The Garaginator: Parking Assistant

Jonathan Staudt, Elliot Rodriguez, Sebastian Rodriguez, Jonathan Gillis

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* -- *This* paper will describe the design and methods used to create a fully functional and cheaply produced parking garage monitoring system for currently existing garages. This monitoring system uses optical sensors at the entrances and exits of the garage to tally the number of current cars in each garage. The Garaginator system will also utilize a kiosk which will allow users to save their car's location for future reference upon returning to the garage. Garage space and availability information will also be easily accessed through a web application.**

Index terms -- Optical sensor, database system, web application, QR codes, ATMEGA328P, C++, ESP8266.

I. Introduction

A common problem among people using parking garages is finding a spot to park when arriving. At the University of Central Florida, parking is such a problem that it is not uncommon for students to be absent or late for quizzes and tests, simply because they could not quickly and efficiently find a parking spot in which to place their car. This often happens despite arriving on campus with an amount of time in excess of the average amount of time, which is an inordinate amount, beforehand. This is despite students paying over $50, at minimum, for a parking pass for one semester. Since public transportation in the city of Orlando is severely lacking, bordering on absent, most students have no other option than to drive to class every day. Another problem that the Garaginator aims to solve, a problem which a commuter may encounter after a long day at work or school, is that of forgetfulness. After spending a substantial amount of time away from the parking garage, and thus the car, it is not uncommon for one to forget where one parked. To this end, the Garaginator will implement a vehicle tracking system.

Numerous attempts have been made in the past to solve this problem, but have ultimately been too costly to implement on a large scale or are not feasible to implement in an already existing garage. As such, the Garaginator system was designed first and foremost with cost in mind. The Garaginator consists of three main parts: optical sensors located at the garage's entrances and exits, a kiosk, and a web application. The sensors are connected to the kiosk which can interpret the data the optical sensors send; this data being whether a car has exited or entered the garage. In addition to interpreting signals from the sensors, the kiosk can interface with the webserver, posting information about the volume of cars currently in the garage and recording vehicle locations. The kiosk has a keypad and a display so that users can interact with the kiosk. User interactions with the kiosk consist of entering user unique identification information, inputting vehicle location, and retrieving vehicle location information. A web application also interfaces with the server, and allows users to query the last recorded location of their vehicle, as well as view how many cars are currently parked within the garage. Lastly, a QR code printed and added to every parking spot can be scanned by the web application, allowing them to enter their vehicle's parking information more easily.

The kiosk will be powered from the power mains, via a wall outlet, and will contain a linear voltage regulator. The kiosk will communicate with the webserver using an ESP8266 to communicate over Wi-Fi. The sensors will be optical sensors which consist of one photodiode and one laser per sensor. The laser will be driving the photodiode to be constantly outputting voltage. When a vehicle obstructs the laser, and prevents it from shining upon the photodiode, the photodiode will stop conducting the voltage output will go low, thus indicating that a vehicle has just entered or exited the garage. The ATMEGA328P, which is the MCU that drives the kiosk, will be able to differentiate whether a car has entered or exited a garage depending upon which pins have gone low. The optical sensors will be hardwired to the kiosk. Hardwiring the sensors serves the purpose of saving money on the cost of wireless transceivers and microcontrollers.

The webserver acts as the main back end component that the user will never interface with. It will handle all requests from both the web application and the kiosk application. It will handle updating the database when the optical sensors send signals to the ATMEGA328P, as well as send the location of a user's vehicle to ATMEGA328P when requested. As stated previously, this communication will be obtained through the utilization of an ESP8266 module. The webserver will also authorize use of the system when a user attempts to login. The webserver will also store a user's vehicles location, and send a simple completed message to the user once complete. The webserver hosted utilizing a node.js framework.

II. Project Requirements

The Garainator system's primary mission is to be a cost effective and reliable solution for parking management, while allowing the application's users clear parking availability information. The below requirements reflect this mission and delineate the specific criterion for making this mission a reality.

