Smart Parking Garage System

Oscar A. Acuna, Jordan C. Johnson, Kyle Carpenter, and M. Ridwan

Dept. of Electrical and Computer Engineering, University of Central Florida, Orlando, Florida, 32816-2450, U.S.A

***Abstract* — Finding an available space in a parking garage can be a hassle during busy hours, mainly because driving around a garage trying to find a free spot is a time-consuming task that creates more traffic and delays. The Smart Parking Garage System is a parking guidance system that can solve this problem by reducing the time it takes to find an available spot. The Smart Parking system uses a combination of a server and LED signs to display the number of available parking spots in different car garage areas by using computer vision to determine where cars are parked. A camera runs a vehicle detection model to detect cars on its built-in AI module, determines whether a car is entering or leaving a parking section, and updates a server with its findings. A Java program running in the server uses the camera information to compute the new number of available spaces in that section, then updates a nearby LED display via a custom printed circuit board. The project is being designed as a proof of concept that can be expanded with the right resources and optimizations. A set of strategically positioned cameras and LED displays throughout an entire garage can guide drivers to areas and levels with more empty spaces.**

***Index Terms* — Computer vision, MySQL database, printed circuit board, LED, control unit, graphics interface, computer application, client-server system.**

I. Introduction

In this world of technological advancements, with time, peoples' lifestyles, utilized systems, and methods of life - have been improved. In many cases, these improvements come from making the technology that life revolves around 'Smart.' The word smart, in this context, refers to devices that are interconnected and able to communicate with other technologies. Vehicles are an integral part of everyone's modern-day lives; even though certain aspects are being modernized, when it comes to vehicle parking and associated systems facilitating vehicle parking - there have not been many advancements.

One of the main problems is finding parking spots before the garages reach full capacity in UCF. One modern solution could be the implementation of a system to report the available parking spots in the parking garages accurately. Thus, the Smart Parking Garage System emphasizes making the daily parking experience more modernized and hassle-free with the implementation of technology. Leveraging the functionalities of Computer vision, the system can detect the vehicles entering or leaving the designated parking locations and also keep track of the total count of the cars. This vehicle data is sent to a remote MySQL database utilized by the control unit. The Control Unit allows administrators to see the parking situation in a garage. The control unit, using the count values from the database, computes the total parking spots left in the location and sends the data to the PCB via the IP of the board. Later, an LED system receives the data from the microcontroller, and the computed number of available parking spots is displayed. A web interface was also planned to be worked on. However, due to the limited development time, the web application portion of the system was not implemented.

II. Requirement Specifications

The requirement specifications for the design of the smart parking system are shown in Table 1.

Table 1 Specifications

Component	Specification
Camera	95% accuracy in detecting cars that are entering or leaving a parking area.
LED Display	Minimum brightness of 1,000 nits
Entire System	Update the LED displays within 10 seconds of car detection.
Control Unit	The software and database should be designed to be scalable to support up to 100 cameras and 100 LED displays.

III. Overview of the System

The Smart Parking Garage System consists of four major components: a camera for computer vision, a control unit to keep track of the available spaces, a custom printed circuit board, and an LED display, as shown in Fig. 1.

