

For the final implementation, a system of dividing each LED display into 4 sections (each 16 pixels wide) allowed us to display up to 3 digit numbers with an arrow on the rightmost side. Examples of this are shown in the final designs (b) next to the initial concept designs.

### 8.2.2 Hardware

To alleviate requirements from the PCB, the LED display was powered by a separate power supply. The LED matrix panel required a 5V power supply at  $\sim 2\text{A}$ . This resulted in two separate power supplies needed for one cluster of panels.

The panel had many shift registers already integrated into the design, greatly reducing the pin count required from the microcontroller. Therefore, the display used a single 16-pin IDC connector, as shown in Figure 21, for the MCU to interface with it.



**Figure 21:** LED Display IDC Connection

Each of the R, G, and B pins correspond to red, green, and blue values for setting the LED colors. There are six of those pins in total, shown in the pinout figure. Pins A, B, C, and D serve as the demultiplexing inputs for the LEDs. CLK, STB, and OE correspond to the clock, latch, and output enable pins for the LED driver, respectively. All other unlabeled pins go to ground. The pinout will be further described under the PCB design section.

### 8.2.3 Software

Once the PCB was finished, the microcontroller needed to be programmed to both interpret data sent from the local server as well as handle driving the LED display. Microchip provides a Software Development Kit (SDK) for programming LCDs called the MPLAB® Harmony v3, which could have helped in the development process via its Graphics Suite. With minimal porting, the provided functions from the Graphics Suite should have allowed the description of shapes and patterns at a high level, making for a much easier process of displaying images on the LED display rather than painstakingly addressing LEDs individually. Even if problems arose with the SDK, porting existing LCD driver functions over should have still been a straightforward task and not required creating a new driver from scratch. However, this ended up not being the case with the display. The display used a wholly unique mode of communication that made porting code not an option and instead required writing our own driver code.

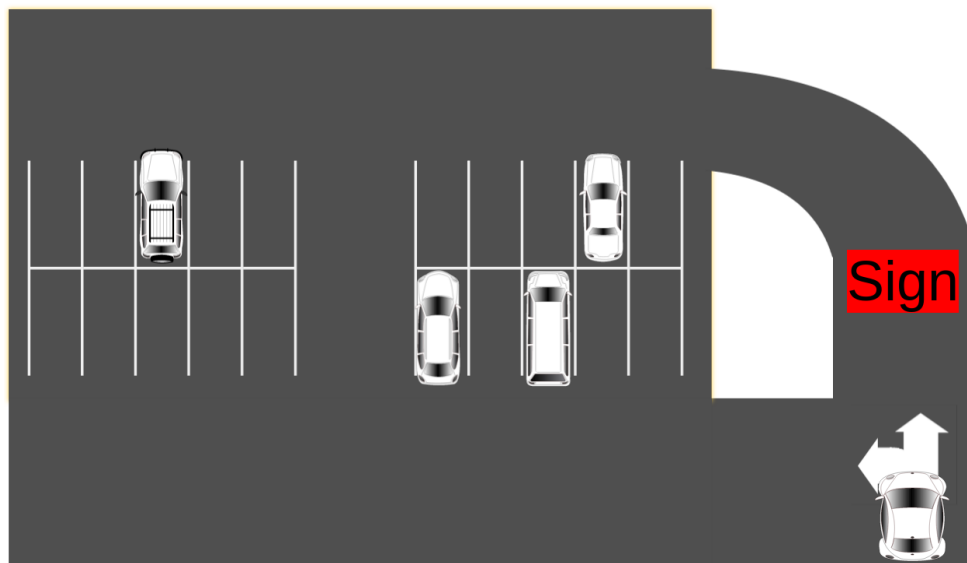
An overall target in the software design was for the code to be easy to work with at a high level. In this case, it would mean having functions that handled tasks like displaying numbers on the LED display just by passing in the value received from the local server, or functions that handle cycling through multiple display images if there was extra data to display, etc. Maintaining this high standard of code readability and usability made for fewer mistakes in the images displayed. Even with a graphics library, the images

displayed are abstracted to fundamental shapes, so functions that handle these shapes to display defined images like numbers make for higher-quality code with fewer errors throughout.

The goal with the LED display driver was for it to be a simple translation layer that had a suite of functions that handled receiving numbers with keywords or other kinds of parameters to potentially indicate the directions the arrows should point in combination with numbers to display. Each of these functions only needed to accept up to two-digit decimal numbers, which allowed for keeping the functions small and efficient. Regular int data types of 8-bit precision more than sufficed, with their value ranging from 0 to 255. A pseudocode function looked like this: `spotsInDirection( section_A_spots, LEFT )`

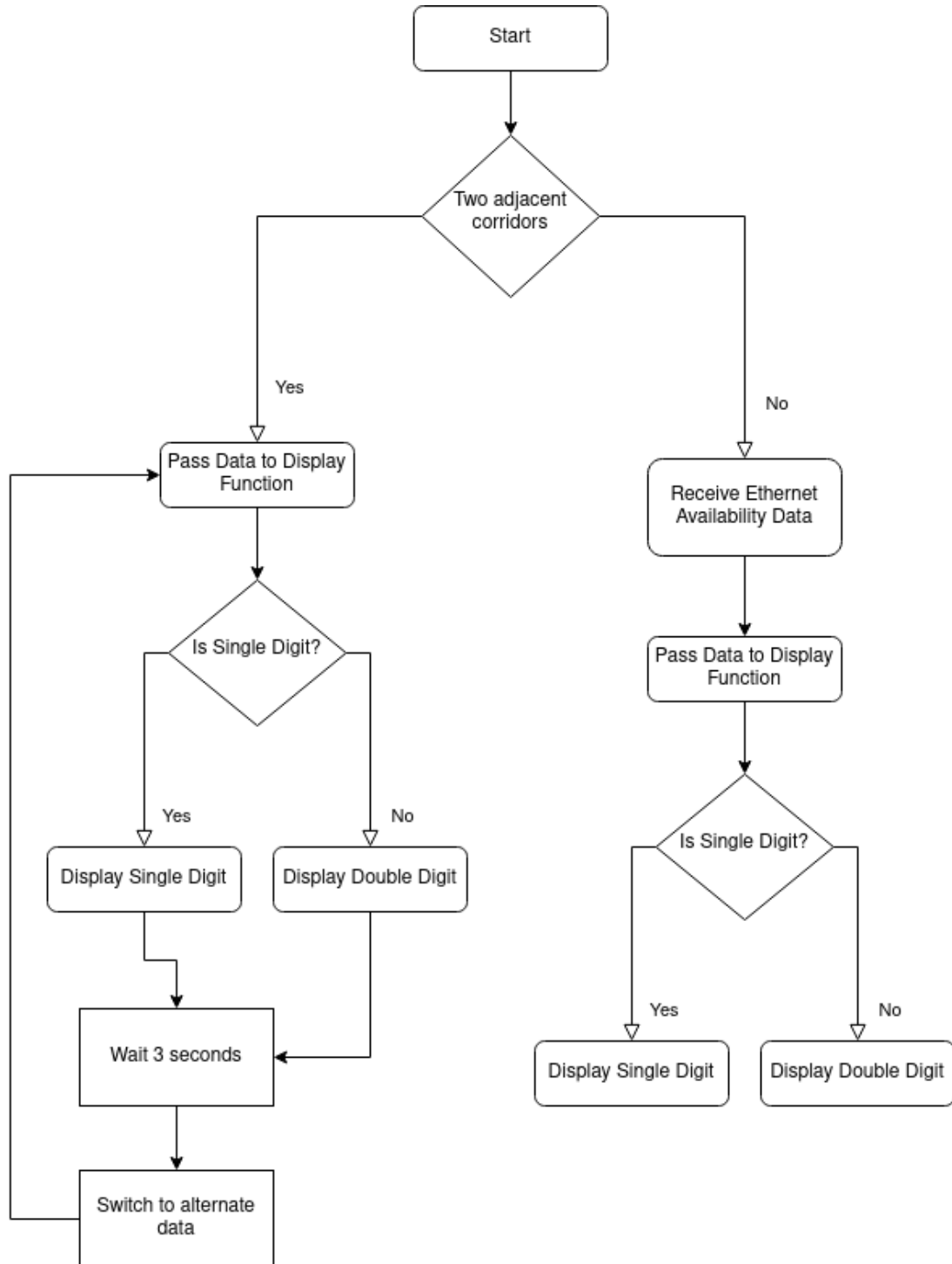
This was a parent function that had subsequent calls to other sub-functions that performed lower and lower level tasks for displaying images on the LED display. This started with a call to a function handling arrow displays, either UP, DOWN, LEFT, or RIGHT. Then there would be a call to a function handling the display of numbers, only accepting inputs ranging from 0 to 99. This function would break down by calling a sub-function for single-digit numbers and a sub-function for double-digit numbers since either of those cases required the numbers to be located in different areas of the 32 x 32 region they were displayed in. The main function of the program would be greatly cleaned up and easy to follow through this design and implementation methodology.

There was another level of complexity to factor in when considering the different configurations that the LED display could be in. As mentioned earlier, sometimes a display could be situated in such a way that two parking alleys could be adjacent to each other when one of them was the end of the garage, and the other was the next alley in. In this case, at the intersection where a driver could enter the last alley of the garage or the second to last, the LED display would show information for both alleys. An example of this is shown in Figure 22.



**Figure 22:** Corner Alley Example,

The general structure of the LED display program is shown in Figure 23. Notice the key decision point of whether the sign is set up to be displayed for two alleys in the case it is at a corner, or if it should display for just one alley, the normal case.



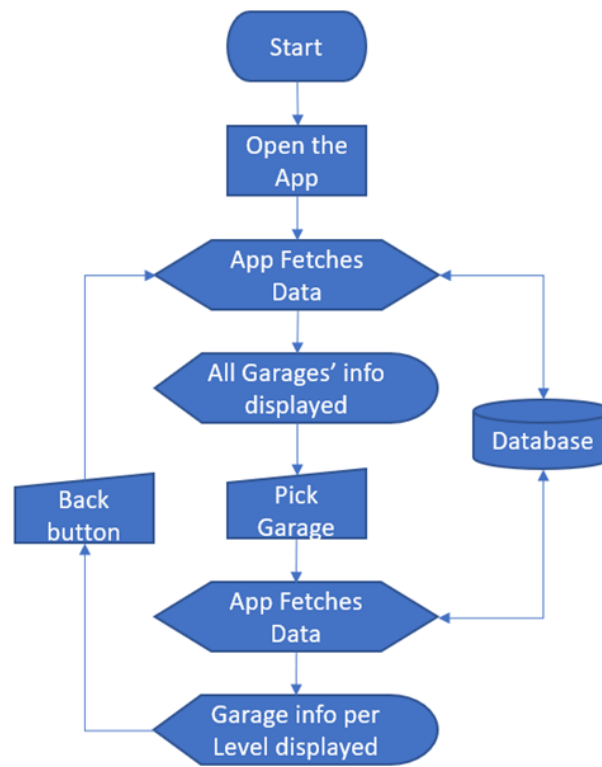
**Figure 23:** LED Display Program Flow

## 8.3 Mobile App Design

Initially, the team researched and planned for a mobile app to provide parking availability to users. The app's primary function was to provide the number of available spaces and the percentage occupancy of all garages. When selecting one of the garages, it was going to provide the number of available spaces per level, information about parking services, mobile app developers, app version, and the last updated time. The app was not going to offer any administration capabilities as they would be offered in the web app version of the parking system. Also, it was designed for the user not to be required to sign in. As soon as the mobile app was launched, it was going to fetch updated data from the cloud database and display it within a few seconds. However, due to not having enough time, the team decided to not implement this software feature to focus on the electrical and computer engineering part of the project. Nonetheless, the designs previously done were left here to show how it was designed.

### 8.3.1 Mobile App Block Diagram

In the block diagram shown in Figure 24, it can be seen that the mobile app design is simple. As soon as the app is launched, the app fetches the data from the cloud database. Then, the app immediately displays a summary of all parking garages on the screen, their available number of spaces, and a percentage of the number of occupied spaces. Users have the option to tap on any of the parking garages on the screen and obtain additional information, such as the number of free spaces per level of that specific garage. The app updates its data continuously every few seconds at peak times and every few minutes during slow times.



**Figure 24:** Mobile App Block Diagram

### 8.3.2 Mobile App User Interface Design

The user interface comprises three different screens. The first screen provides information from all the garages, the second screen offers the availability of spaces per level from the selected parking garage, and the third screen displays information about the app and its developers. The prototype design is shown in Figure 25.



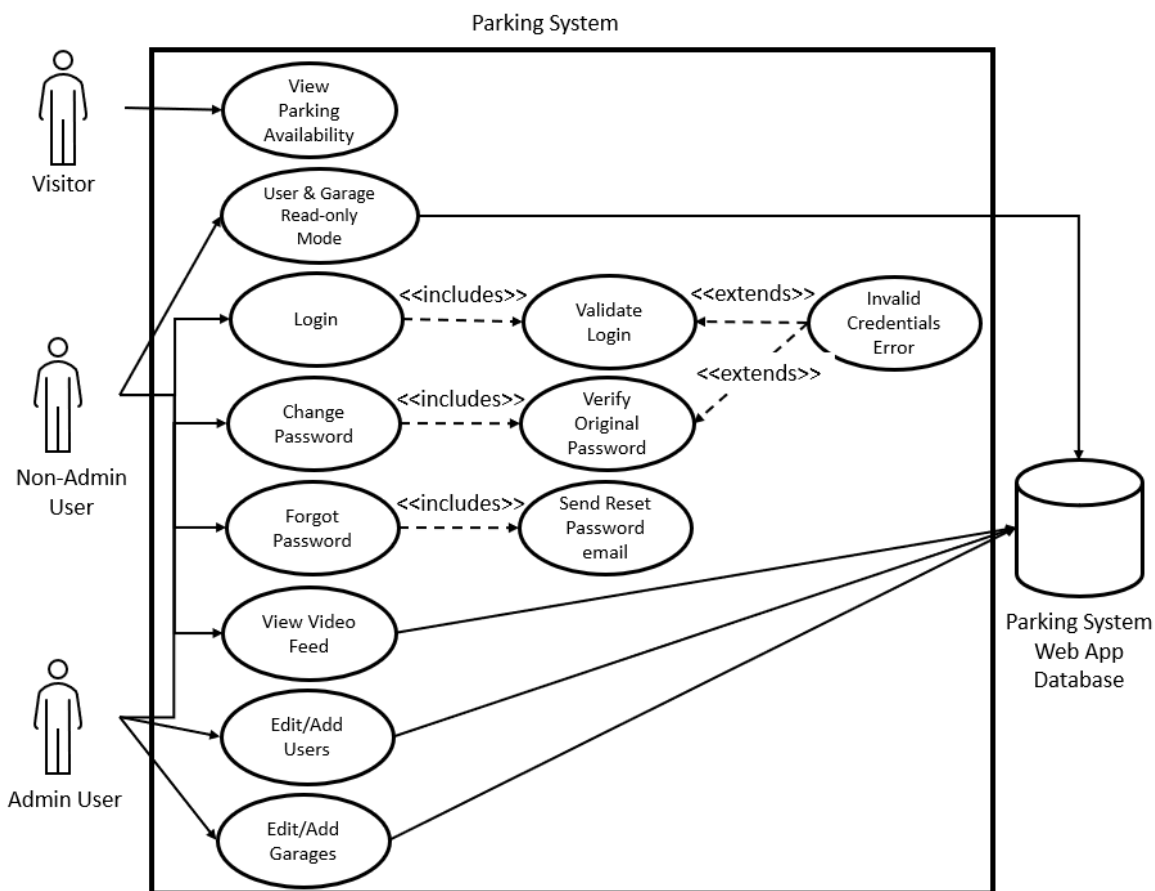
**Figure 25:** Mobile App GUI Prototype  
(Left: main screen, Center: 2nd screen, Right: About screen)

### 8.4 Web App Design

Initially, the team researched and planned to have a web application to offer the same information provided by the mobile app in addition to offering parking administration capabilities. Administrators would have been able to create, read, delete, and edit new users, parking garages, levels, and sections. In addition, the web app's landing page was going to show the parking garage system general information such as the number of available spaces per garage, the ratio of occupancy, and details of the parking garage levels. This information was going to be displayed by default without the need to log in. The web app was going to also offer a login page where admin and non-admin users could access a dashboard where managing the parking system was possible. The dashboard also was going to offer the option to see a video feed of the selected camera in which the computer vision work in progress could be seen. It was planned for non-admin users to have read-only access to the web app, while admin users to have complete access. However, due to not having enough time, the team decided to not implement this software feature to focus on the electrical and computer engineering part of the project. Nonetheless, the designs previously done were left here to show how it was designed.