- The system will be able to count the vehicles entering and exiting the garage with minimal error. Cars should generally be counted entering and exiting as close as possible to exactly once each. Sensors shall be able to update within five seconds.
- The kiosk should be quick and easy to use. Students often wait until the very last minute to drive to campus, especially for ones early in the morning. If it is not quick and convenient many students will not want to use it. The kiosk system shall be able to respond to input commands within five seconds.
- The system should be able to determine if a vehicle and only a vehicle entering the garage. The system shall utilize two sensors per entrance or exit and AND the signals from each sensor together. This will reduce false positives, such as an animal or some other being entering or exiting the garage, from triggering the sensors.
- Users shall be able to enter in their vehicle information and their parking spot to a kiosk system to be remembered.
- If a user has stored their vehicle's location they should be able to retrieve it via the kiosk or the website, if they enter their correct user information.
- The Garaginator system will be able to operate in the extremely inclement climate that Florida plays host to.
- The system will be able to provide information about parking and the garages all week, all day, and all year, to provide a reliable and informative system to users.
- Each sensor will draw less than one Watt of power
- The MCU shall send information about parking to the webserver no less than once every thirty seconds
- The Garaginator system shall utilize standard 110V AC and convert this to the required DC voltages used within the Garaginator system

III. The Garaginator System

The Garaginator system is comprised entirely of four distinctly different areas, which all come together to create a beautiful product. These four areas are the hardware/PCB design, the microcontroller and its firmware which processes data, the webserver hosted using node.js, and the web application.

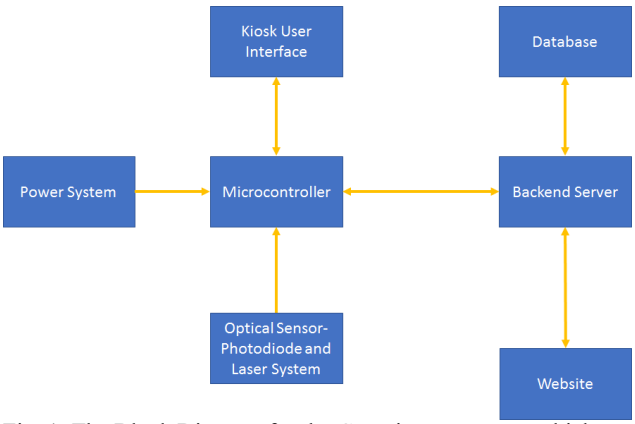

Fig. 1. The Block Diagram for the Garaginator system, which displays not only the various parts that make up the system, but also the signal flow.

The block diagram above clearly displays the various parts and subsystems which serve to make the overall Garaginator system. The power, which is the standard 110V AC converted to DC voltages for use within the Garaginator system, powers the microcontrollers as well as the sensors. The microcontroller receives data from the sensors. This data serves to tell the microcontroller whether a car has exited or entered the garage. The microcontroller receives data from the kiosk user interface as well, this data is unique user data which serves to store where a user car has been parked. The microcontroller also sends data to the kiosk user interface. This data is sent when a request to see where their car was parked. When this request is sent, the corresponding data about the location of the vehicle is sent by the microcontroller.

Data from the sensors is processed by the microcontroller and then sent to the backend server. From this server, the data is sent both to the website and the database. Both the website and the database send information back to the backend server. The database will update the backend server which will in turn update the microcontroller. The website updates the backend server with information regarding user information and logins which then updates the microcontroller, aka the kiosk system.

IV. Design

A. Sensors

The section serves to provide information about the optical sensors, which the Garaginator system uses to detect whether a car has exited or entered a garage at any given moment in time. Other options were considered, such as smart video systems as well as ultrasonic sensors. Ultimately these sensors were dismissed due to their wide cone of sensing and their inability to reduce false positives. The optical sensors, when used in conjunction with an AND gate to AND the signals togethers, serve to provide a small cone of sensing, as well as an incredible degree of accuracy. Not only this, but the optical sensor is incredibly simple, consisting of a power source, a resistor, a photodiode, and a laser dot diode. The optical sensor must be mounted in a fashion which places the optical sensor in question in an orientation which is perpendicular to the entrance or exit of the garage. This is the optimal placement for the sensing of vehicles entering or exiting a garage for the optical sensors utilized by the Garaginator system. False positives in the Garaginator system are defined as the optical sensors being triggered by anything entering or exiting the garage that is not a vehicle, that also triggers the microcontroller to send a signal to the webserver that a vehicle has entered or exited the garage. The way that the Garaginator system reduces false positives is by having two optical sensors per entrance or exit and running the signals from these optical sensors through an AND gate. In this fashion, the microcontroller does not trigger unless both optical sensors have been triggered, which will only happen if an object as long as a vehicle triggers the optical sensors.