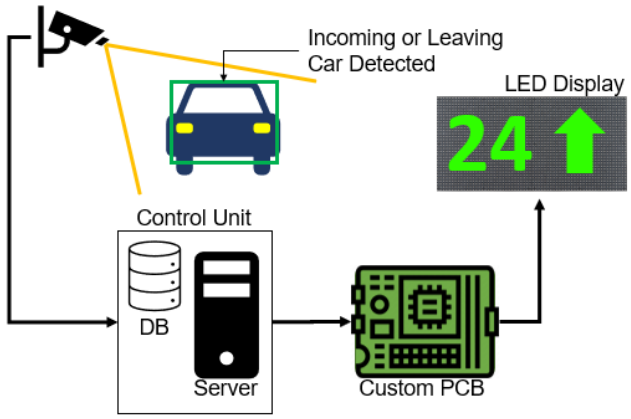


Fig. 1. Parking Garage System Overview

The system starts with the camera's onboard computer vision detecting moving vehicles and determining whether they are entering or leaving a parking row. It sends that information to the control unit, which is inserted into a database running on a server. The camera sends its id number and either a -1 for every car entering to denote one less parking space available or a +1 for every car leaving to indicate that a parking space has become available. A java program running in the server constantly checks for any updates from the camera by searching for new rows in the camera log table in the MySQL parking system database. When it finds one, it computes the new number of available parking spots in the section where the camera is located, which is determined by the camera id, updates the garage level's open spaces, and the total garage available spaces. Then, the program sends the new number to the corresponding printed circuit board using the board's IP address. The microcontroller grabs the new number and updates its connected LED display with the number of free spaces in the parking section it covers.

IV. Design

The project specifications and the team's decisions guided the current design. One decision made was to implement a wired network over a wireless one (e.g., Wi-Fi). A wired network allowed the system to take advantage of power over ethernet (PoE), a technology that enables the injection of electricity into the same cable that transmits data, which could be used to power the cameras and the custom printed circuit boards. In contrast, a Wi-Fi network comprising many access points and wireless devices could create Wi-Fi congestion caused by too many transmissions within a parking garage using the same radio frequencies, potentially slowing down the system and creating interference for other Wi-Fi devices nearby. This and other design decisions are discussed further below.

A. Camera and Computer Vision

Two of the team's goals were to implement computer vision to detect vehicles and choose a powered-over-ethernet device. In addition, to meet the project's scalability specification, the camera needed to have onboard processing capabilities, which allowed for doing all the computer vision computation on the camera itself rather than on a host computer. Having standalone cameras meant a virtually unlimited number of cameras could be added to the parking system without worrying about having a powerful host computer to process all the computer vision computation. Rather than building a camera by putting together several components, the team decided on an off-the-shelf solution; therefore, the OAK-1 PoE camera was chosen. The camera's embedded AI processing capabilities, along with RGB vision, allow the camera to work independently from a host computer and run necessary models to detect objects. This camera differs from all other considered cameras because of the communities' online support, the company's active online support, high onboard computing power, and energy efficiency.

The camera's manufacturer offers many object recognition software models based on DepthAI, a spatial AI platform that allows robots and computers to recognize objects and features and their location in the real world. At an early stage of the project, their car detection model, vehicle-detection-adas-0002, included in their DepthAI demo python script from their DepthAI repository [1], was utilized for the initial testing of vehicle detection and count functionality. For the final implementation, the sample python program, gen2-people-tracker, from the DepthAI experiments repository [2] 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 [4] was imported to replace the people tracker model. The vehicle model conveniently returns the image 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 return 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 (x1, y1) and (x2, y2) pairs, where the 1 indicates the initial location and the 2 the final location of the vehicle, where the origin is the upper left corner of the captured frame. Then, a 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.

In some occasions, a vehicle blocked by another one from the camera view is counted twice. Once when the camera no longer sees the car when it is being blocked, and a second time when the car comes back into the view of the camera when either the blocking car moves or when the car itself moves into the camera view again. To overcome this issue, a threshold of .25 (i.e., 25% of the camera frame width) was implemented. A vehicle has to travel in the x direction for at least the threshold to be counted as moving.

A remote database was utilized for the camera 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.

B. Control Unit

The control unit sits between the cameras and the custom-printed circuit boards (PCB) that controls the LED displays. The control unit, shown in Fig. 2, consists of a server, an ethernet switch, and a battery backup to keep the system running in case of a power outage.