### 8.4.1 Web App Use Case Diagram

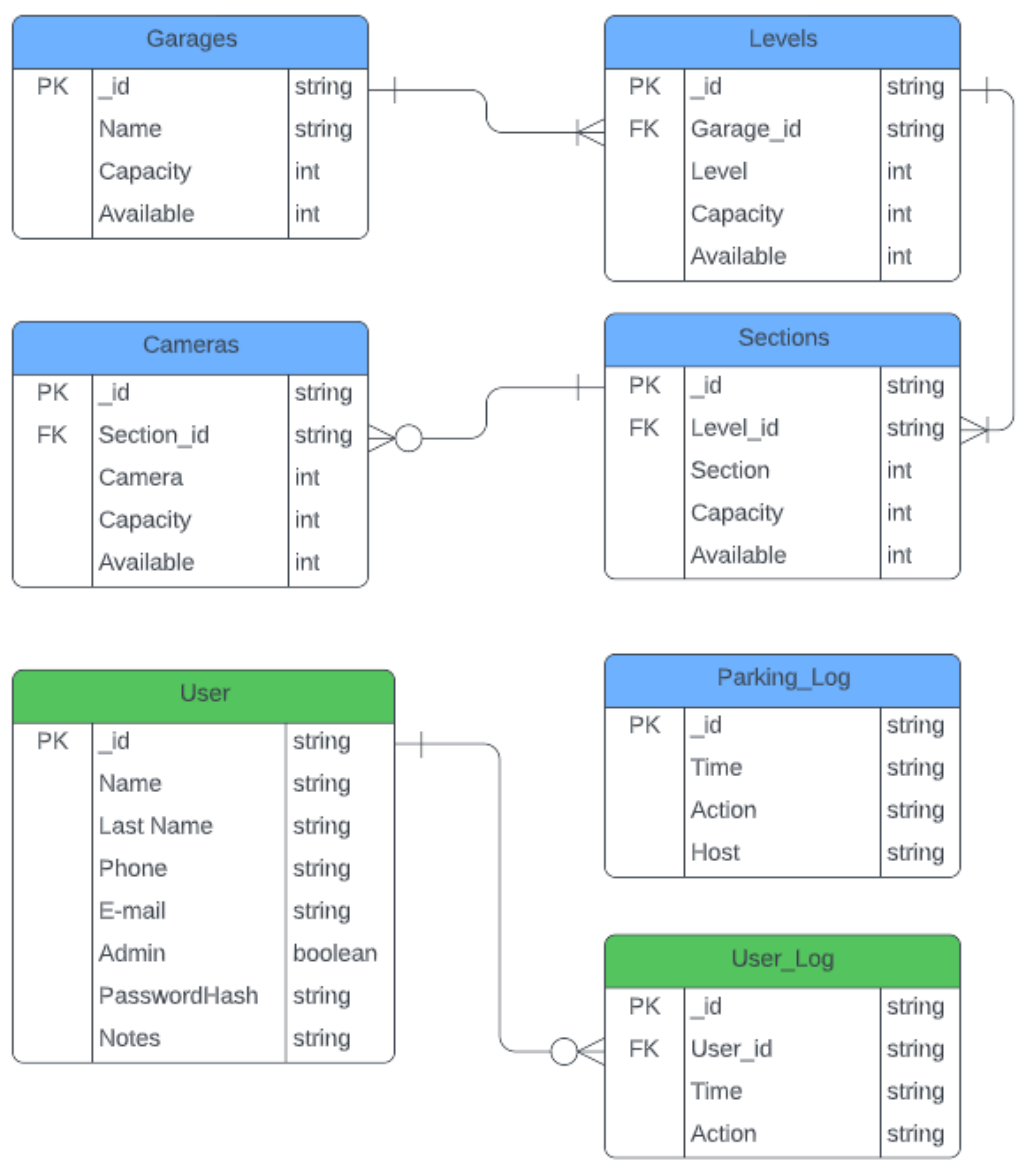
The use case diagram shown in Figure 26 depicts the user's possible interaction with the web app system.



**Figure 26: Web App User Case Diagram**

### 8.4.2 Database Entity Relationship Diagram (ERD)

The web app, the mobile app, and the local server (clients) have access to the MongoDB database (DB). In order to serve the clients, this document-oriented DB contains several collections and documents. There is a parking system collection that contains documents for garages, levels, sections, cameras, and logs. These documents keep data related to the parking system, and the log document is used to record any incoming and outgoing parking space requests for troubleshooting and maintenance purposes. A user collection keeps information related to the admin and non-admin user accounts to administer the parking system. In addition, a user log document keeps and maintains a record of who did what within the web app. A summary of the collection/documents is shown in Figure 27.



**Figure 27:** Database Entity Relationship Diagram (ERD)

### 8.4.3 Web App User Interface Design

The web app was to be designed using ReactJS, a part of the MERN stack as previously discussed. Figures 28 through 31 show some preliminary designs for the essential pages of the app.





Figure 28: Front Page Design

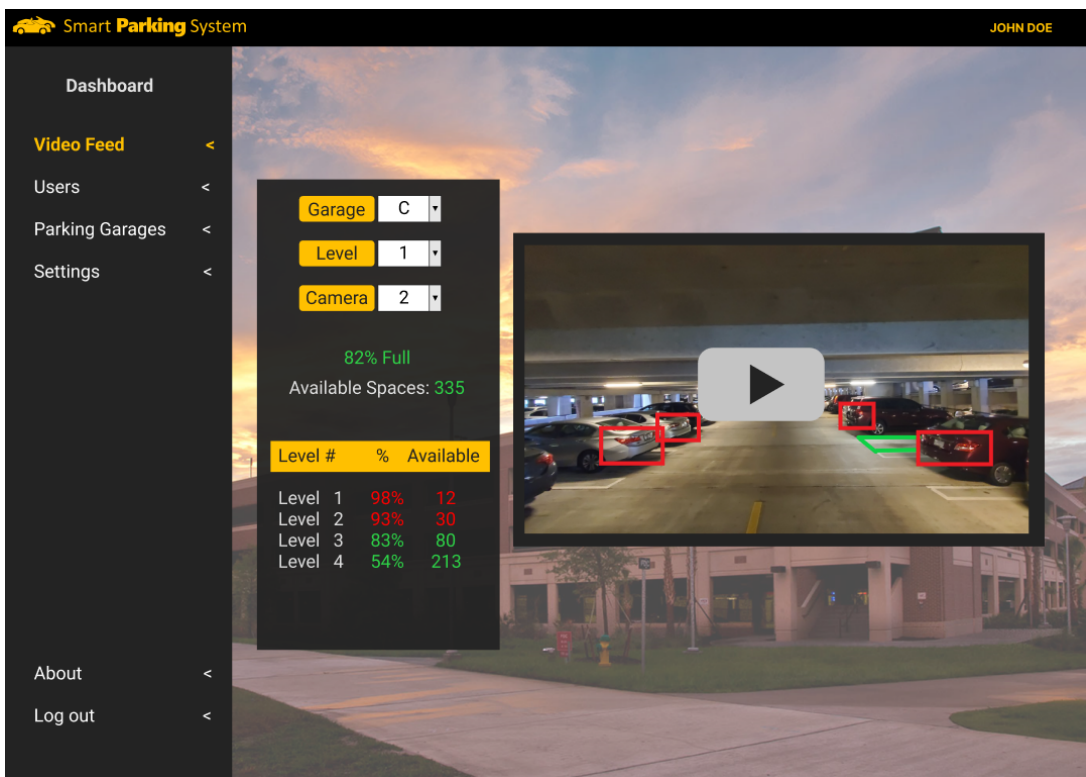


Figure 29: Video Feed Page Design

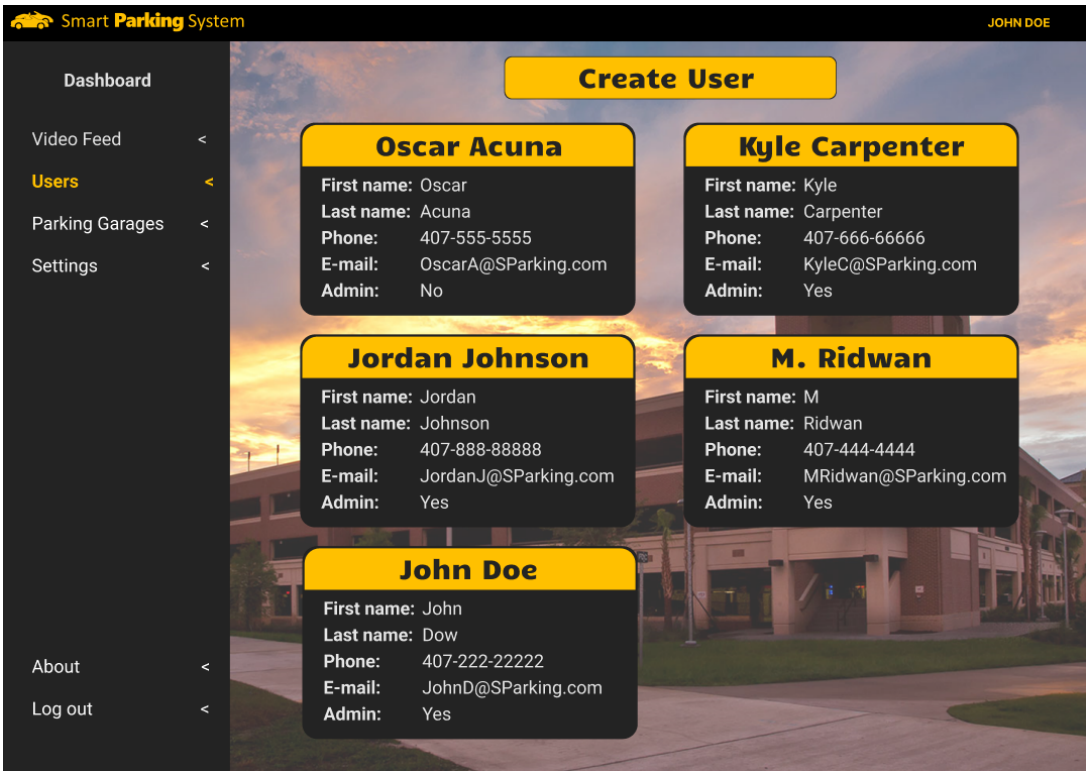


Figure 30: User Administration Page Design

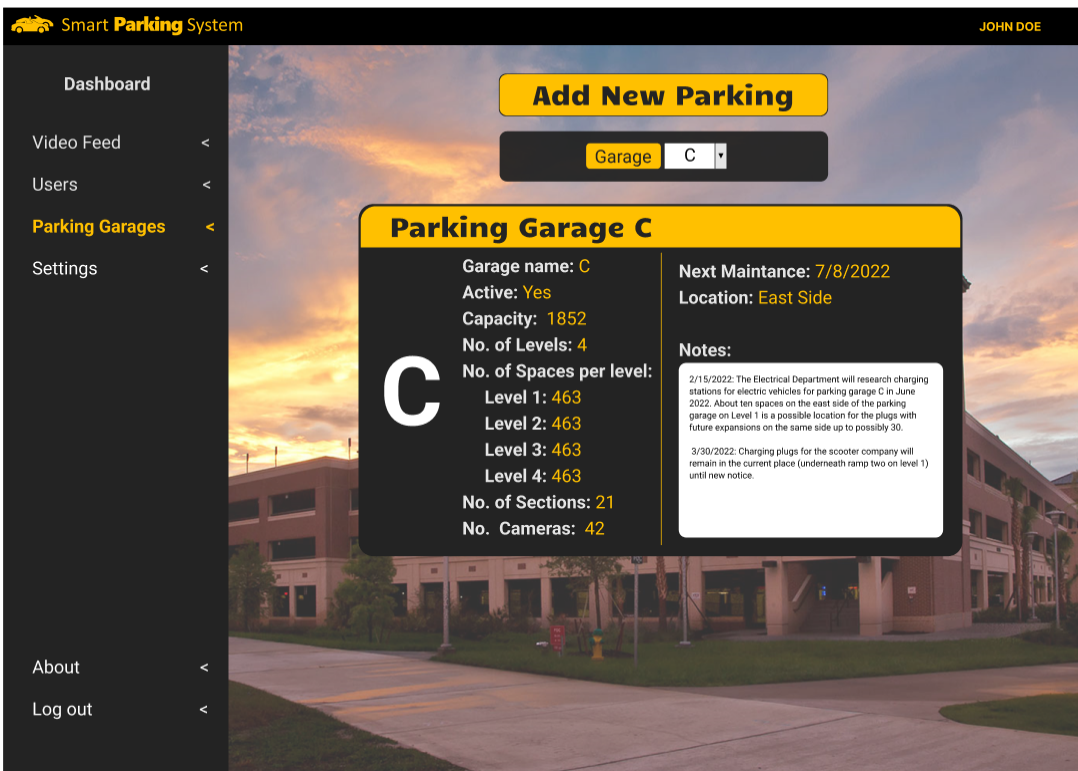


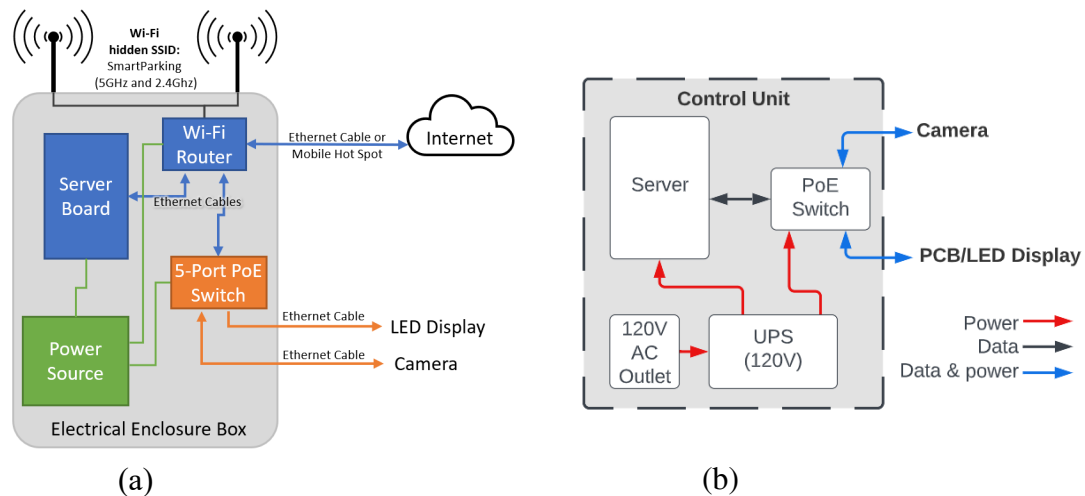
Figure 31: Parking Administration Page Design

## 8.5 Control Unit Design

The control unit serves as the central node between the cameras, LED displays, and the cloud web server, and it contains all the necessary equipment to support the local network, internet connectivity, Wi-Fi signal, and the software running in the server. The control unit is divided into two main components, the hardware (i.e., the equipment) and the software running on the server. Both parts are discussed in the following two sections, including a preliminary design of each major part. In the final implementation of the control unit, the connectivity to the internet, Wi-Fi, and web server were not needed as the web and mobile app was scrapped, and therefore, they were not implemented.