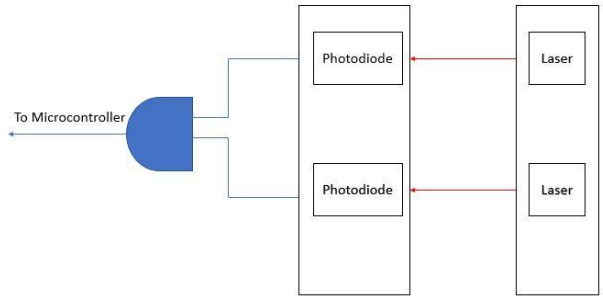

Fig. 2. Mockup of optical sensor system. The AND gate can be seen on the left, with the laser bridging the entrance to activate the photodiode.

A mockup of the optical sensor system and the method utilized to reduce false positives can be seen in the figure above and the specifications for this optical sensor system are shown in the tables that follow.

5V DC Supply
Working Temperature Range of -10°C to +40°C
650 mW Power
0V DC Ground

Fig. 3. Laser Dot Diode Specifications

Working Voltage	5V DC
Maximum Detection Range	~3 – 4 meters
Minimum Detection Range	~1 centimeters
Cone of Measurement	0 degrees
Maximum Output Voltage	1V DC
Resistor Load	2000 Ohms
Product Dimensions	(8.89 x 5.842 x 1.016) centimeters
Weight	17 Grams
Operating Temperature	-10°C to +40°C

Fig. 4. Photodiode Specifications

B. Wi-Fi Transceiver Module

Wireless communication is a large enabler of the Garaginator system. Due to this requirement of wireless communication through Wi-Fi between the microcontroller and the webserver, which ends up updating the database and the web application, it has been decided that the ESP8266 will be used with the Garaginator system. The ESP8266 uses up to 2.4 GHz bandwidth. Utilizing the ESP8266 module, the ATMEGA328P will be able to communicate wirelessly with the webserver, thus allowing the database and web application to be updating with the sensor information which contains the information about car entrances and exits. The ESP8266 was the most elegant and simple solution to the problem of communication with the webserver, due to only requiring the one ESP8266 module and therefore not requiring any superfluous hardware and or software. Only one ESP8266 module will be used and that will ESP8266 module will be used with the ATMEGA328P microcontroller that forms the so-called brains of the kiosk. The ESP8266 module contains eight pins that it utilizes for power and sending of data. Not only this, but the ESP8266 also has an antenna built right into the module. The ESP8266 module utilizes two UART pins, a receive and a transmit pin respectively, to send and receive information from the ATMEGA328P microcontroller. The ESP8266 also contains three GPIO pins. Finally, the ESP8266 has two power wires, as well as a ground wire. At one point in the development cycle of the Garaginator system, it was considered that the ESP8266 module would be used with the sensors to wirelessly send signals that a car has either entered or exited the garage. This experiment in wireless optical sensing was eventually scrapped and pushed to the wayside due to requiring too many modules and microcontrollers. If this idea had been realized, a microcontroller would have been needed for each optical sensor, which would not only drive up cost but also make the overall Garaginator system rather cumbersome. The largest advantage that the use of the ESP8266 module affords to the Garaginator system, is mostly on the development side. This particular module is ubiquitous amongst hobbyists, which may not seem like a large advantage at first, but one must realize that this means there is a massive amount of documentation available, nearly any problem that has been encountered, either in regard to the module itself or the firmware that drives it. This serves to be a huge boon to the development of the system and to the development team. Other communication protocols were considered besides Wi-Fi, such as Bluetooth, but ultimately, Wi-Fi prevailed due to the simplicity of its implementation.