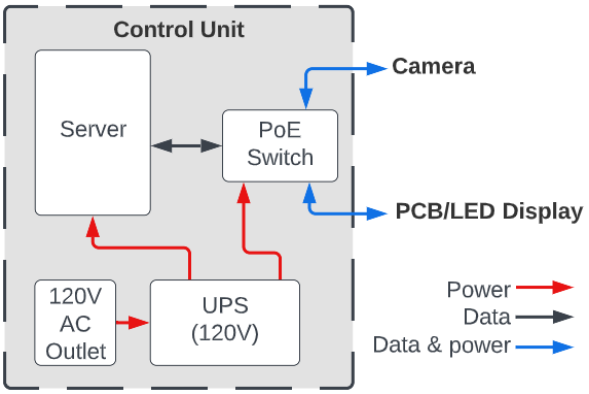


Fig. 2. Control Unit Block Diagram

1. Control Unit - PoE Ethernet Switch

Deploying this parking system to an entire garage, such as Garage C at UCF, would require a switch with many ports, but a small 5-port PoE switch would be enough for a proof of concept. There are different versions of PoE. There is PoE or Type 1, PoE+ or Type 2, all the way to type 4, each backward compatible with the previous version. The camera requires PoE Type 1 (802.3af), and after searching online for PoE ethernet switches, the team settled with a 5-port ethernet PoE switch with four PoE+ ports (STEAMEMO brand, model GPOE204), enough to run up to two cameras and two custom PCBs.

2. Control Unit – Uninterruptible Power Supply (UPS)

A battery backup was added to the system to keep the cameras and the PCB boards running for at least 10 minutes in case of a power outage. The capacity of the UPS needed was calculated as follows: 52 watts maximum power consumption of the PoE switch plus the 36 watts consumed by the server for a total of 88W. The team opted for the UPS model Back-UPS 600 from APC, which provides about 23 mins of electricity at 100W.

3. Control Unit – Server Hardware

One of the most cost-effective computers comes with a built-in CPU and Memory RAM and usually has no PCIe expansion slots, such as a Raspberry Pi. These boards are called Single-Board Computers, and there are many options to choose from. 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.

4. Control Unit – Server Software

The server hosts the system's database, a program that tracks the available spaces in the garage, and the graphical user interface (GUI) program to interact with the parking system. Although several database engines and programming languages could run on Windows, the team decided to use a MySQL database and Java as the primary programming language for one main reason; one of the team members already knew how to integrate these two technologies.

The MySQL Workbench program is part of the MySQL engine, and it was used to create and manage the database and to load sample data for the development and testing of the java programs. Several database tables were designed to keep track of the available spaces in each section, each level, and the entire garage. The database also includes the camera log table, where the camera inserts the change in available spaces (i.e., +1 or -1, as explained in the overview section). For every new update from the camera, the database engine automatically marks that row as new by setting the field isNew to 1. Fig 3 shows a sample of the camera log table design.

	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

Fig. 3 Camera log table in MySQL database.

A program was needed to keep track of the parking spaces' availability by reading the camera log, updating the sections, level, and garage data in the database, and sending updated values to the LED display's PCB. So, a custom program was developed to constantly check the camera log table for those rows where the isNew field is 1. The code grabs the ±1 from the new row to compute the new open spaces for that section and transmits the result to the corresponding LED display via ethernet using the IP address of the display's board; it then sets the field isNew to 0 in the camera log table to avoid processing them as new again. This program is the heart of the Smart Parking Garage System and runs in the background independently from the graphical user interface (GUI).

A GUI was developed to manage the parking system. Written using JavaFX and Scene Builder, the user-friendly program was designed with the essentials in mind. As shown in Fig. 4, the GUI has several sections to allow the user to see the current usage of the garage, change settings, and add, remove, and configure cameras and LED displays.

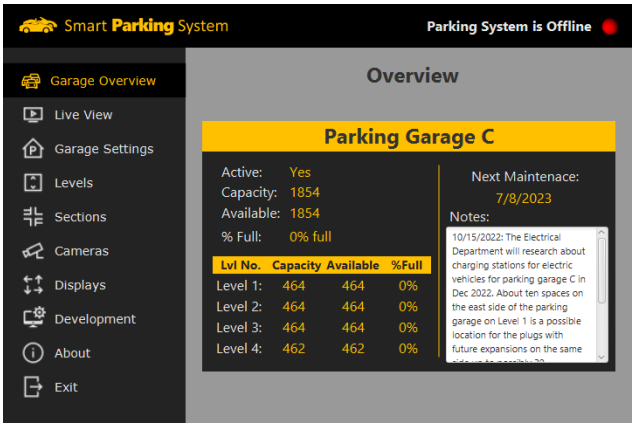


Fig. 4. Graphical User Interface

C. *Custom Printed Circuit Board (PCB)*

The custom PCB in our smart parking system serves as the link between the control unit and the LED signs. It receives data on available parking spots from the server and then uses this information to display it on the LED signs. In our final design, there will be two PCBs, as each LED sign will have its own PCB connected to it.