### 8.5.1 Control Unit's Hardware

The control unit comprises several elements, as shown in Figure 32 (a). For the server board, a single-board computer is used as the control server of the parking system. The selected server board comes with the necessary parts: the CPU, Memory RAM, storage on where to install the operating system, and a power supply that runs on 120V/60Hz. Although this server board provides a display port, no display will be needed for the server to function on its daily tasks; the server can be accessed remotely using remote control software such as TeamViewer or Remote Desktop. As long as the server is connected to the garage network, an administrator can access it remotely for updates and maintenance. Nevertheless, a display will be needed for the development stage and for demonstrating the proof of concept since a video feed from the camera with the AI analysis of the parking spaces overlaid onto the video feed is desired.



**Figure 32:** Parking System Control Unit. (a) Initial design, (b) Final design.

A 5-port PoE+ switch provides connectivity between the server board, the Wi-Fi router, cameras, and LED displays, effectively creating a local network. The PoE switch provides the Power over Ethernet (PoE) necessary to power the cameras and the PCB boards. Fortunately, for the proof of concept, this switch does not require any configuration as it is an unmanaged switch; the electricity on PoE ports is not turned on unless a PoE device is connected, in which case, the switch automatically negotiates the

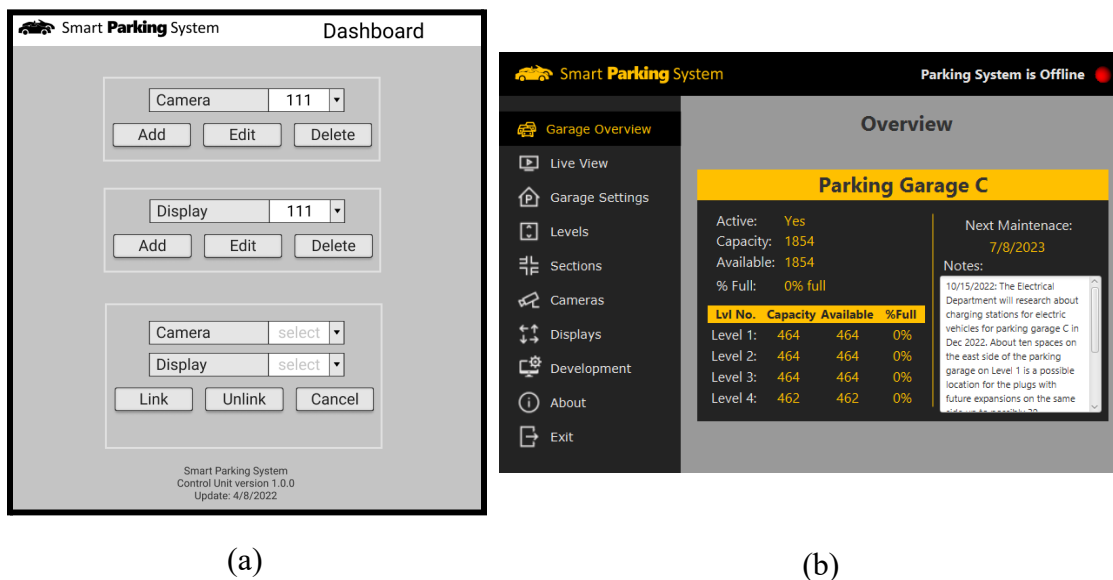
power allocated for the plugged-in device. A bigger switch would be needed for a complete parking system, probably with 24- or 48-ports, with most of them providing PoE. For the Wi-Fi network, a Wi-Fi router would be used. There was a conversation with UCF's IT department about the proper way of providing internet access to the control unit via UCF's network. But, it was not approved for security reasons. However, since internet connectivity was no longer required since the web and mobile app were not going to be implemented, using a Wi-Fi router was no longer needed. In addition, one 3-foot patch ethernet cable is used to connect the server board to the 5-port switch, and two 25-foot ethernet cables can be used to connect to the camera and the LED display to the switch. These Ethernet cables can be either Cat5e or Cat6 as both support PoE and speeds of 1Gbps.

These components would be installed inside a wall-mounted IP-66 electrical enclosure box with 16in x 12in x 6in (height x wide x depth) dimensions. In a real scenario, the site would provide the power source as an electrical outlet where the equipment can be plugged in; however, an battery backup (i.e., an UPS) was used to power the system while doing tests and demonstrations. The final hardware design is shown in Figure 34 (b) above.

### 8.5.2 Control Unit's Software

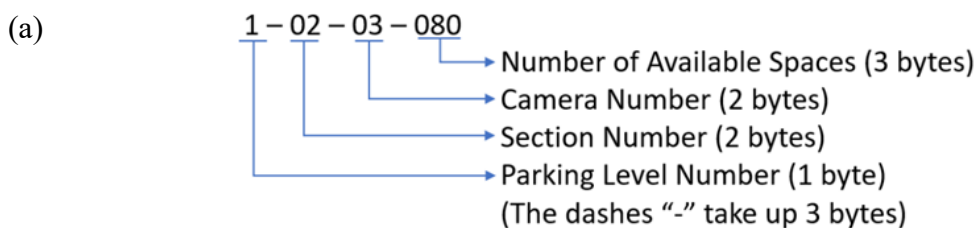
The custom software was designed to run on the server, and it consists of three main parts, the operating system, the local database, and the custom program that runs the parking system. The custom program facilitates the communication between the cameras, the PCBs, and the web server in the cloud. For the operating system, Ubuntu Server version 22.04 LTS was planned to be used; the LTS designation stands for long-term support, which means that throughout the lifetime of this release, there is a commitment to update, patch, and maintain. Without long-term support, the operating system can become a security risk since vulnerabilities develop over time [29]. Besides being open source and free of charge, this Linux distribution is compatible with all the necessary software needed for the development of the parking system, such as OpenCV, Java, Python, DepthAI, MySQL, and most importantly, the server board. However, for the final implementation, the server that was purchased already came with a Windows operating system, which was also compatible with everything described above.

Python is needed since DepthAI, the program that controls the camera is written in Python. In addition, the custom program was written in Java. It contains a graphical user interface where cameras and displays can be added, edited, and deleted. Also, the program provides the means to link cameras to LED displays. This feature is necessary since many LED signs can display different information based on the location they are installed. For example, two cameras can monitor the spaces in a section of the parking garage, and the information provided by them can be displayed in one or two LED signs; thus, these cameras and LED signs are said to be linked. A preliminary design of the graphical user interface (GUI) of this program is shown in Figure 33 (a). The final GUI was redesigned using the color theme and overall look of the web app, as shown in Figure 33 (b).



**Figure 33:** Local Server Java Program GUI Design. (a) Initial Design, (b) Final design.

Initially, the data received from the cameras were going to be stored in a text file called MM-DD-YYYY-1-02-03.log; this file was going to be created by the DepthAI (the camera control software) with a name that included the date of the log, the parking garage level, section, and camera number. The data inside this file was going to be stored in the following format: a timestamp + 12 bytes that will include the level where the camera is located, the section number, the camera number, and the number of available spaces. (e.g., MM-DD-YYYY-hh-mm-ss 1-01-01-08). The meaning of each digit is summarized in Figure 34 (a). However, in the final implementation, the data is written directly to the camera\_log table in a MySQL database. Figure 34 (b) shows data injected by the camera.

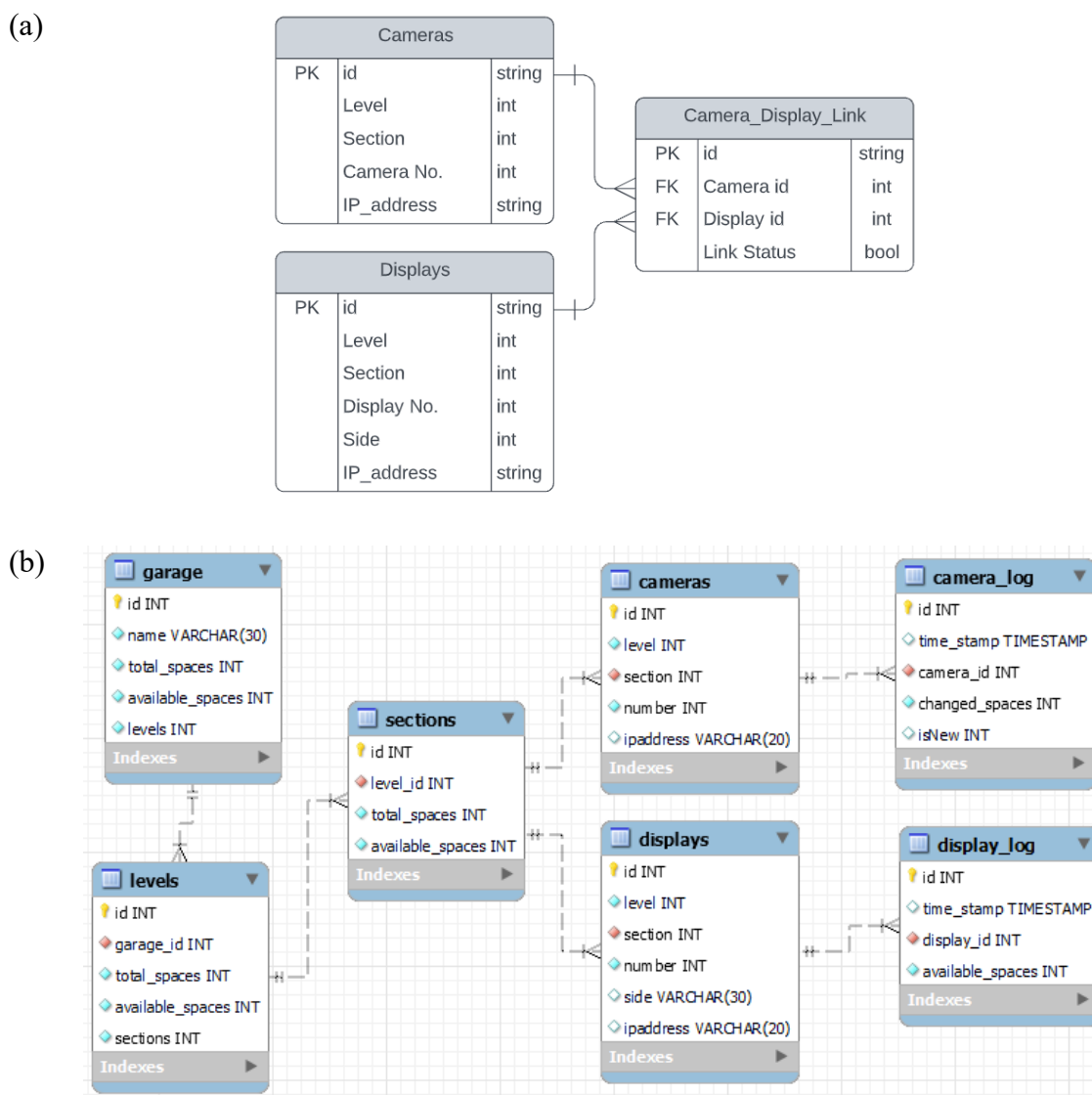


(b)

	id	time_stamp	camera_id	changed_spaces	isNew
▶	1	2022-11-11 21:31:27	1	-1	1
	2	2022-11-11 21:31:27	1	-1	1
	3	2022-11-11 21:31:27	1	-1	1
	4	2022-11-11 21:31:27	1	-1	1
	5	2022-11-11 21:31:27	1	1	1

**Figure 34:** Camera Text File Data Format (a) Initial Designed, (b) Final design.

The software server reads this data from the data; it updates its local variables and sends the number of available parking spaces to the LED display linked to the camera that sent the data, and updates the database section, level, and garage tables with this number as well. This data is sent via ethernet to the LED's microcontroller to be displayed.



**Figure 35:** Local Server MySQL Database Design (a) Initial Design, (b) Final Design.

The database is hosted locally using MySQL; this database is an open-sourced relational database management system (i.e., tables, rows, and columns). The MySQL program is

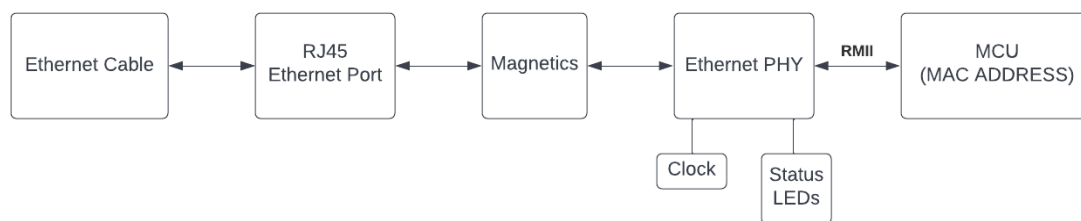
compatible with the chosen operating system, and it works well with Java and Python. The database can be managed using a command line terminal or through software called MySQL Workbench. This software is a graphical user interface version of the command line terminal, which facilitates the management of the database. Fortunately, one of the team members had some experience integrating Java and MySQL, which reduced the learning curve when coding the program. The initial database design was rather simple, as shown in Figure 35 (a); basically, it contained three main tables: a camera table that maintains information on every camera; a display table containing the information on every LED display; a camera/display link table to determine what display to send data from which camera. The camera and display tables also contain the IP address of each device for the server to determine what address to send what data. The camera id and the display id are the primary keys for their respective tables, and they are the foreign keys in the camera/display link table. Figure 35 (b) shows the final database design, which includes tables for the garage, levels, and sections used to keep track of the number of available spaces that changed in real-time.

## 8.6 PCB Components

This section will introduce the components that exist on the PCB board along with the microcontroller. The components that are discussed include an Ethernet PHY Chip, an RJ45 Ethernet Port, Insulation-Displacement Contact (IDC) Connectors, a Step-Down Voltage Converter Circuit, an oscillator, and a push-button. A detailed schematic including all of these components is shown in Section 9.0.

### 8.6.1 Ethernet Components

To interface the ethernet cable with the microcontroller, there were three major components that were needed, as shown in Figure 36 below. The components are the RJ45 Ethernet Port, Magnetics, the Ethernet PHY, and an external clock. Each of these components is further explained in the following sections.