C. Database Management System

For our database management system, we chose MySQL. It is the most familiar DBMS to our group, as well as a free one which goes along with our design philosophy to keep cost as low as possible. For our database, we only need to store user credentials, parking locations, and some garage information. The database will be used when the web server receives certain requests to send/update/delete information in the database. The database will contain a counter column for each garage in the garages table that will be incremented and decremented based on the sensors that send signals to our garage kiosk this is what we will use to display real time data to our users that use the web application. For our users, we are only saving their unique id and passcode as well as a salt that is randomly generated when the user is first created and is unique to that user. Our last table is the vehicle_location table which allows users to store and retrieve where they parked at any garage on campus by storing the garage_id, floor_level, and current_numbered_spot with their account_id.

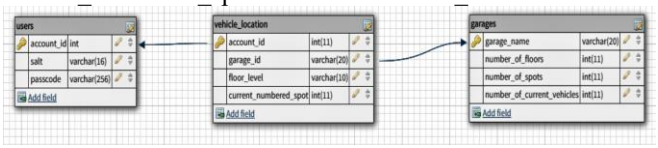

Fig. 5. Database Schema

D. Web Application

Our web application is hosted using NodeJS. The backend web server is written in Node, while the front end uses HTML, CSS and AngularJS. On the initial page, a user will be able to see a table of all the garages with the current number of vehicles in that garage, allowing that user to decide where they want to try and find a parking spot based on the information given by our system. This table is updated by having the ESP8266 send a http post request to our web server which than adds or subtracts from the current amount of vehicles column for that specific garage where the ESP is setup. When the current number of vehicles in the garage equals or exceeds the maximum that garage can hold we change the status to red giving the user a quick visual they can go by to make their decision. Below is a picture of the table on a mobile phone.

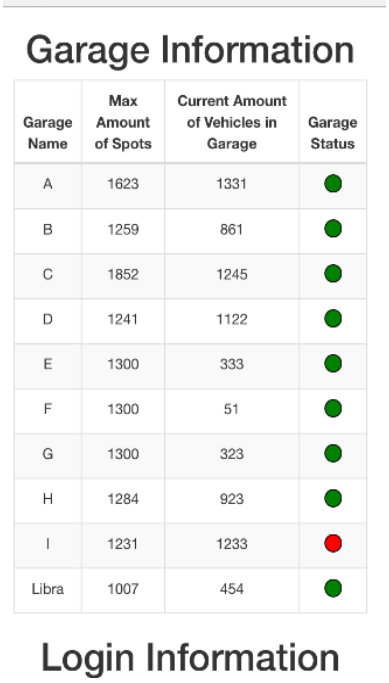

Fig. 6. Mobile Version of garage table

The next feature in our web application is like the save and retrieve vehicle location feature in the kiosk system. Where we allow users to save and retrieve where they parked their car by manually entering their exact location using the garage name floor level and spot number. We also have added a feature to allow a user to take a picture of a QR code and then submit this picture into the system which will than extract the JSON string embedded into the QR code which holds the information of that spot and send it to the web server to be saved. The QR code parser was a 3rd party library that was initially created in C++ but then ported over to JavaScript which worked perfectly for our project allowing the client side to handle all the workload leaving the server to handle other tasks. If our project were to be implemented on campus, QR codes would be placed on every spot in every garage to make it easier for users to store their vehicles information. With the save spot feature we leave it up to the users to save their spots to allow for the garage map to display what spots are available on a floor to floor basis. This gives users more detailed information about each floor of the garage.

The last feature in the web application is the retrieve spot feature, once a user saves their vehicles location they can retrieve that location every time and not have to worry about losing that data. A user can change his vehicles location at any point without consequences happening to the system. The only downfall is that two users can save their vehicles location in the same spot if entered manually, if we forced the user to use the QR code this error could not happen but we decided not to force the user to enter information in a specific way.