One of the goals with the PCB was for it to receive both data and power from the PoE switch. This was accomplished by using a PoE splitter which takes the cable from the PoE switch, carrying both data and power, and isolates them onto two separate cables. The data is transferred to another ethernet cable while the power is transferred to a 5V 2.4A DC Male Plug. A simple block diagram of the PCB is shown in Figure 5, where the black lines represent data, the red lines represent power, and the blue lines represent data and power.

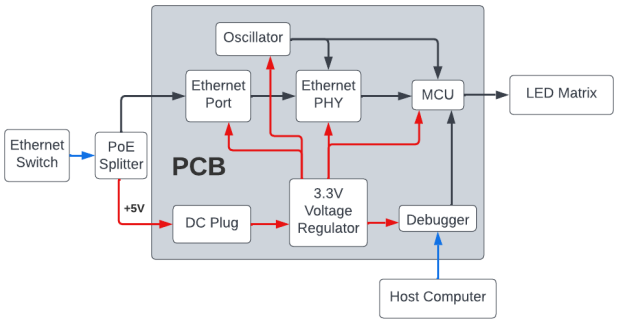


Fig. 5 PCB Block Diagram

1. PCB Components

The components on the PCB, excluding the LED Matrix, are listed below:

- 5V PoE Splitter
- LD1117V33 Linear Voltage Regulator
- 2.5 x 5.5 mm Barrel Jack Connector
- RJ45 Ethernet Port with Integrated Magnetics
- ACHL-50MHz-EK Oscillator
- ATSAME70J19B Microcontroller
- TLK111 Ethernet PHY Chip
- Three 2x1 Female Headers
- One 1x8 Female Header
- One 2x8 Male Header
- MPLAB Snap In-Circuit Debugger
- Various SMD Components

Two of the biggest influences on decision making when it came to picking each of these components were what would allow us to have the least amount of components on our PCB as possible, and what would work well with our microcontroller. Since the microcontroller required a 3.3V power input, we only picked components that also required a 3.3V connection, allowing us to use only one regulator circuit. The component choices affected by this included the Ethernet PHY Chip, the 50 MHz Oscillator, and the RJ45 Ethernet Port.

Continuing on the regulator circuit, one component that has raised some questions was using a linear voltage regulator rather than a switching voltage regulator. Our decision to use a linear regulator comes from the fact that our board does not pull a lot of current (150 mA at the most). Because of this, regulator efficiency was not something that we were concerned about. With this said, after testing, we did decide to change our voltage input in our original design from 12V to 5V as this made more sense for reducing heat dissipation from the regulator.

The header pins that exist on the PCB give access to components that are external to the board. The three 2 x 1 female headers are used for voltage test points and erasing the microcontroller if necessary. The 1x8 female header is used to interface the debugger with the PCB to communicate with the host computer, and the 2x8 male header is used to interface with the LED sign.

2. Board Layout

The PCB for the smart parking system implements a two-layer design with components soldered on both the top and bottom of the board. The top of the board includes SMD components along with all the major components, including the ethernet port, barrel jack connector, oscillator, microchips, and header pins. The bottom of the board only includes SMD components such as resistors, ferrite beads, and capacitors.