**Figure 36:** Simple Schematic of Interface between Ethernet Cable and MCU

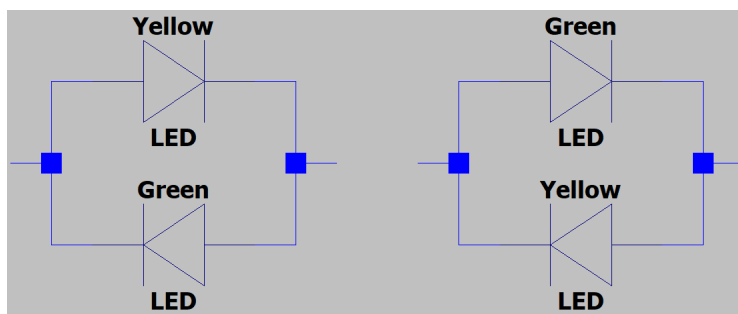
#### 8.6.1.1 RJ45 Ethernet Port

While operating our smart parking system, we made use of a Power over Ethernet Splitter/Extractor close to the PCB to separate the power and data coming from the ethernet cable. The data from this splitter was transferred onto a non-PoE cable and connected to the RJ-45 Ethernet port on the PCB. The ethernet port we used was one of Würth Elktronik's non-PoE RJ45 LAN Through Hole Reflow Ethernet Ports. The connections between the ethernet port and the PHY chip allowed for there to be two

separate channels where data can flow with ease. One channel was for transmitting data, and the other was for receiving data, however, we did not figure out how to get this working in our final design.

This port is integrated with magnetics which simplified the design that needed to be done on the PCB and made for easy connections with the PHY chip. Magnetics were able to be implemented external to the port via an ethernet magnetics module, so using a port where the magnetics were already integrated into it was convenient. The magnetics were a vital part of the connection between the ethernet port and the PHY chip as they mitigate electrostatic discharges, ground loops, and noise in addition to providing galvanic isolation.

This ethernet port had two status LEDs which were connected to the Ethernet PHY to communicate information such as whether there was activity on the PHY chip or not, data speed, and if there is a link between the ethernet cable and the RJ45 port. The LEDs were designed in a way where they are parallel and facing opposite directions such that we could choose between two colors for the LEDs, as shown in Figure 37.



**Figure 37:** Simple Schematic of LEDs on RJ45 Ethernet Port

### 8.6.1.2 Ethernet PHY

The Ethernet PHY is a physical layer transceiver device for sending and receiving Ethernet frames based on the OSI network model. The OSI model describes seven layers that computer systems use to communicate over a network, and the Ethernet PHY was used to help cover two of them, the physical layer and the data link layer, as defined by the IEEE 802.3 standard. The PHY chip we used was Texas Instruments' TLK111. This is a single-port Ethernet PHY used for both 10-BASET and 100-BASET signaling. It fit nicely with our design as it had the same power supply requirements as the MCU and provides all the pins we need for our system to work correctly.

The connection between the PHY chip and the media access controller (MAC) of the MCU is called the media-independent interface (MII). There are many variants to the MII; however, we used the reduced MII (RMII) as this reduced the number of pins required to connect the PHY chip with the MCU. Additionally, since we were using the 64-pin package version of our chosen MCU, it is the only interface that was supported. A table containing the pins that were used for both the ethernet port interface and RMII is shown below.



**Table 20:** Ethernet PHY Port Interface/MCU MAC Interface Pins

Pin	Description
<b>Ethernet Port Pins</b>	
TD-, TD+	<b>Differential Transmit Output:</b> Differential outputs that will be automatically configured to 10Base-T signaling.
RD-, RD+	<b>Differential Receive Input:</b> Differential inputs that will be automatically configured to 10Base-T signaling.
<b>MCU MAC Pins</b>	
MDC	<b>Management Data Clock:</b> Clock signal for the MDIO interface.
MDIO	<b>Management Data I/O:</b> Bidirectional command/data signal synchronized to the MDC. Either the MCU or PHY may drive this signal.
TX0, TX1	Transmit Data Bit 0 and Bit 1
RX0, RX1	Receive Data Bit 0 and Bit 1
TX_EN	<b>Transmit Enable:</b> Indicates the presence of valid data inputs on TX0 & TX1
RX_DV	<b>Receive Data Valid:</b> Indicates that data is present on RX0 and RX1.
CRS_DV	<b>Carrier Sense/ Receive Data Valid:</b> Combines the Carrier Sense and Receive Data Valid indications.

### 8.6.1.3 External Clock Source

The RMII required a 50 MHz external clock source to work effectively and we used the ACHL-50MHz-EK for this purpose. This clock signal was fed into the Ethernet PHY and then sent to the MCU as a reference. We did not ever verify that that this clock signal worked correctly, however, a table containing the pins used for the clock interface is shown below.

**Table 21:** Ethernet PHY Clock Interface Pins

Pin	Description
XI	<b>Crystal/ Oscillator Input</b> <b>RMII Reference Clock:</b> Primary clock reference input. For RMII, this must be connected to a 50MHz $\pm$ 50ppm-tolerance CMOS-level oscillator source.
CLKOUT	<b>Clock Output:</b> With the RMII this pin provides a 50 MHz clock signal. This allowed the MCU to use the reference clock from the TLK111.

### 8.6.2 MCU LED Interface

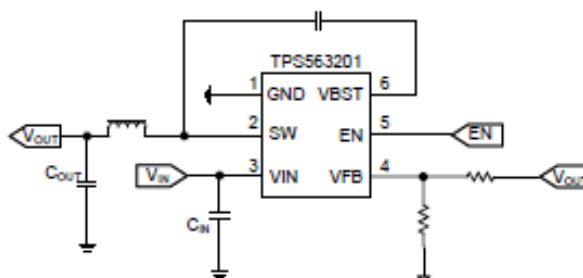
Once the MCU received the data sent over the ethernet connection originating from the central server through the MAC, it would then need to be reflected on the LED sign. Since the directions for pin connections were written for applications with an Arduino Uno or Arduino Mega, we had to look at the datasheets for the chips on both of these MCUs and see how the pins corresponded with the MCU we were using. After doing this, we found that we would need three types of signals for the MCU to communicate with the LED sign. The signals we used and a description for each of them are shown in the table below.

**Table 22:** MCU LED Interface Pins

Signal Name	Description
PA3-PA27	<b>Parallel I/O Controller (Channel A):</b> Managed fully programmable GPIO pins. Six of these pins served as digital pins to communicate RGB color values, four of them were connected to the LED Demux, and three were connected to the LED Drivers' Clock, Latch, and Output Enable.

### 8.6.3 Step-Down Voltage Converter Circuit

When our smart parking system was in operation, our first design was going to use a Power over Ethernet Splitter/Extractor close to the PCB to separate the power and data coming from the ethernet cord. This power was transferred to a 12 V 5.5 x 2.5 mm DC plug and connected to the PCB via a DC jack. From here, Texas Instruments' TPS563201, a synchronous step-down voltage regulator, would've been used to step down the voltage from 12 V to 3.3 V. Stepping the voltage down to this level would allow us to provide power to the Ethernet PHY, MCU, and oscillator without damaging any components on the board. Referring to Figure 38, a simplified schematic for this voltage regulator is shown.



**Figure 38:** Simplified Schematic for the TPS563201 [ $V_{out} = 0.768 \cdot (1 + (R1/R2))$ ]

In order to choose the electronic components we needed to step down the voltage to 3.3 V, the formula above was used.  $R2$  is the vertical resistor connected to ground and  $R1$  is the horizontal resistor connected to  $V_{out}$ . After plugging in 3.3 V for  $V_{out}$  and solving for  $R2/R1$ , we found that  $R2/R1$  was equal to 3.3. Since this is a ratio, we could choose any set of resistors that fit this ratio and decided to work with 33 k $\Omega$  and 10 k $\Omega$  resistors. In

addition to these resistors, the datasheet suggests using an inductor with a value of 2.2  $\mu\text{H}$ , and  $C_{out}$  can be any value between 20 and 68  $\mu\text{F}$ .

In our current design, we moved away from the TPS563201. Once we realized that our design did not pull a lot of current and did not need something that was very efficient, we went with a simple linear voltage regulator in the LD1117V33. We were continuing to work with a 12 V input but then decided to use a 5 V input as this caused less heat dissipation from the regulator.

#### **8.6.4 IDC Connectors**

In our original design, we had the wrong idea about the connectors that would be needed. In Senior Design 1, we thought the PCB would need 3 IDC connectors to communicate with components that are external to the board. Two of these connectors would've been used for the Ethernet PHY and MCU such that the host computer can communicate directly with each of these chips. The third connector would've been used such that the MCU can communicate the LED sign. The IDC Connector that was going to be used to communicate with the LED sign was Samtec's STMM-108-01-G-D. This is a 16-pin connector that we thought would allow all connections to be made between the LED Sign and MCU through a ribbon cable. The pins on this connector are set in 2 rows of 8 and have a 2.0 mm pitch.

The two IDC Connectors we were going to use to communicate with the MCU and Ethernet PHY chip for debugging purposes would've been Samtec's FTSH-105-01-F-D. This is a 10-pin, 1.27 mm pitch, micro connector which would give a host computer the ability to communicate with the two chips on the board. Each of these chips supports JTAG, which is a common way of interfacing with chips. To use these connectors effectively, we would need to use a debugger as this will allow us to change the signals on the 10-pin connectors into a signal that is able to be plugged into a USB port on a host computer.

In our current design, we took a simpler approach and decided to use a single 1x8 female connector for debugging. We made this choice once we realized that Serial Wire Debugging was cheaper than JTAG and that you could program the Ethernet PHY through the MCU. Additionally, we added three 2x1 female connectors, one was used for erasing MCU firmware, and the other two were used as voltage test points. Lastly we changed the LED connector to one with a pitch of 2.54 mm similar to a breadboard.

#### **8.6.5 Push Button**

In our original design, the PCB included one push button, which would've allowed us to easily reset both the MCU and Ethernet PHY Chip if need be. The reset pin is default high and in order to activate it, it needs to be driven low. This would've been accomplished by tying the positive terminal of the button to the reset pin and the negative terminal to ground. When the button was pressed, the reset pin would be driven low and reset the MCU. The Ethernet PHY would've also reset when this happened as we would use the MCU to alter an internal register on the PHY that is latched with its reset function. In our current design we decided to omit the push button.

### 9.0 Prototyping

The subsystem in our design that will use a PCB is the LED sign system. The components described in section 8.6 were used to develop an initial schematic which is shown in Figures 39.1 and 39.2. The large chip that is split in half between the two pictures is the Ethernet PHY chip. As we learned more about our system, our design changed and we moved away from this initial schematic. Our current design which was tested and built onto PCB is shown in Figures 39.3 and 39.4 and the final PCB layout is shown in Figures 39.5 and 39.6

### 9.1 PCB Schematic Capture

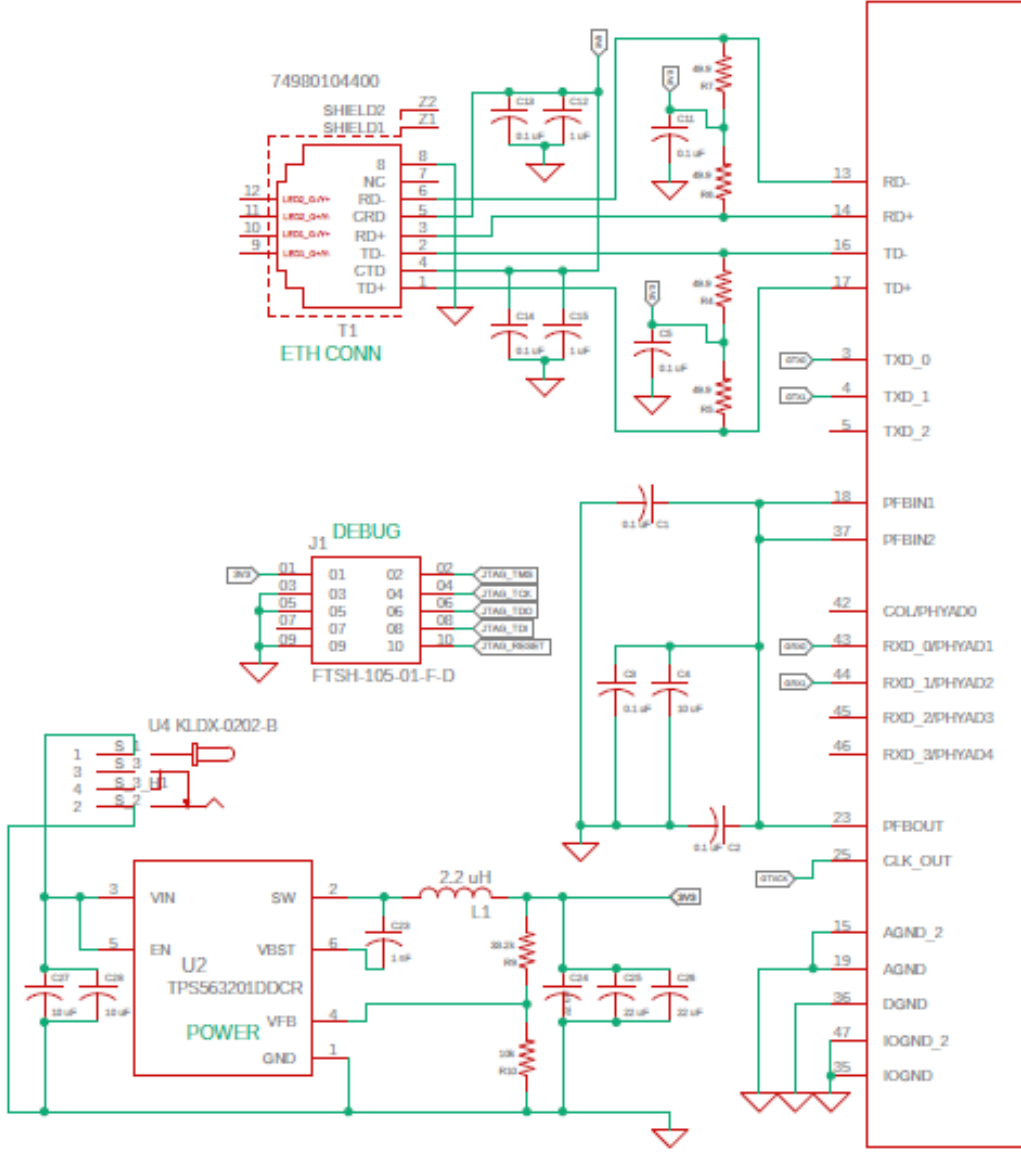


Figure 39.1: PCB Schematics (Left Side)