When a user visits the website, they have access to these features if they created an account. When saving a spot or retrieving a spot the user must enter their account user ID as well as their 4-digit passcode. Once they have entered their information they can use the save and retrieve spot feature. To create an account a user can visit the create account page, we don't take any information from the user besides their user ID and passcode, this was intentional because we just wanted to show functionality of the overall project. If needed the database table and web page can be adjusted to retrieve more specific information from the user. When they create an account, we save their password by generating a random 16-byte salt string, this gets added to their password and then hashed using a sha256 algorithm, this password then gets stored in the database with the user's ID and salt used to hash the user's password.

For demonstration purposes, we are hosting this website on a Linux virtual machine run on one of our team members home computer. In production, this would be able to run on either a Windows or Linux environment.

E. Printed Circuit Board

The creation and design of the printed circuit board (PCB) was all done through Autodesk's EAGLE CAD scriptable electronic design automation program. This program was chosen for its great online support and widespread use. Due to its extensive free trial and support of all popular operating systems, the program is one of the most used PCB programs for hobbyists and amateur designers. There are only minor flaws within the program but the majority of which are easily overlooked. Such as the minor inconvenience of having to log into a account with the company every time the program is booted up. One of the program's strongest features is its streamlined process of creating new footprints, libraries, and components within the program. Along with the ability for real time synchronization when designing with other's involved. The real-time synchronization feature allows multiple users to create and edit the same design together on different machines over the internet.

The design began at the schematic level. This side the EAGLE CAD lets the user place any type of hardware they need to first form their connections. Each piece of hardware can either be design by the user within the program, or find the correct hardware within their given library or libraries download from other users and design sharing websites. The majority of components needed for schematics can be found created by someone else already on a variety of

filesharing and support websites for printed circuit board design.

Shown in the figure below is the layout of the microcontroller within the entire EAGLE schematic.

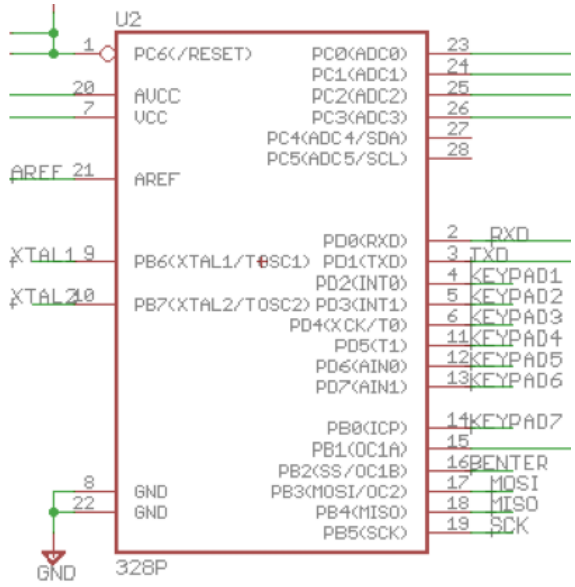

Fig. 7. Microcontroller EAGLE Schematic

This schematic level allows us to layout the general placement of all the hardware components and map their connections accordingly. Each outgoing pin connection is either directly connected or jump connected with a label. As you can see some pin on the microcontroller aren't needed for this design such as pin numbers twenty-seven and twenty-eight for PC4 ADC4 and ADC5. For this microcontroller, it is fine to leave the pins floating and not connected to ground or pull high to keep them inactive without messing with the functionality of the microcontroller.

After the general schematic of the printed circuit board is finished the next step is to create the layout on a circuit PCB. Once the placement of each piece of hardware is complete the next step is to map the connections between all the pins. The layout of the hardware on the PCB largely determines the effectiveness and organization of the wired connections. The final design of the printed circuit board is displayed below.