There are two reasons we decided to design the board this way, with the first being that we wanted a PCB with a compact design of 100 x 100 mm. Our design satisfied this goal with final dimensions of 99 x 43 mm. The second reason is that the ethernet port, microcontroller, and ethernet PHY required that some of their SMD components, such as bypass capacitors, be as close to their respective pins as possible, which required us to place some of these components on the bottom of the board. Although this was a requirement to our design that we were not initially expecting, it further helped us reach our goal of having a compact design since placing SMD components on the bottom saved us a lot of space on the top layer of the board. Images of the top and bottom of the board are shown in Figures 6 and 7.

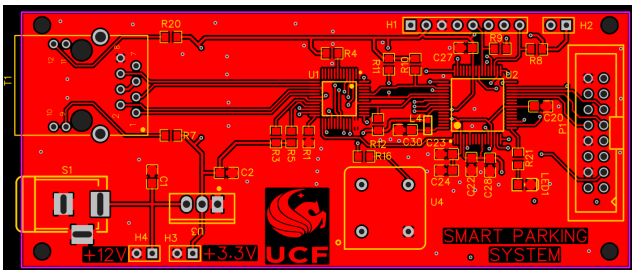


Fig. 6 PCB Layout (Top)

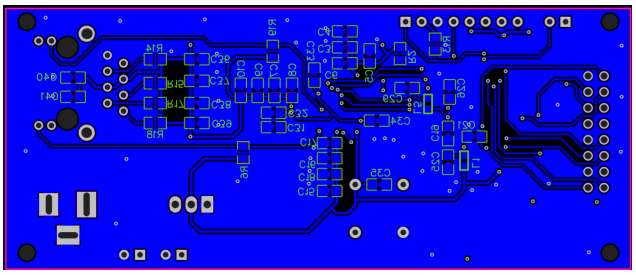


Fig. 7 PCB Layout (Bottom)

D. LED Display

The LED display is the primary interface for drivers in the parking garage to determine available parking spots. The displays are simple in nature, only providing an arrow indicator and the number of available spots. The display has to be adequately bright (hitting the 1000 nits baseline) and have clearly decipherable numbers and icons.

The physical dimensions of the display are 5 x 10 inches with a resolution of 32 x 64 pixels. This results in numbers with a maximum height of 5 inches, making them adequately visible from distances within the parking garage. The symbols on the display will be easy to distinguish with basic fonts for the numbers, as shown in Figure 8.

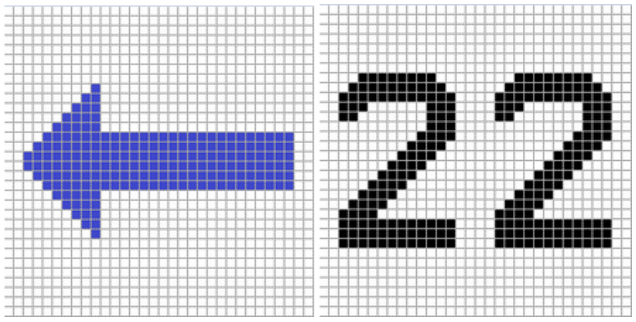


Fig. 8 Example Display Content

1. Powering the LED Display

To supply power to the LED Display, we are required to input a regulated 5V connection. This power input reaches the LED display through a 4-pin female Molex connector which is terminated on the other end with a pair of spade terminals, one for Vcc and one for GND. In a worst-case scenario, one of these 32 x 64 LED displays can pull up to about 3.4 A, so we have decided to use a 5V 4 A switching power supply to meet these needs [5]. This switching power supply is terminated with a 5.5 x 2.1 mm male DC plug, so we will be using a female barrel jack adapter to interface with the spade terminals on the cable connected to the LED Display.

2. Driving the LED Display

In order to interface with the LED display, a number of connections are needed:

- 4 Demux lines
- R0, G0, and B0 lines for rows 1 - 16
- R1, G1, and B1 lines for row 17 - 32
- Clock
- Latch
- Output Enable
- GND

Since the display cannot continuously drive all 2048 pixels without having exorbitant hardware requirements and excessive power draw, it utilizes a 1/16th scan rate. This results in two rows displaying at any given moment before another set of two rows are driven. The display utilizes 24 LED controllers, which output to specific regions of the display with specific colors (red, green, or blue). To get a specific pattern, the desired order of high or low values needs to be clocked into the LED controllers and shifted into each following LED controller (8 in total per color, 4 per row). Once all the LED controllers are filled, the latch is triggered, and the values are pushed to the pixels. The only obstacle left is the output enable being triggered, then the display updates to the new data.