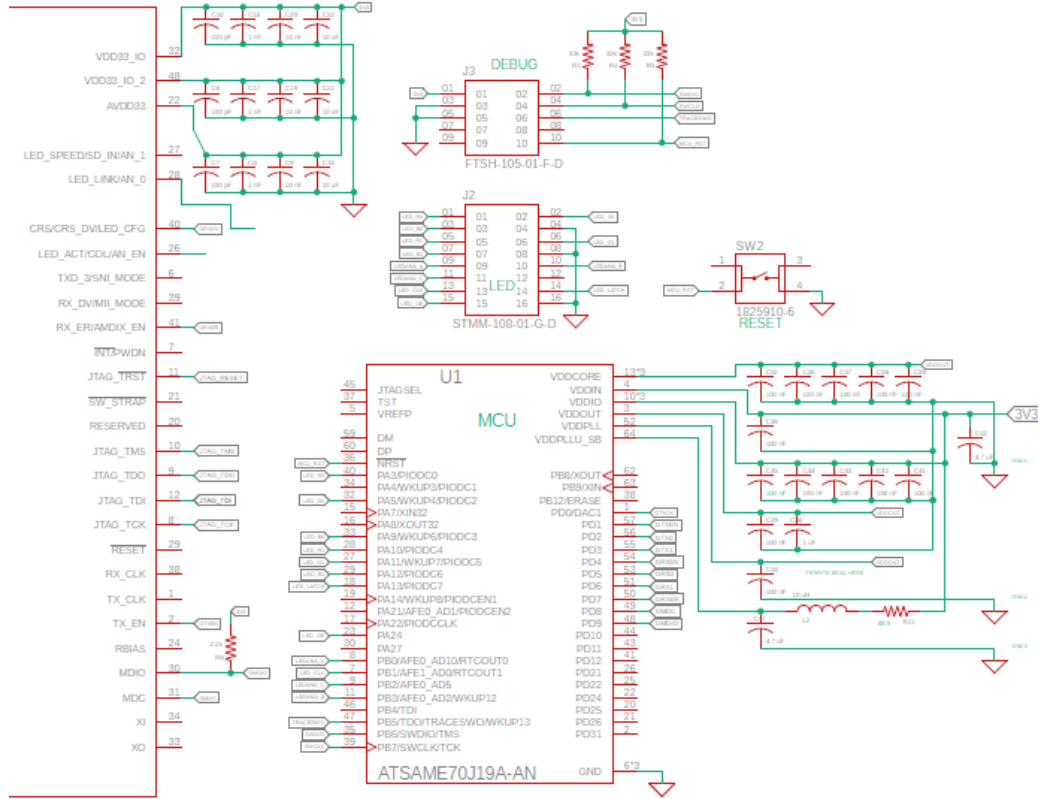


Figure 39.2: PCB Schematics (Right Side)

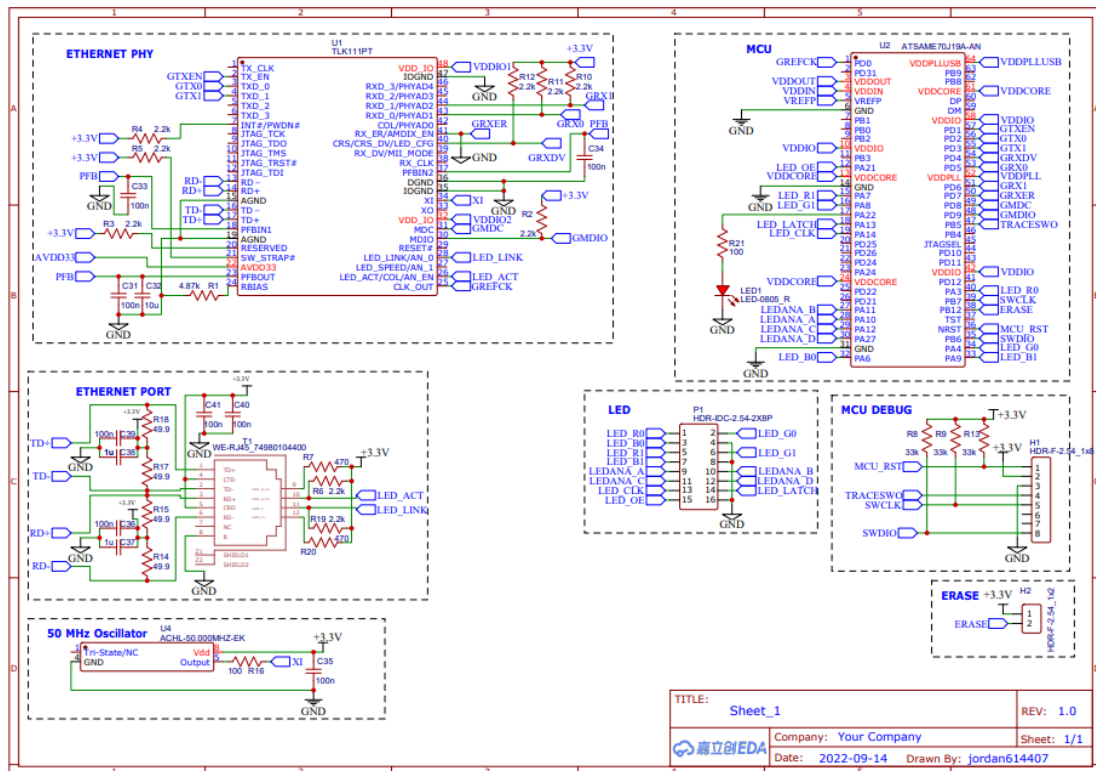


Figure 39.3: Current PCB Schematics (Signals)

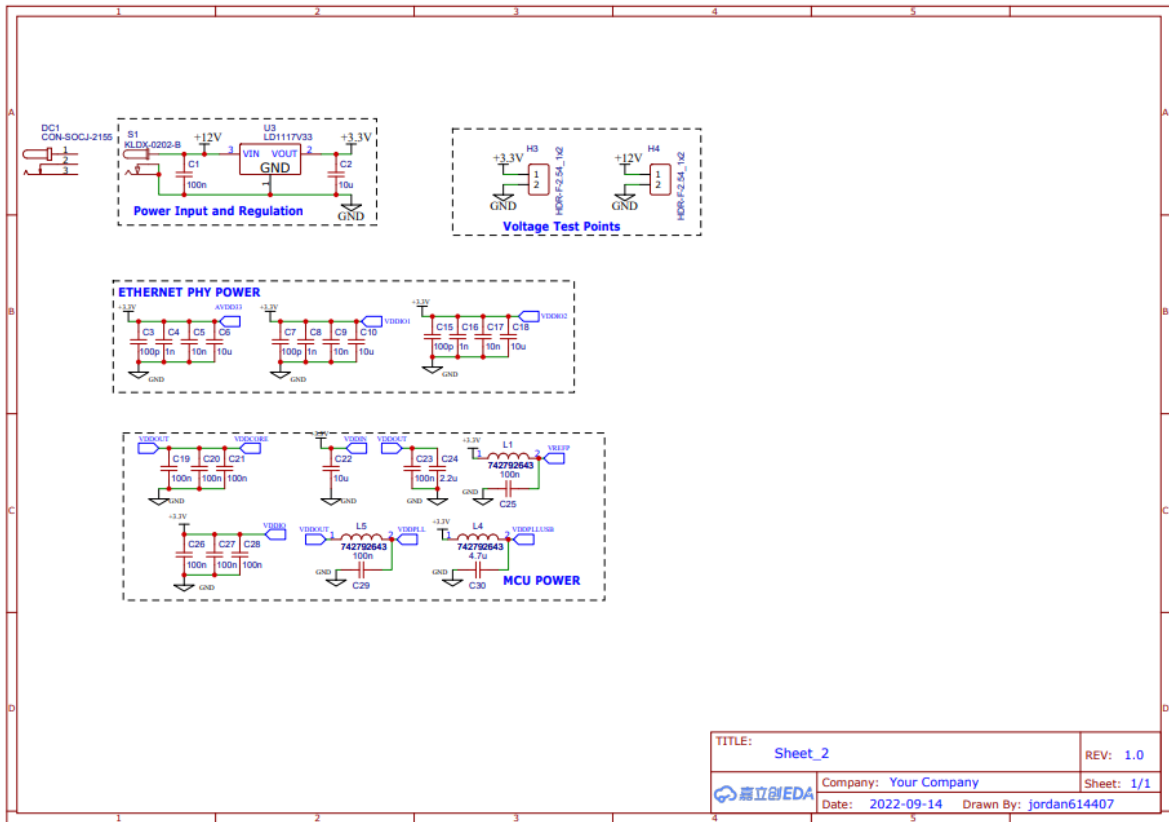


Figure 39.4: Current PCB Schematics (Power)

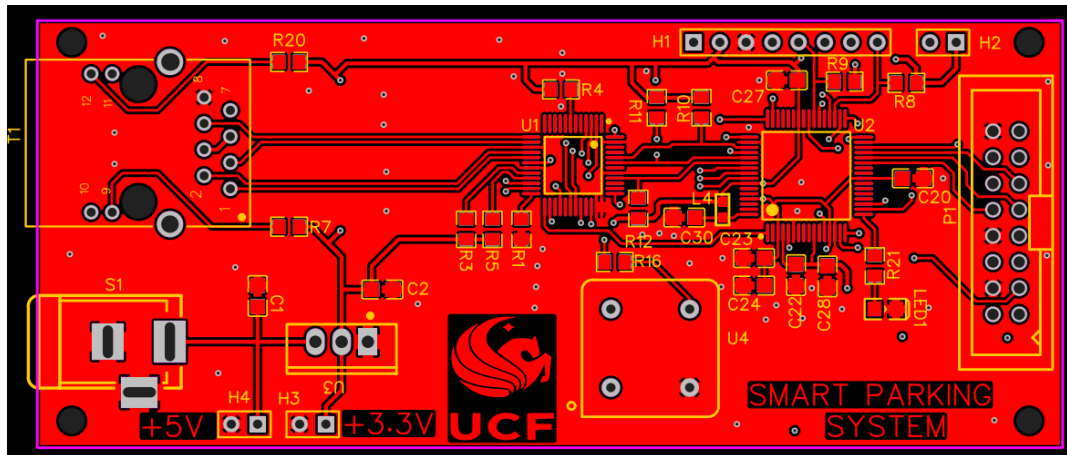
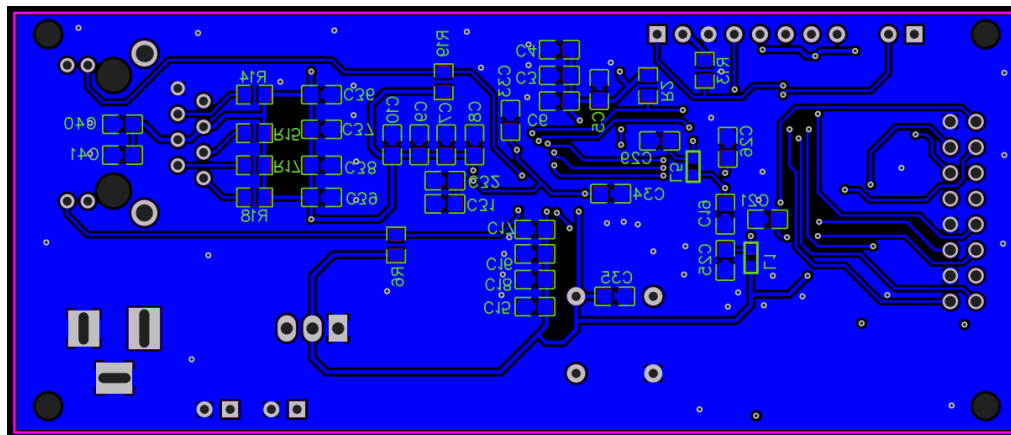


Figure 39.5: PCB Layout (Top)



**Figure 39.6:** PCB Layout (Bottom)

## 9.2 Bill of Materials

Table 23 below displays a BOM for the current design of the PCB. The price per unit is represented in the third column, and the value at the bottom of the table shows the total cost of all components with quantity included. All components that were not passive SMD components, were ordered early so that we could begin our prototyping and testing process. We worked with through-hole components at first, and then when we had our PCB manufactured, we ordered SMD components for the board. This is a final BOM for the PCB and shows what it costs to make each PCB.

**Table 23: PCB Bill of Materials**

Name	Footprint	Unit Price	Manufacturer	Quantity	Price
100n	C0805	\$0.03	Digikey	18	\$0.54
10u	C0805	\$0.07	Digikey	6	\$0.42
100p	C0805	\$0.07	Digikey	3	\$0.21
1n	C0805	\$0.04	Digikey	3	\$0.12
10n	C0805	\$0.04	Digikey	3	\$0.12
2.2u	C0805	\$0.11	Digikey	1	\$0.11
4.7u	C0805	\$0.10	Digikey	1	\$0.10
1u	C0805	\$0.05	Digikey	2	\$0.10
HDR-F-2.54_1x8	HDR-F-2.54_1X8	\$0.65	Digikey	1	\$0.65
HDR-F-2.54_1x2	HDR-F-2.54_1X2	\$0.58	Digikey	3	\$1.74
HDR-M-2.54_2x8	HDR-M-2.54_2X8	\$0.40	Digikey	1	\$0.40
Ferrite Bead 470	SMD-0603	\$0.15	Digikey	3	\$0.45
LED-0805_R	LED0805_RED	\$0.47	Digikey	1	\$0.47
4.87k	R0805	\$0.04	Digikey	1	\$0.04
2.2k	R0805	\$0.04	Digikey	9	\$0.36
470	R0805	\$0.04	Digikey	2	\$0.08
33k	R0805	\$0.04	Digikey	3	\$0.12
49.9	R0805	\$0.04	Digikey	4	\$0.16
100	R0805	\$0.04	Digikey	2	\$0.08
KLDX-0202-B	KLDX-0202-B	\$0.74	Digikey	1	\$0.74
WE-RJ45	74980104400	\$6.20	Mouser	1	\$6.20
TLK111PT	LQFP-48	\$11.34	Mouser	1	\$11.34
ATSAME70J19A	LQFP-64	\$12.63	Digikey	1	\$12.63
LD1117V33	TO-220-3	\$1.32	Digikey	1	\$1.32
ACHL-50MHZ-EK	OSC-TH_4P	\$2.61	Digikey	1	\$2.61
				<b>Total</b>	<b>\$41.11</b>



## **10.0 Testing**

To ensure that our final product for senior design 2 functions as expected, each of the major components within our overall system must go through a testing process to ensure they are working properly. This section is divided into two sections: hardware testing and software testing, and we detail the process each component will go through before being implemented in our final product. Additionally, we will address the results we expect from the testing processes, and the steps we will follow in the case testing does not go as expected.

### **10.1 Hardware Testing**

The subsystems within our overall design that needed hardware testing include the computer vision system, the microcontroller, the PCB, and the PoE switch. The testing procedure for these components is explained in the following sections.

#### **10.1.1 Computer Vision System Hardware Testing**

With the camera being the primary sensing element in our system, the hardware components within the camera needed to be functioning as expected. The testing procedure for the computer vision system hardware was done to verify that the easily testable components were functioning properly. This included powering up, Ethernet connection, data transmission over Ethernet, and verification of the camera sensor working. By nature of the OAK-1 PoE being an All-in-One solution, we were limited in exactly what we could test. Therefore the hardware testing for the computer vision system was not extensive.

The power supply over ethernet needed to be tested at the beginning. If the camera did not get powered on, then that needed to be addressed immediately before any other tests could have been done. The unique power supply setup, more specifically power delivered over the Ethernet or the PoE system was tested following the Luxonis documentations by observing continuous green light flashing on the camera. It was planned that if the camera did not receive power upon connection, the troubleshooting procedure would start at the switch level (e.g., restarting the switch, changing cables, and getting a new POE switch).