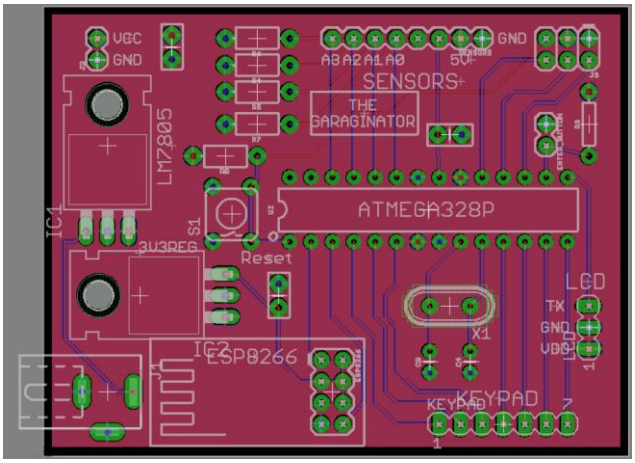

Fig. 8. PCB Schematic

The connections for the hardware ended up being very organized and clean because of the pre-planned layout. The PCB manufacturers allow designers to apply painted on text and figures onto the PCB for easy reference. In the layout above the long rectangular chip in the center labeled ATMEGAA32BP is the microcontroller. Above that is the logo and name for this project "The Garaginator." Above that is the headers for the sensors used at every entrance and exit to the parking garages. Only eight ports are needed. The top-right six pin headers are only used when programming the ATMEGAA32BP microcontroller. The far-right bottom seven pin header will be used to connect the keypad to the MCU and above to the right is the three-pin header connection for the LCD output screen. To the bottom middle of the PCB is the eight-pin connection and layout for the Wi-Fi connection module, to the left of that is the DC barrel jack from supplying power to the entire PCB. The two largest three pin chips above it is the voltage regulators for 5 volts DC and 3.3 volts DC. The rest of the components are typical passive circuit components for power regulation such as resistors and capacitors.

The manufacturer JLC PCB was chosen as the printed circuit board supplier because of their high speed and low cost of production. Ten printed circuit boards from JLC cost only $2.00 before taxes and shipping. The layout of the sight is simple and straight to the point. Below shows a figure of their quote process and options.

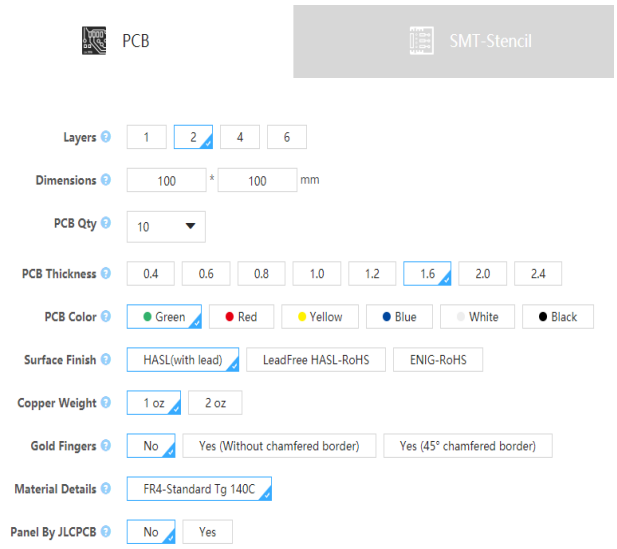

Fig. 9. JLC PCB Quote Options

Most PCB manufacturers give a lot of extra options for every aspect of the PCB design. For this project's application, most of these choices are very unnecessary, such as whether there are bevels on the edges of the board. JLC's website only gives the basics, which are all that are need for this application. The PCB in this design is a two-layer FR4-standard material, about 1.6mm thick, and has a HASL lead finish. As the PCB will be housed within a Kiosk enclosure, there is not worry about needed a lead-free finish on the surface of the board since it will be not handled by the users or somehow find its way into the surrounding environment.

F. Housing

Most of the hardware will be housed in the Kiosk. This includes the printed circuit board, keypad, LCD display, and power supply. The Kiosk and sensor housing will contain a hard-plastic case which has an aluminum bottom sealed by four screws in each corner. The housing allows for great insulation and grounding capabilities. It is also capable of withstanding a wide range of temperature and humidity with little to no deterioration. It isn't necessary for the housing to be water or air tight as it will be placed in the shelter of a concrete garage, hidden from storms and rainfall. The conditions of the surround air will be able to penetrate the housing, and it is expected the heat generated from the running hardware will deter any build-up of condensation within or on the housings.