Normally an LED display would have existing driver code and libraries available to easily get started communicating with it, but in our case, that was not an option. The existing code is targeted at Arduinos and is built on existing Arduino libraries, making it extremely complex for porting over to another platform. With the options limited, the choice became obvious to write a custom and simple driver for the display that would suit our needs. Through resources from Ray's Logic blog post [6], we were able to determine how the display operates as described previously.

The current driver code handles driving the display using loops that iterate through each row address combination by driving the demux pins with specific values (2^4 = 16 total addresses). When a specific address is set, two rows 15 lines apart can have data pushed to them through the R0, G0, B0, and R1, G1, and B1 lines. Data is pushed into these lines 64 times, then latched and sent to the pixels. This entire process must happen at a high enough speed for the display to not have flickering discernible to the human eye, hence the choice of the ATSAME70J19B microcontroller, which can run at 300 MHz, making it more than capable of driving the display.

2. Ethernet

The ethernet controller onboard the MCU is referred to as the 'GMAC' and implements a 10/100 Mbps ethernet MAC. Though it is referred to as the GMAC, it does not support gigabit ethernet.

Implementing ethernet on our microcontroller (MCU) was thought to be comparatively easy to that of other components since the ethernet controller was integrated into our MCU. Our design would not have required the microcontroller to send any transmissions to the server (except for the initial handshake) but to instead listen for any new values sent in by the server and, in turn, route the data over to the LED display handler. Since the MCU only needed to receive data over ethernet, only one direction of ethernet functionality needed to be prioritized.

Ethernet being a common interface, seemed to make it easy to find information on best practices or methods to implement into our code. For handling ethernet transactions, the MCU was to receive ethernet hardware interrupts whenever a new packet arrived, at which case the value from the data packet would be passed into the function to display a value on the LED display. By utilizing hardware interrupts, the design would perform far more efficiently than a software polling solution which could result in a loop checking for new changes before every cycle of driving an address of the LED display.

However, the ethernet functionality was not successfully completed by the end date due to a number of problems related to porting the needed TCP/IP stack over to our MCU. The existing TCP/IP stack provided by Microchip lacked information on porting it over to other MCUs and was extremely complex. The recommended Harmony software by Microchip also did not create working ports of the stack either, resulting in generated designs that yielded no results due to improper connections and configurations with the PIO. With more time, we may have been able to port a new TCP/IP stack like the open source lwIP or managed to fix Harmony's code generation problems.

V. Testing

At the time of this writing, the integration of the whole system has not been tested yet. Although a preliminary integration test between the camera and the control unit was completed, the PCB's ethernet portion was still in development.

The main computer vision testing was done in the UCF parking garage C. The first test was done at night, on the first floor of the parking garage, overlooking multiple lanes. The Smart Parking Garage system is designed to have a camera detecting the cars entering or leaving a parking row (i.e., monitoring two lanes only, one entering and one leaving). The vehicle detection model did not perform well when more than two lanes were in the camera's view because it caused the camera to count those cars moving in the background occasionally. Thus in the first test of a batch of 56 cars, the camera had an accuracy of 87.5%. In most of the inaccuracies, the camera detected those extra cars in the background. To overcome this issue, the camera angle was adjusted to have the desired two lanes in the field of view of the camera as much as possible. Additionally, the code was modified to ignore any car captured in the top 25% of the camera video feed since this region was still capturing the other parking row over. A second testing batch was done with 72 cars passing by, from which 68 were detected and counted as expected, resulting in a 94.44% accuracy rate. The team is still investigating ways to improve the accuracy even more. The testing results highlighted the need to calibrate a camera for the area it monitors, as camera angles and regions of interest need to be adjusted to obtain the highest accuracy possible.