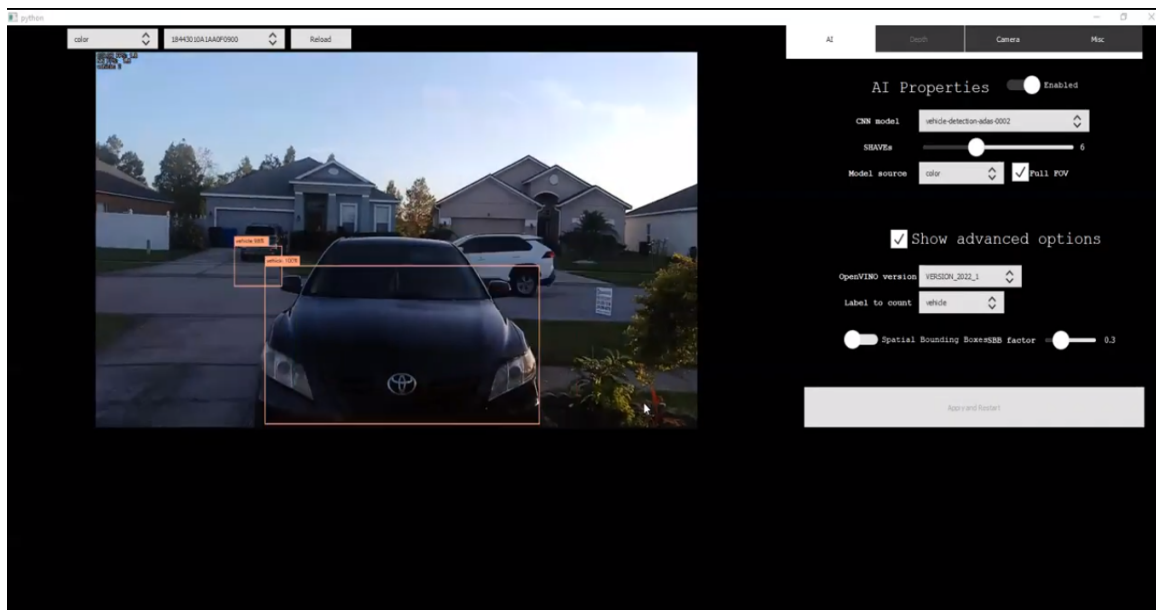
Data transmission over the ethernet was the next step in the testing procedure. For the initial testing, the depthai-demo program was tested and observed. The time it took to get the spatial recognition models running, the maximum amount of time duration the camera could maintain a stable connection, and frame rate fluctuations under certain circumstances were observed for a better understanding of the system implementation.

The stability of the camera mount and the camera angle were tested after. To improve the vehicle detection capabilities adhering to the system requirements, the stability of the mount was tested beforehand. A tripod was used to hoist the camera to capture the video feed from a higher point of view. This height was tested by observing the video feed and adjusting the height of the tripod simultaneously. The wheel locks on the tripod were also

set for better stability. The tripod had to be angled perfectly to look over the chosen lanes to detect the vehicles with maximum precision.

The final aspect of computer vision testing was that the control unit was able to run the spatial recognition model as well as capable of sending the data to the database simultaneously. The recognition model was run for an extended period of time with the database updating at each detection, with the purpose of stress testing the control unit system. The goal was to observe any sort of discrepancy within the system due to the database access and AI models running at the same time for a long period of time.

With these tests all passing, the OAK-1 PoE camera is ready to go for implementation.



**Figure 40:** Luxonis documentation example result of `depthai_demo.py` running

### 10.1.2 Microcontroller Hardware Testing

The microcontroller needed to be tested for a number of its capabilities, including communication over MAC from the integrated Ethernet controller to the Ethernet PHY chip and PWM capabilities for driving the LED display. In order to easily prototype our design, we made use of a breakout board that fit the specifications of our MCU. This breakout board was designed for the Quad flat package (QFP) and a 64-pin layout. Its dimensions were 10 x 10 mm with a 0.5 mm pitch between pins. This fit perfectly with the dimensions of the Microchip ATSAM70J19A-AN, which used LQFP (low-profile QFP) and fit the same standard 10 x 10 mm area with a 0.5 mm pin pitch. The breakout board outputted all of the pin connections to contact pads that allowed for either through-hole connections or surface-mount type connections. This allowed for headers to be soldered onto the board, which would enable the use of jumper cable attachments. Additionally, since the spacing of the output pins with headers lined up with a standard

breadboard, the breakout board was connected directly to a breadboard, making for simple testing and prototyping.

The only way to confirm whether any component was working as desired was to program the microcontroller and run the test code. When testing MAC, PWM, or any other communication interface, a C program had to be written to drive the pins on the microcontroller to output through the corresponding pins for the aforementioned purposes.

For observing the outputs, a multimeter was needed to observe the voltage levels. This way, we could ensure that the pins were within the rated parameters described in the datasheet. When reading data sent over the pin, we needed an oscilloscope to observe the waveforms and a logic analyzer for reading the digital signals. The oscilloscope allowed us to inspect the integrity of the signals, while the logic analyzer read the waveforms as digital signals, giving us more clarity in debugging and analyzing the interfaces.

In our initial testing, all we were looking for was basic functionality from all the needed peripherals on the microcontroller and to verify they behaved as expected. As long as this was the case, we could continue testing the results of code flashed onto the microcontroller without worrying whether there were any defects in the MCU itself or whether our understanding of how the function on the MCU worked was wrong. This sped up the debugging process greatly because fewer variables were in play to contend with when debugging possible points of failure in our design.

In order to run code on the MCU, we needed a proper interface to flash the onboard memory with our code to execute. Like many modern microcontrollers, the Microchip ATSAME70J19A-AN could be programmed through JTAG. Specific to Arm-based microcontrollers, it could be debugged through a 2-wire variant of JTAG called Serial Wire Debug (SWD). A common problem was finding a proper hardware debugger that was both compatible and cost-efficient. There were many vendor-specific hardware debuggers available, but most of them came with a hefty cost. Thankfully, due to the open-source community, there was another option to use any hardware debugger. Through the open-source software OpenOCD, we could use an existing hardware debugger like the popular ST-Link (which is designed for STMicroelectronics devices) for our own Microchip MCU.

With a compatible hardware debugger, we could write any code to our microcontroller for testing. There was another feature that the debugger can do besides being a mode of flashing the MCU; it also debugged the code, as the name implies. This was further highlighted when describing the software side of testing.

In the final implementation, PWM was not needed for driving the display. Instead, a precise pattern of inputs bit-banged to the display was sufficient. Additionally, SWD was utilized but only worked via Microchip's MPLAB® Snap In-Circuit Debugger. For the debugger to work, it had to be used in conjunction with Microchip's IDE MPLAB®.

### 10.1.3 PCB Hardware Testing

The PCB will have a variety of hardware components that need to be tested separately from the MCU, including the Ethernet Port and Cable, Ethernet PHY, and the Step-Down Voltage Converter Circuit. The following sections will detail the procedures we will follow for each of these components before mounting them onto the final PCB.

#### 10.1.3.1 Ethernet Port and Cable Testing

Initially, we wanted the testing procedure for the ethernet port to verify that ethernet data can be delivered to the MCU. Before we had our PCB manufactured, we utilized an ethernet connector breakout board which allowed the eight pins of the RJ45 ethernet port to be easily interfaced with a breadboard. Since the ethernet port we used had integrated magnetics, the circuit between this breakout board and the breakout board used for the PHY chip was fairly simple. From here, we would've prototyped and troubleshooted any connections between pins before making the decision to place the RJ45 ethernet port onto the PCB.

After learning more about ethernet components, we realized that this method would cause issues with signal distortion since ethernet uses high frequency signals and decided to move straight to the pcb. Since the ethernet port will help in linking the Ethernet PHY with the rest of the ethernet network, it was important that we made the correct connections such that there were two channels for transmitting and receiving data with no interruptions. To do this, we followed the schematics available in the datasheet for the ethernet port and development board documents very closely. Additionally, we thought of using an ethernet cable tester between the ethernet cable and the ethernet port to tell us if each end of the cable we used follows the same wiring standards described in section 6.2.1. If the ends of the wire did not follow the same standards, we would've needed to change the wiring inside the cable. Unfortunately, at the conclusion of our project we never got this component to work.

#### 10.1.3.2 Ethernet PHY Testing

The testing procedure for the Ethernet PHY would've verified the data speed we can achieve with our system, 10BASE-T or 100BASE-T, in addition to making sure all connections between the ethernet port and MCU were correct. Since the PHY we used has a JTAG interface, we initially thought we could communicate directly to the hardware chip through a host computer. We would've use this JTAG interface to complete a boundary scan on the PHY chip and ensure that every external wire connected to this chip is on the correct pin. Since we decided to switch over to SWD this method was no longer possible

In addition to the major connections for data transfer between the ethernet port, MCU, and PHY chip, we also wanted to test the status LED drivers and the clock. Testing the status LED drivers would've show us each of the modes that these pins can function and the type of data transmitted with each mode. To test the external oscillator connected to the PHY chip, we simply built this onto a breadboard and verified that the PHY chip was receiving a 50 MHz signal and transmitting a 50 MHz signal to the MCU.

### 10.1.3.3 Step-Down Voltage Converter Circuit Testing

The testing procedure for the step-down voltage converter circuit verified that the voltage signal coming from the DC jack can be stepped down to a level that is suitable to power the MCU and Ethernet PHY without causing any damage. The initial voltage regulator circuit used a 6-pin 1.6-mm x 2.9-mm SOT-23 package, so we used a breakout board that fit this to get access to the pins for prototyping. After working with this for a short time and making the switch to a linear voltage regulator, the testing became more simple as the voltage regulator was a three pin through hole. The power circuit in our design used a variety of electronics, so it is important to mention the type we used. We used surface-mount electronics when we ordered our PCB; however, during the prototyping phase, we used through-hole electronics to model all power circuits.

The step-down voltage converter circuit used a DC jack at the input, so the first thing we did was use a multimeter to measure the voltage across the input and ensure that we were seeing 5 V. From here, before we made any connections between the MCU, Ethernet PHY, and the output of the step-down voltage converter circuit, we verified that we were seeing the expected output voltage of 3.3 V. This confirmed that the voltage output was at a level required for the safe operation of the MCU and Ethernet PHY and no damage would be caused.

In the case that we decided that we would like our design to be a bit sleeker, we had the idea that instead of using a DC jack, we would replace the ethernet port on the board with one that is designed for PoE applications. This would have some drawbacks, though, as this introduces challenges with finding a voltage regulator or buck converter that can handle a PoE voltage input range. With this said we did not follow through with this idea.

## 10.2 Software Testing

The subsystems within our overall design that needed software testing include the computer vision system, the central server, the microcontroller, the web app, and the mobile app. The testing procedure for these components is explained in the following sections.

### 10.2.1 Computer Vision System Software Testing

The computer vision software testing procedure verified that the software could define what a parking space is and receive/transmit data with the central server over ethernet. In the beginning, the testing was done with just recording being sent to the python script. No live testing was done at the beginning stages, which enabled the team to execute the testing and sharpen the accuracy of the detection model in smaller scopes. Several vehicle detection models were utilized and tested to be selected for the final implementation. For example, at the beginning stages of the development, the depthai-demo repository and its vehicle detection model, called vehicle-detection-adas-0200, were tested for accuracy. Later with further clarification of project goals, the depthai-experiment repository was utilized and tested for final project implementation.

The second requirement tested was the models being able to detect correct objects. In a parking lot other than cars, there may be people walking in front of the camera, or there could be different objects captured in the video feed. To make the parking system more realistic, emphasis needed to be put on the capabilities of the model detecting and counting only vehicles. This would mean any other objects would need to be ignored by the spatial recognition model. The vehicle detection model used for the system, named vehicle-detection-0202, made this testing much simpler. This model was specifically pre-trained to detect only vehicles and return certain preset outputs. Thus the testing procedure to ensure only valid objects(vehicles) were detected was extremely simple. The camera was run for at least three hours, and the team observed if people captured in the camera frame or any other objects; created inaccuracy within the detection model. And as expected, the model was not affected by any other objects being within the camera feed and detected only the vehicles available on the camera frame.

Tests for data transmission to the database were also done for efficient and uninterrupted implementation of the parking system. The initial plan was to implement a function that would connect with the MySQL database, and the data would be updated every 5 seconds interval. However, the final implementation had set up the database update function to communicate and transmit value at every valid detection counted towards the leaving or entering label. This functionality was tested for over 300 detected cars and was observed for database updates for accuracy.

With us using advanced computer vision techniques in our design, we needed to have a backup plan for the software design in the case that parking space detection did not function as expected and presented us with too many issues. Some of the methods described below would have required our system to need human input before it can function instead of working autonomously; however, we knew they would be easier to implement and not present as many issues.

The first method we were looking at implementing was using bright-colored squares on each of the parking spots that we want to be monitored. The computer vision software system would then count these squares and return the number of open parking spots based on the number of squares it can see. This method would get rid of all object detection features and be much more simple than the method described in previous sections.

The second method we were looking at implementing would involve using OpenCV's mouse as a paintbrush function. The moment we set up the camera, we would simply use a mouse connected to the server to draw on the screen the parking spaces we would want to be monitored and then define these as our regions of interest. From here, we would then use the chosen object detection method to compute whether a car is occupying a region of interest or not. Using this method would be simpler than our original method but also more complex than the method described in the paragraph above since we would still be using object detection functions.

The final method we were looking at implementing would involve not keeping track of each parking spot but instead keeping track of the number of cars that enter and exit a row of parking spaces. This method does have one drawback being that it would require us to buy a second camera so that we can monitor the entry/exit points on each side of the row; however, it would also allow us to keep track of many more parking spaces. Using this method would negate the constraints caused by the ceiling, as explained in section 7.2, which makes this a very attractive option.

### 10.2.2 Local Server Software Testing

Several tests were required to ensure the correct functioning of the local server. The team needed to ensure that all the components were functioning as designed, such as having connectivity with the camera and with the PCB. One way to check connectivity with the camera and the PCB is by pinging their IP address (e.g., ping 192.168.1.200) from the server. If there is a response from them, it means there is connectivity.

Although the web app and mobile app features of the system were scrapped from the project, the testing procedures between the server and the web and mobile components were left here to show how they could be done.

After establishing the server's internet connection, connectivity to the cloud database must be performed. Since the custom programs were developed in Java, the MongoDB drivers for Java must be downloaded and imported into a small Java program. Then, a new MongoDB client variable is declared and initialized with the internet address of the MongoDB database site, then JAVA is instructed to get the database information from the provided address and provided credentials. If everything is configured correctly, the program will display the database information; otherwise, an error will be shown instead.

Checking the connectivity of the local server with the cameras is straightforward. Once the cameras are connected to the PoE, switched and turned on, and python and DepthAI are installed in the local server, a simple python script provided by DepthAI documentation can search the local network for connected cameras. The script will display the camera's IP address and other camera information. If the script does not find any cameras, the camera may not be on, or it may not have obtained an IP address.

The connectivity with the display's microcontroller is also straightforward. A ping command ran from the Windows command terminal using the microcontroller's IP address which has previously been assigned, can help determine if the server can talk to the microcontroller. Connectivity has been established if there is a response from the microcontroller. At the end of the project, the ethernet portion of the PCB could not be completed; therefore, there was no connectivity between the server and the PCB to test.