The housing for the kiosk is only slightly larger than the housing for the sensors. It is five by five inches with a small outlet for the power supply and a window to mount the LCD display. The Sensor's housing is 4-1/8 inches by 2.75 inches with two small holes, one a half a centimeter in diameter as an outlet for the sensor to be in the line of sight of the laser, and the other hole is two millimeters diameter for the input and output voltage wires coming from the kiosk. The figure below shows the housing used for the Photoresistor and laser housings.

Fig. 10. Sensors and Lasers Housing

The following figure shows the housing used for the Kiosk before any drilling modification are made for the power supply and mounting of the LCD display.

Fig. 11. Kiosk Housing

V. Testing

At the beginning of this semester we had already completed individual testing on all hardware components. This gave us time to focus on the integration of all the hardware as well as the software. As well as our PCB design which would bring all the hardware components together.

Since we had to switch our microcontroller in the middle

of senior design II we needed to test the new microcontroller independently by running a few programs and powering it on a breadboard. Once this was completed we could focus on finishing our PCB design and send out an order for it. Throughout the entire semester both the high-level webserver and embedded software were continuously tested during development. We used GitHub as our source code repository and were not allowed to commit into the master branch unless we knew our code could at the least compile. Once we got the code to a working point we hosted the web server on a Linux VM and programmed one of the MCUs we bought with the embedded software. We wanted to test our entire system lifecycle in a controlled environment at first, we did this by testing out our vehicle enter/exit feature first by breaking the two lasers pointing to the photodiodes using a textbook. From here we checked if the database column had been updated for that specific garage, once that was checked we checked if the web application table updated. Once we checked all three of these items we could check off that this feature was complete. Then we went into testing the kiosk user interface, by going through the entire process a normal user would go through, passing in their user information, saving their spot and then retrieving their spot, once we tested this and received no errors we checked off the kiosk UI as complete. This ended our controlled environment testing and now we moved on to a real-world environment. We did this by taking our project over to garage B at UCF and going through the same steps as our controlled environment except we used a vehicle to break the sensors instead of a textbook.

VI. Conclusion and Improvements

The Garaginator takes a very common problem that many students and faculty run into every day which is finding a parking spot and simplifies it for them. It allows users to make an educated decision on where they can focus their time to try and find a parking spot. Instead of going into a garage blindly they can see how many vehicles are already inside vs how many it can hold which gives them a sense of confidence when entering the garage they choose. We added other features such as the saving/retrieving vehicle location to allow users who sometimes forget where they park can easily store that information right after they find their spot using our application. Improvements we would like to have had were being able to track individual spots without needing user input, this would have been ideal to allow users to see garage availability on a floor level view. We also would have liked to include user input on certain features they would like to have seen in this project.

Due to time constraints as well as the scope of the project we decided to not include these items. We wanted to focus on keeping our design simple to the point and cheap to assemble.

VI. Garaginator team

Fig. 12. From left to right: Jonathan Staudt, Sebastian Rodriguez, Elliot Rodriguez, Jonathan Gillis

The Garaginator design team is composed of two electrical engineering students (Sebastian Rodriguez and Jonathan Staudt) and two computer engineering students (Elliot Rodriguez and Jonathan Gillis). Sebastian Rodriguez was primarily responsible for the optical sensor. Jonathan Gillis was responsible for the microcontroller and the firmware load. Elliot Rodriguez developed the entire backend webserver as well as the web client and database. Jonathan Staudt was made responsible for soldering. The PCB design was worked on cooperatively by Sebastian Rodriguez, Jonathan Gillis, and Jonathan Staudt. As John Donne once poetically stated, no man is an island, and nowhere does that saying more aptly fit than when applied to the Garaginator team. Although each member has his primary focus, every member was called upon to help as needed in the disciplines of the other team members; whether it be troubleshooting a piece of hardware or testing software.