Testing the main java program was partially done by having it read the database looking for new data inserted by the camera, and updating the database with the new number of available spaces. The program successfully read and updated the database according to whether more spaces became available or not. The GUI was tested by loading the database with sample data and then having the GUI program read the entire database and display it on the screen. Additional tests will be done very soon.

VI. Conclusion and Reflections

The Smart Parking Garage System tries to solve everyday parking difficulties by leveraging the latest advancements in Computer Vision, software, and hardware implementations. The current system introduces the possibility of utilizing multiple cameras to aid the overall parking experience and a proof of concept of a web-based app for the users. With more time, the team could have finished the ethernet portion of the PCB to have a finished product.

Implementing software development, database usage, administrative graphics interface, computer vision for detection and firmware development, a customed PCB,

and LED implementation to construct a fully functional system makes the Smart Parking Garage Project unique. Such implementation would bring engineers from different practices together and collaborate to develop and improve the system further. This project has enabled the team to work in a collaborative environment where teammates have helped each other to achieve a similar goal despite the constraints of time limits or less experience in the respective technologies used.

VII. Future Improvements

One of the initial improvements the team agreed on was the usage of multiple cameras. However, as a Senior Design project, the team did not have enough financial capability to afford multiple cameras for a more robust implementation; but, with more resources, the system can be upgraded to cover more locations.

One of the fastest and most convenient ways to reach people is through mobile or web applications. A web app and a mobile application to reach consumers would take the system's functionality to another level. The team explored this solution; the team researched and decided on a MERN stack for a web-based and the phone app for the consumers to use. However, after extensive discussion between the team and the advisor halfway into the semester, it was decided that implementing such a web and phone application would not be possible within the time limit. Thus, the web and phone app were left as a stretch goal.

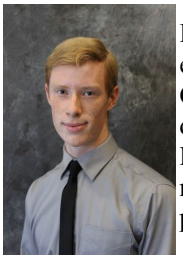
Acknowledgment

The authors wish to acknowledge the assistance and support of Dr. Samuel Richie, who provided guidance throughout the entire process of this project. The authors would also like to thank the support teams of Microchip and Luxonis, who also guided our project's design.

Biography

Oscar A. Acuna, a computer engineering student at the University of Central Florida, developed the database and the Java programs and contributed to the camera vehicle detection and LED matrix driver programming. After graduation, he plans to become a full-time employee at NASA Kennedy Space Center, where he currently interns.

Jordan C. Johnson, a 6th-year electrical engineering student at the University of Central Florida, developed the PCB for the parking system. After he graduates, he plans on starting his career at Universal Creative or Burns & McDonnell while pursuing his MBA and P.E. license.



Kyle Carpenter, a 5th-year computer engineering student at the University of Central Florida, developed LED display driver code, ethernet driver code, and MCU software. Plans to work in fields relating to Computer Architecture and processor softcore design.

M Muhtasim Ridwan, a computer engineering student at the University of Central Florida, contributed to the computer vision and database update for the parking system. After he graduates, he plans on starting his career as an automation engineer, where he currently interns.

References

[1] Luxonis, DepthAI Demo v3.2.0 [Computer software]. https://github.com/luxonis/depthai

[2] Luxonis, depthai-experiments v3.2.0 [Computer software],https://github.com/luxonis/depthai-experiments

[3] Luxonis. (n.d.). Converting model to MyriadX blob¶. Luxonis. Retrieved November 13, 2022, from https://docs.luxonis.com/en/latest/pages/model_conversion/

[4] 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

[5] Sparkfun. Tutorials. RGB Panel Hookup Guide. Powering the Panel. Retrieved November 14, 2022, from https://learn.sparkfun.com/tutorials/rgb-panel-hookup-guide?_ga=2.114944391.1425153588.1668401408-493762454.1647527398#powering-the-panel

[6] RaysLogic. Adafruit RGB LED Matrix. Retrieved November 14, 2022, from https://www.rayslogic.com/propeller/Programming/AdafruitRGB/AdafruitRGB.htm