It is important to note that the camera and the display's microcontroller are assigned IP addresses before these connectivity tests. The wireless router's DHCP server handles and gives IP addresses to these two devices. Two IP addresses must be reserved beforehand for these two devices using each ethernet port's MAC addresses, so when they initially connect to the network and request an IP address, the DHCP will see their respective

MAC addresses and assign the reserved IP address. This is essential because the local database will maintain the IP address of the devices connected to it in its records, and these addresses must not change.

One last test can be done on the wireless router's Wi-Fi network. We can scan for available Wi-Fi networks using a smartphone built-in Wi-Fi app to determine if the Wi-Fi parking system is on the list. If it does, we can ensure that the phone can connect to that Wi-Fi using a predetermined password.

The test of the Java programs was completed at every step during development, which included writing and reading to the database and sending data via ethernet to a PCB/LED display emulator. There was no connection to the PCB, an emulator had to be developed that could behave as an ethernet device that responded to the server whenever the server tried to send the new number of available spaces to the emulated displays.

### **10.2.3 Microcontroller Software Testing**

Testing C code written for a microcontroller can be more cumbersome than that of PC-targeted C code. When writing code for a PC, it can be quickly debugged since the target platform is the same platform the code is generally written on. When writing C code targeted at a microcontroller, the code has to be compiled and then flashed over to the MCU. This adds extra steps to the process, discouraging programmers from implementing minute changes in code before recompiling and testing again. Additionally, there are few options for observing the output of code when running on a microcontroller, unlike a PC-targeted executable which can be observed in the terminal, with functions like print statements used for aiding in debugging.

There was a solution that helped to alleviate the problem of debugging for microcontrollers, and that was the Serial Wire Debug capabilities of the Arm Cortex-M7 used for our target microcontroller. Through SWD, we could step through code and manipulate the flow of the program to better observe the code's behavior on the MCU. Though we could not see direct output to a terminal through print statements, we could observe the direct effects of the code on the physical outputs through a logic analyzer or oscilloscope, as mentioned in the hardware testing section. This almost eliminated the problem posed by the limitation of needing to flash the target microcontroller when code could immediately be run on a step-by-step basis when it was connected via a hardware debugger.

### **10.2.4 Web App Testing**

Although the web app and mobile app features of the system were scrapped from the project, the testing procedures were left here to show how it was planned to be done.

For web applications, some of the core testing to make sure the smart parking system is catered to the end-users sufficiently are security aspects of user info, the performance of the website while broadcasting video footage, graphics user interface usability also the core functionality of reading the parking data and showcasing them onto the web application.



Functionality testing is indeed the main testing requirement for this project. The web application needs to be able to access the data from the server database and showcase the data on different components added to the website. So, after the computer vision testing, when the data of parking locations and the number of open parking spots have been determined, the web app API needs to be able to gather the data from the local server database. This data further needs to be displayed on the front end of the web app with proper effects and positioning for the end-users to access the data easily. Thus, the receiving and updating of the data and showcasing of the updated data needs to be tested deliberately for proper implementation. This testing can be done through numerous interactions of parking data gathered over time. Frequently changing parking situations and observing the data displayed on the web application will ensure accuracy.

Along with the functionality, efficient implementation of the graphical user interface is crucial. In modern days one of the main drivers of usability and increasing user traffic for a system is to have a user-friendly interface for the consumers to interact with. Smart Parking is planned to have a user-friendly web app interface for the customer to interact with the system with ease. This includes visual representations, complementing color themes, and user-friendly components to enhance the user experience. Since the smart park app is catered to UCF students, a black and gold theme is intended to be applied. The web application would allow multiple pages to be hosted; that way, the data on the web application are not all clustered together. The vacant positions in the parking spots are designed to be in green font color and red for the taken spots. A web page is designed to showcase the live video feed of the parking spots. To test the user experience, multiple students would be asked to try out and browse through the web application. That was, the usability of the web application can be tested, and with the feedback gathered, there would be the potential scope for improvements.

For a steady and reliable connection between the web application and the local server set (which gathers the data from the cameras), interface testing needs to be implemented. Smart Parking web application would communicate with the server database in real-time to ensure a direct data feed for the end-users. Thus, live and agile communication between the application and server needs to be thoroughly checked over a long period of iterations to confirm edge cases. Also, the database can crash, or momentary network interrupts would cause the web application to stop acting accurately. Thus, to prevent mistakes in showing the intended data, a notification system or an error message system must be set and tested to communicate with the users if needed. Interrupts while browsing through the web applications for the users also would be checked.

Web applications for smart parking also should be tested for compatibility. In the present day, there are a lot of technologies that support a lot of different types of web applications. However, depending on operating systems, different devices, and web browsers, the architecture of the web applications must be programmed to support all different types of technologies. Thus, for the smart parking system, the web application should be launched and tested on different devices and different web browsers to ensure complete compatibility across technologies. Responsiveness is also one of the

components that need to be considered. Some end users may use mobile phones to access the web application externally. In that case, the web application must be able to cater to the users' needs. Thus, responsiveness across multiple devices needs to be implemented and thoroughly tested. The web application should retain its readability and accessibility across platforms. An iPad screen view and an iPhone screen view would be different. However, during implementation, the front-end side of the web server must be designed to change as the platforms of the devices are changing. Google lighthouse can be used for compatibility and accessibility metrics.

With functional testing, the concept of performance testing must also be considered. Smart Park app will dynamically access the local database and have a feature to showcase real-time video footage of the parking slots for the end-users on a web page. Any real-time application can put stress on performance metrics. It would require high computational power to constantly communicate with the server and display the video footage to the users. At the same time, load testing is one of the more important concepts as well. UCF being one of the largest schools in Florida, the app potentially would need to support a lot of students accessing it daily. This would stress the performance of the web application due to the huge amount of digital traffic. For the project's scope, even though the web application would not need compatibility with thousands of students, multiple individuals need to be able to access the web application simultaneously. Thus stress load testing must be done on the web application for some iterations over the senior design two periods. This includes testing the connection between the web application and the users. This would ensure web applications have the capability to support a large number of data requests and, at the same time, handle a large amount of data from the database.

Potential vulnerabilities for the web application would need to be thoroughly tested as well. The user login would need to be authenticated for the smart parking application. Features like password strength and password matching while creating and accessing user profiles must be tested on multiple occasions to ensure secure authentication. This needs to be implemented by adding multiple password restrictions while creating a user profile and validation API while trying to access the profile. The restriction could allow standard security measures like having numbers, special characters, and lowercase and uppercase letters. The web application needs to be tested to ensure only correct combinations of usernames and passwords are utilized to access the web application. Right user name and wrong password, wrong user name and right password, wrong user name and wrong password - all three of these scenarios should be checked and tested thoroughly for the security aspects of the web applications. At the same time, the validation capabilities need to be checked and tested. Whether the users can connect and access the website with the correct credentials must be tested on different occasions.

### **10.2.5 Mobile App Testing**

Although the web app and mobile app features of the system were scrapped from the project and not implemented, the tests for the mobile app were left here to show how it was planned to be done.

The Smart Parking system is designed to enable the feature of a mobile app. A lot of the new technology corporations are focusing on developing mobile applications along with web application implementations. This caters to different users, is cross-platform, and plays a vital role in increasing digital foot traffic and improving user experience through multiple outlets. However, for the current project scope, keeping workload and time constraints in mind, a mobile application along with a web application could be hindering productivity. Thus the mobile application part of the smart park system is planned to be designed with barebone concepts.

The functional testing would need to be done across the mobile application. Like web applications, the core functionality is the main aspect of the success metric of mobile applications. Being able to properly handle the functional properties like showing empty parking lot status and the login and logout functionalities. The mobile application must also access the local database to showcase the data gathered. The system is planned to be designed to gather data from the computer vision algorithms from the cameras and make an educated decision on detecting vacant parking spots. Later this detection data would be stored in a local database, and the API would need to communicate with the database to ensure data transmission. Later from the frontend stack, the website would access the API through different requests to transmit the necessary data through web pages. This functionality needs to be accurate and must be tested over several iterations throughout senior design 2. Since the mobile application is not planned to have as many features as a web application, the testing processes would be less rigorous. However, functional testing has to be done to provide the end-users with the most accurate data.

The web application's graphical interface is planned to have an easily maneuverable content interface so that the user experience can be improved to increase digital foot traffic for the system. With the integration of the mobile app and since, in today's world, usage of mobile devices is increasing with time, the importance of a robust graphic user interface is immense. As depicted in the Figma snapshot in an earlier section, the mobile app is designed to follow the UCF color theme of black and white. On top of that, the data is set to display on a light blue box for better visibility, and different font colors are set to be used for vacant spots or parking that are not available. To test the user experience of the mobile application's graphics interface, several students would be asked to use and browse through the application and provide feedback on their experience. Improvements can be made to alter the interface based on the test results.

Like the web application, the mobile application would also require flawless data transmission from the local server to the application. So the communication network and efficiency in communication need to be tested. However, the testing procedure would be similar to the web application interface testing. The process would include observing numerous communication interactions between the database and mobile application and measuring interrupts while using the mobile application. Notification of the error message system would also need to be implemented for any network outage or server crash situations.

The memory storage for mobile devices is not the same as a desktop application. Thus memory tests need to be done to make sure that the memory usage for the mobile app is optimized. This would include figuring out the core components of the mobile application functionality and only utilizing the components necessary for the main function of the system. This would ensure memory optimization and metric performance enhancement.

Performance testing is one of the main tests that need to be done for the web application. However, the web application is set to be able to provide a real-time video feed to its web page for the users. However, the mobile application is set to only showcase the number of parking available within the parking slot. Thus the application would not require as much in-depth computation power. However, performance testing still needs to be done to ensure correct data is being shown to the users at all times. In addition, mobile applications have a new aspect of performance testing. Mobile devices operate on DC power, and depending on the computational operation done on the device; the battery gets drained faster or slower. Thus the mobile application would need to be optimized to preserve battery power. For this aspect of testing, the mobile app is intended to be activated at multiple intervals throughout the day and observe the battery drainage metric over time. Also, the loading and start-up time for the mobile application should be tested while the phone is in regular usage as well as in more intensive usage. This is also done to test the response time while a user tries to put in a request to fetch data from the database.

The mobile application needs to be compatible with multiple operating systems and devices. There are a lot of mobile device technologies introduced daily, and the mobile application needs to be designed to be supported on any device screen size and device operating systems like android or iOS. Thus the compatibility test should include using the application on numerous devices with different screen sizes and observing the changes. Based on the observation, improvements can be made to integrate the application with the different devices.

The login and sign-up options would be provided in the mobile application; thus, security testing is required. Like the web application, password requirements of lowercase and uppercase letters, number, and special characters must be implemented for the mobile application sign-up process. Also, the validation process while logged in needs to be tested so that users can log in with the correct credentials. The same three scenarios need to be tested for security purposes - login with the wrong username and right password, right username and wrong password, and wrong password and wrong username. The mobile application system should reject logging in for all these scenarios.

## 11.0 Mounting and Installation Procedure

Parking systems require proper positioning and a clear view to be as efficient as possible and to serve their purpose to the fullest of their functionalities. Some of the implementations have been discussed in the existing technologies section earlier in this document. Some of the more popular mounting systems include overhead - onto ceiling sensor mounts, on-wall camera mounts, camera, and network sensors mounted on lamp posts, and one of the more innovative applications of drilling sensors within the streets or on the pavement for smart detection purposes. These implementations were researched and implemented in an efficient way to gather data or capture video recordings with optimum lighting conditions and to provide the safest location for the sensors to communicate the data gathered.

Such installations often require extensive measures, such as getting permission from authorities to be able to drill into the pavements or attach additional devices to walls or electric posts around the city, which would also require extra funding set aside just for the mounting procedure. Such costs are usually counted within the total spending budget for the research and development of large corporations. However, for the current Smart Parking Project, such extensive measures for installation and mounting would hinder the development project of the parking system. Thus, much caution and creative thinking were needed for the installation plan module for the project.

The Smart Parking system at the University of Central Florida was planned to utilize computer vision algorithms through video cameras to make educated detections of vacant car parking spots from selected parking slots. With the increasing student count within the university and students returning to campus after a long remote session of the school curriculum, during certain times of the day - the parking conditions in the garages become unbearable - due to the lack of indication systems for the students. The Smart Parking System was planned to be designed to provide students with live video feeds of parking situations and indications of vacant parking spots utilizing LED signs. Therefore, the positioning and mounting of the camera equipment are really important.

The camera equipment needed a clear view from a high/elevated position to record the selected parking spots. Also, the luminosity of the location needed to be optimized for smart detecting purposes. Keeping all these requirement categories in consideration, the UCF parking garage C top floor would be utilized to implement, test, and demonstrate the smart parking system. The top floor of the parking garage would provide ample sunlight as well as space to mount camera equipment in an elevated position to be able to capture the recording of the selected parking area.

The Smart Parking System was designed to use two cameras set in opposite directions to look over approximately ten parking spots. Thus two tripod mounted cameras would be able to capture the spots between them. However, after further testing with the cameras and tripod positioning, the final number of parking spots that were intended to be monitored can be adjusted. The motivation behind the positioning of the cameras was to be able to capture multiple cars positioning from side angles. A depiction of the camera view is added in Figure 41.



**Figure 41:** Point of View from Camera

If the computer vision algorithm failed to detect the cars from this angle in the garage, there was another angle that was in the discussion which needed to be tested. The point of view from the camera from the second angle can be seen in Figure 42.



**Figure 42:** Secondary Point of View from Camera

The smart parking system utilized a switch, Wifi router, and local server for the main control unit. The control unit was the main component of the Smart Parking System, which was practically in charge of communicating between the cameras and transmitting data to the web and mobile applications. The database would directly gather its data from within the local server, and the server would also be running algorithms in real-time to detect the parking situation in the allocated parking slots. Due to the importance of the control unit, a VEVOR electric enclosure box, which is waterproof and dustproof, could protect the control unit in severe weather conditions. Inside the protection box, plywood could be screwed in, and on the plywood, the switch, wifi router, and local servers could be set up for steadiness. Ethernet cables had to be connected from the switch to the cameras for data transmission to the local server. The Wifi router could provide an internet connection for the control unit and enable the communication system.

The box containing the control unit could be mounted on the wall with commercial equipment. However, that would have required extra funding with much more production time and paperwork for authorization. The scope of the smart parking project was not designed to support such installation within the timeframe and budget. Considering the project circumstances, such extensive mounting was not possible; thus, to support the control unit box, it was set to the side temporarily for testing and demonstration. Until further advice from the UCF authority, the temporary setup for the control unit box was considered for the smart park project.

LED signs were utilized for visualization as well. A lot of parking garages had implemented the usage of LED signs displaying the number of available parking spots, the garage levels with open parking locations, as well as directions indicated with arrows through LED signs. Orlando's Disney Springs was one of the corporations that had utilized this method to lessen the parking difficulties for visitors. To enhance the user experience with the Smart Parking system, LED signs gave directions and indicated the number of spots that were in use. To mount the LED signs, there were a lot of wall mounts available. However, the mounting equipment could exceed the budget for the scope of the intended project. Keeping that in mind for demonstration purposes, the double-sided heavy-duty adhesive tape would be used to mount the LED signs on the wall. This way, the effectiveness of the LED signs could be tested as well as future improvements as far as the mounting and installation could be discussed as well.

A few changes in the final implementation started with the use of ethernet on all devices, instead of using Wifi to communicate between the display assembly and the control unit. This reduced complexity in the mounting needs and resulted in only a network switch being needed to route all the ethernet connections. The detection system for the parking garage also changed, resulting in cameras needing to only be at each end of a parking row. Therefore, cameras were situated to the side of the row so that they had an angle (via tripod mount) to the choke point of the row where vehicles entered and exited the region.

## **12.0 Project Operation**

For the proper operation of the smart parking system, there are a few requirements that should be noted. This section outlines those requirements for all systems in our project, including the camera, the server, the PCB, and the LED Display.

### **12.1 Camera and Server**

For the camera to be operational, what is needed is a sufficient PoE connection that can power the camera in addition to allowing for data transfers. As mentioned in section 11.0, a proper mounting system is also required. For how the team implemented the vehicle detection, the cameras must be mounted at either end of a parking row, pointed perpendicular to the traffic flow. This reduces the amount of noise in the camera feed so that primarily vehicles entering the row are seen.

The server requires a 12V/3A power supply for operation. It must be connected to the network switch via ethernet, which is, in turn, connected to the camera and the display assembly. On powerup, the server must launch a script for handling the database, then a script for connecting to the display assembly, and finally launch the GUI. The GUI is optional for the operation of all the LED displays inside the garage, but it is necessary for viewing all of the data of available parking spots on a per-level basis for the garage.

### **12.2 PCB and LED Display**

For proper operation of the PCB, it should be connected to the PoE switch via an ethernet splitter to have both power and data connections. An MPLAB Snap In-Circuit Debugger should be connected to the 1x8 female header pins to allow communication with the chips on board the PCB. Additionally, a ribbon cable should be connected with the 2x8 male header pins to communicate with an LED Display. If the compiler does not recognize the microchip, a jumper cable should be used to connect the ERASE pin to +3.3V to erase the device firmware.

For proper operation of the LED Display, it should receive a 5V connection rated for 4 A from an exterior power source. Additionally, a ribbon cable should be connected from the PCB to the input of the LED Display. This will allow the user to show the desired information on the LED sign. It is important to ensure the LED Display is connected to the input rather than the output used for daisy chaining; otherwise, the LED Display will not work properly.

## **13.0 Project Budgeting and Financing**

The estimated budget we set for ourselves at the beginning of the semester was about \$1,500. A breakdown of the current budget with the components we plan to use is presented in Table 24 below. All financing will be provided out-of-pocket by the team members.



**Table 24: Budget Breakdown**

<b>Item</b>	<b>Description</b>	<b>Qty</b>	<b>Availability</b>	<b>Unit Price</b>	<b>Price</b>
<b>Camera</b>	Oak-1-PoE (with 10% discount)	1	Available	\$224.10	\$224.10
<b>LED Displays</b>	RGB LED Matrix Panel - 32x64	2	Available	\$49.95	\$91.90
<b>Custom PCB</b>	PCB + Components estimate	3		\$48.51	\$145.53
<b>Power Supply (PCB &amp; Displays)</b>	Power Supply 12V/5V (2A)	3	Available	\$10.95	\$32.85
<b>Server Board</b>	ODYSSEY - X86J4125864	1	On backorder (6/3/2022)	\$238.00	\$238.00
<b>Wifi Router</b>	GL.iNet GL-AR750S-Ext Gigabit	1	Available	\$65.61	\$65.61
<b>5-Port PoE Switch</b>	STEAMEMO - GPOE204	1	Available	\$33.99	\$33.99
<b>Heroku</b>	Free Basic plan	1	Available	\$0.00	\$0.00
<b>MongoDB</b>	Free Basic plan	1	Available	\$0.00	\$0.00
<b>Ethernet Cables</b>	25 feet Cat5e ethernet cable 3-foot patch ethernet cables	2 3	Already owned Already owned	\$0.00	\$0.00
<b>Miscellaneous</b>	Parts	1		\$100.00	\$100.00
			<b>Total</b>		<b>\$886.45</b>

## 14.0 Project Milestones for Each Semester

### 14.1 Semester 1 (Senior Design 1)

The major milestones we pursued throughout senior design 1 are shown in Table 25 below.

**Table 25:** Senior Design 1 Milestones

<b>Week #</b>	<b>Dates (Sunday - Saturday)</b>	<b>Milestones</b>
1	1/9/2022 - 1/15/2022	Form Project Group
2	1/16/2022 - 1/22/2022	Begin thinking of project ideas to pursue
		Attend SD Bootcamp on Thursday (1/20)
3	1/23/2022 - 1/29/2022	Submit Bootcamp Assignment on Friday (1/28)
		Begin working on DCV1
4	1/30/22 - 2/05/2022	Finalize DCV1 and submit on Friday (2/4)
5	2/06/22 - 2/12/2022	Attend a meeting with Dr. Richie on Wednesday at 8 AM (2/9)
		Begin working on DCV2
6	2/13/2022 - 2/19/2022	Finalize DCV2 and submit on Friday (2/18)
7, 8, 9, 10	2/20/2022 - 3/19/22	Begin working on 60 page draft SD1 Documentation
11	3/20/2022 - 3/26/2022	Finalize 60 page draft SD1 Documentation and submit on Friday (3/25)
12	3/27/2022 - 4/2/2022	Receive feedback on 60 page draft SD1 Document
		Begin working on next 40 pages of SD1 Document
13	4/3/2022 - 4/9/2022	Finalize 100 page draft SD1 Documentation and submit on Friday (4/8)
14, 15	4/10/2022 - 4/23/2022	Begin working on the final 20 pages of the SD1 Document
16	4/24/2022 - 4/30/2022	Finalize and submit Final Documentation on Tuesday (4/26)

## 14.2 Semester 2 (Senior Design 2)

The major milestones for senior design 2 have been broken down into two phases: the prototype phase and the final product phase. The dates and week numbers that each of these milestones happened are represented in Table 26 below.

**Table 26: Senior Design 2 Milestone**

<b>Week #</b>	<b>Dates (Sunday - Saturday)</b>	<b>Milestone</b>
<b>Prototype Phase</b>		
1,2	8/21/2022 - 9/3/2022	Begin acquiring materials for construction of first prototype
3, 4, 5, 6, 7, 8, 9, 10, 11	9/4/2022 - 11/12/2022	Construct first prototype and begin testing on hardware and software
		If need be, make revisions to the design, requirement specifications, and functions of our project
12	11/13/2022 - 11/19/2022	Finalize any software development needed for the final product
		Construct updated prototype and begin final design testing on hardware and software
<b>Final Product Phase</b>		
12,13	11/6/2022 - 11/19/2022	Work on conference paper
14, 15	11/20/2022 - 12/3/2022	Work with the updated prototype to create the final working product
		Prepare for final presentation (Final Demo, Final Presentation PowerPoint)
		Present and conclude project
		Make updates to SD1 documentation to finalize SD2 documentation

## 15.0 Project Management

With such a large project and only a limited amount of time to complete, it is important to discuss how we managed our team and project throughout the semesters. Project management played a large part in the way our team functioned because it ensured that we were all working well together and following agreed-upon ways of conducting work and communication. This section will describe the tools we used and the methods we followed throughout our design process that helped contribute to our success.

Proper communication is a vital part of any team's success, and some tools that our team used to help us accomplish this were Discord and Zoom. Discord is a popular VoIP, instant messaging, and distribution platform. This application allows users to create unique text channels and designate the type of information that should be discussed in it, in addition to allowing users to enter and exit a voice channel to make for quick voice communication. Discord served as our primary means of communication regarding sharing documents, organizing team meetings, asking questions, and talking about anything relating to senior design. Doing this allowed us to keep all our conversations in one place and keep a record of anything that was talked about.

The second tool we used for communication was Zoom for any virtual meetings that our team had. Zoom is a web conferencing platform used for both audio and video conferencing. The reason we chose to use Zoom for voice meetings rather than Discord is that Zoom gives you the ability to record meetings. Since there were instances where not every single team member could attend a meeting, Zoom allowed us to all remain on the same page.

Our team met twice a week during the design process on Mondays and Thursdays. On Mondays, we would have a virtual meeting through zoom, and on Thursdays, we would have a physical meeting on campus. In these meetings, we would brainstorm ideas for our project, discuss any upcoming deadlines, outline the expectations expected to be met by the next meeting, and discuss in detail how we design our project.

With four people working on one paper, it was important to ensure the paper read as if one person had written it. To accomplish this, a tool that our team chose to use was google docs. While this platform does not have as many functionalities as Microsoft Word, it is free and designed to allow many people to work on one document. Using Google Docs allowed us to see each member's work as it was being written in real time since we were all working on a single document instead of several unique copies. Additionally, this platform allowed formatting issues to be alleviated since we could change what another member wrote.

How we managed our team for senior design one proved to work well; thus, we followed the same structure during the senior design two semester. However, our meetings transitioned from ones where we were brainstorming ideas to ones where we were prototyping them, producing real deliverables, and having discussions on what we needed to do to deliver a final product at the end of the semester.

## 16.0 Conclusion

The smart parking system that we have presented in this paper brought all of our team members' skills to test in addition, it allowed us to see how all of our skills would complement each other. The implementation of sensors aided us in learning about embedded system programming and chip and power design. The implementation of a video camera helped broaden our horizon of skills in machine learning and artificial intelligence development. Our website would have posed as an effective outlet to show creativity in web development through UI-UX design; however, the control unit's GUI program was leveraged highly in that matter.

The ongoing implementation of Smart Parking shows that there is a huge possibility of this sector growing from a business perspective, and from a user endpoint of view, it can be observed that the need for a robust smart parking system will only increase with the flow of time. Acquiring data from the current system providers and utilizing their vision as a fraction of this project's motivation, the team has tried to potentially present a way for a promising system that can revolutionize the technologies implemented for smart parking and aid in lessening the daily hassles of urban city parking issues.

PCB design implications and communication between the camera and control unit via ethernet bus make the Smart Parking Project competitive against the current technology. The project brings in possibilities for technological improvements on the firmware side as well as the web development side. Implementation of segment LED signs can also aid in a fail-safe process if a website outage occurs. Throughout the project, various research and articles have been considered to pick the perfect technology that supports the system. A lot of different technologies were considered and looked into throughout the process, and in consideration of the developers' skill set as well as to stay relevant with the ever-expanding technologies - the product mechanism choices were made. Various standards and requirements were kept in consideration to meet the safety protocols of the system and the individuals during the implementation and the test process.

With the ambitious range, this project has helped the team grow as engineers as well as provide value to not only UCF but a solution that could potentially be able to be utilized globally. The scalability of the implementation of the system in terms of computer vision allows the Smart Parking Garage project to be marketed in various usages. In the simplest form, the project's goal was to assist everyone in making a part of their significant lives easier. As engineers, we were motivated to achieve this goal to serve the community as well as learn new technologies and become a better version of ourselves in the future.

## References

- [1] <https://docs.luxonis.com/en/latest/>
- [2] <https://ams.com/en/belago1.1>
- [3] <https://opencv.org/>
- [4] “The Good and the Bad of Ionic Mobile Development.” Altexsoft.com. Available: <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-ionic-mobile-development/>
- [5] “Benefits of Ionic Framework in Mobile App Development.” Biz4group.com. Available: <https://www.biz4group.com/blog/benefits-of-ionic-framework-in-mobile-app-development>
- [6] W. Rozwadowski, “Pros & Cons of Flutter Mobile Development.” Futuremind.com. Available: <https://www.futuremind.com/blog/pros-cons-flutter-mobile-development>
- [7] M. Berka, “Flutter Pros & Cons – Should You Use it in Your Project.” Invotech.co. Available: <https://invotech.co/blog/flutter-pros-cons-should-you-use-it-in-your-project/>
- [8] K. Shah, “Advantages and Disadvantages of React Native Development in 2022.” Thirdrocktechkno.com. Available: <https://www.thirdrocktechkno.com/blog/pros-and-cons-of-react-native-development-in-2021/>
- [9] S. Vidjikan, “Xamarin App Development: Advantages and Disadvantages.” Softjour.com. Available: <https://softjournal.com/insights/xamarin-app-development-advantages-and-disadvantages>
- [10] “Xamarin.” Microsoft.com. Available: <https://dotnet.microsoft.com/en-us/apps/xamarin>
- [11] S. Watts, and M. Raza, “SaaS vs. PaaS vs. IaaS: What’s The Difference & How to Choose.” BMC.com. Available: <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/>
- [12] <https://newsroom.intel.com/wp-content/uploads/sites/11/2017/08/movidius-myriad-xvpu-product-brief.pdf>
- [13] <https://www.seedstudio.com/ODYSSEY-X86J4125864-Win10-Enterprise-Activated-p-4917.html>
- [14] <https://www.sparkfun.com/products/14718>
- [15] [https://en.wikipedia.org/wiki/Canny\\_edge\\_detector](https://en.wikipedia.org/wiki/Canny_edge_detector)
- [16] <https://www.mygreatlearning.com/blog/introduction-to-edge-detection/>
- [17] <https://stackoverflow.com/questions/45322630/how-to-detect-lines-in-opencv>
- [18] [https://web.ipac.caltech.edu/staff/fmasci/home/astro\\_refs/HoughTrans\\_lines\\_09.pdf](https://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/HoughTrans_lines_09.pdf)
- [19] <https://www.ieee.org/about/corporate/governance/p7-8.html>
- [20] <https://www.osha.gov/sites/default/files/publications/osha3075.pdf>
- [21] <https://inst.eecs.berkeley.edu/~ee122/sp07/80211.pdf>
- [22] <https://www.pcmag.com/encyclopedia/term/cellular-modem>
- [23] <https://store.hologram.io/store/nova-global-cellular-modem/36/>
- [24] <https://www.hologram.io/pricing/flexible-data>
- [25] <https://www.quectel.com/company>
- [26] <https://www.verizon.com/internet-devices/>
- [27] [https://en.wikipedia.org/wiki/IP\\_Code](https://en.wikipedia.org/wiki/IP_Code)

- [28] <https://rainfordsolutions.com/products/ingress-protection-ip-rated-enclosures/ip-enclosure-ratings-standards-explained/>
- [29] <https://ubuntu.com/blog/what-is-an-ubuntu-lts-release>
- [30] <https://www.baesystems.com/en-us/definition/what-are-single-board-computers>
- [31] <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=578032>
- [32] <https://www.protoexpress.com/blog/ipc-j-std-001-standard-soldering-requirements/>
- [33] <https://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/>
- [34] [https://en.wikipedia.org/wiki/IEEE\\_802.3](https://en.wikipedia.org/wiki/IEEE_802.3)
- [35] [https://en.wikipedia.org/wiki/Power\\_over\\_Ethernet](https://en.wikipedia.org/wiki/Power_over_Ethernet)
- [36] Luxonis, depthai-experiments v3.2.0 [Computer software], [github.com/luxonis/depthai-experiments](https://github.com/luxonis/depthai-experiments)
- [37] OpenVino. (n.d.). Vehicle-detection-ADAS-0002 - openvino™ toolkit. OpenVINO. Retrieved November 13, 2022, from [https://docs.openvino.ai/2021.2/omz\\_models\\_intel\\_vehicle\\_detection\\_adas\\_0002\\_description\\_vehicle\\_detection\\_adas\\_0002.html](https://docs.openvino.ai/2021.2/omz_models_intel_vehicle_detection_adas_0002_description_vehicle_detection_adas_0002.html)