Automatic Parking Meter

ECE 445 Design Document

Team 43: Elliot Salvaggio, Kishan Surti, Rutu Patel

TA: Shuai Tang

3/2/2020

1 Introduction

1.1 Problem and Solution Overview

In recent years, we have seen rapid innovation in the car industry. The push for quality electric vehicles and self-driving capabilities for cars has been more and more apparent over the last decade. Despite this progress that has been made in simplifying vehicles, the same has not been made for paid metered parking, an archaic experience everyone with a car must deal with. Overpaying for parking costs Americans over $20 Billion a year [5] and an average of 125,000 people receive a parking citation every single day in the US [6]. We believe this can be attributed largely to the inconveniences associated with parking meters. Although in some cities one can use a simple app to pay a meter, this does not remove the root of the problem. It can be hard to estimate the length of stay one anticipates on being parked at a location, and might end up underpaying, which can lead to a ticket. People also clearly overpay for parking significantly, which adds up over time. It can be stressful knowing you have to leave your location when you're running low on parking time, and difficult to decide how long you believe you will be there for. Additionally, if you're in a rush, one might forget to pay at all. We believe we can solve all these problems with a simple solution.

We propose a solution that is an add-on unit to existing meters in the parking lot. Users would add their name and license plate information to the app associated with our system. Once they park in a parking spot, the unit would detect the car moving into the parking space and take a picture of it's license plate. Our unit will analyze this picture, find the user's license plate number, and using this information, charge them for their stay automatically as the user returns to the spot and drives away. Our solution eliminates the need for someone to worry about whether they've paid for their parking, or have to figure out the length of their stay before they've even gotten out of the car. With our solution, people can be relieved they no longer have to worry about receiving a parking citation.

1.2 Background

We decided to do something about this problem because we have cars and often complain about the issues that come from having to park in paid parking spots on campus. We've been in situations where we forget to pay for parking and get tickets, or have to leave a meeting because we don't want to have to pay for an additional twenty minutes of parking that may not be needed. We believe that with the basic technology we have today that we can come up with a solution to fix this old broken design.

The basic problem is that most parking meters just take coins to pay for parking. We know that on campus the MobileMeter app is the solution to this issue, and it does solve the problem of using coins, but all the issues that come from overpaying and underpaying for parking persist with this simple app [6]. This is why we feel like we can still do better than this existing basic solution. In ideal conditions, someone will simply drive into a parking spot, leave for some time, then return and drive off, and behind the scenes our product will do all the work to keep track of their stay.

1.3 Visual Aid

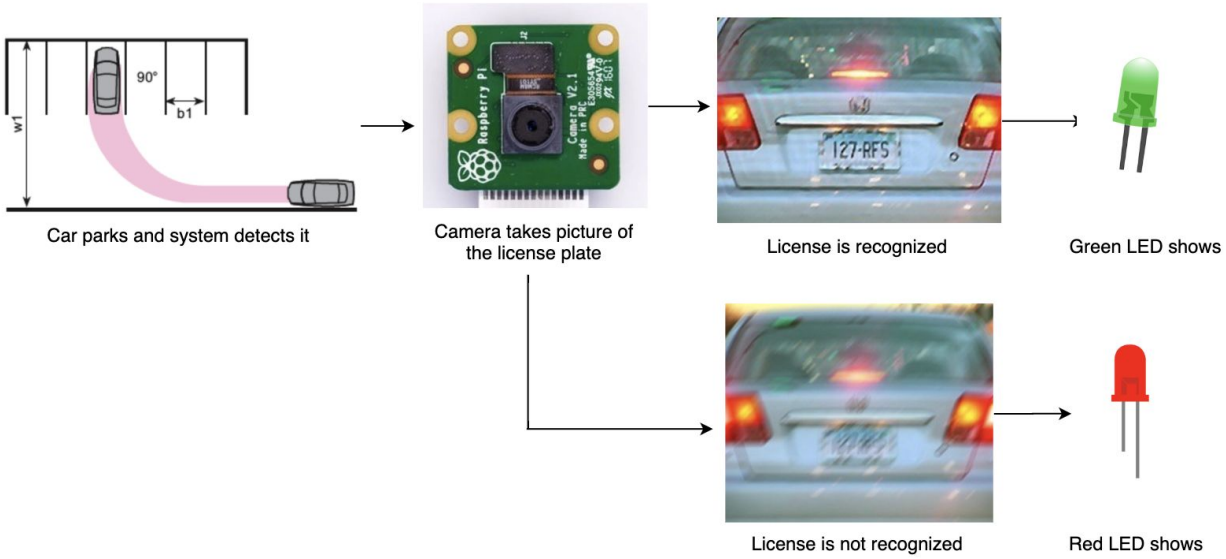

Figure 1: Visual Aid

1.4 High-Level Requirements

- Able to detect a car entering and leaving a parking spot when it is within 5 feet of our system.
- Able to extract a vehicle's license plate, with exactly seven characters of precision, from a picture taken of a parked car's license plate.
- Able to give feedback to the user that our system has recognized the parked car's plate number and registered their stay through a single status LED within 30 seconds.

2 Design

2.1 Block Diagram

There are three main parts to the design of our system. There is the PCB microcontroller system called control unit that contains the ultrasonic sensor and LEDs, the Raspberry Pi system that has the camera attached called OpenALPR module, Power Supply module for required energy to run the overall system, and the Server-App-Database system that communicates with the Raspberry Pi to keep track of user data. *Figure 2* shows the block diagram of the proposed Automatic Parking Meter.

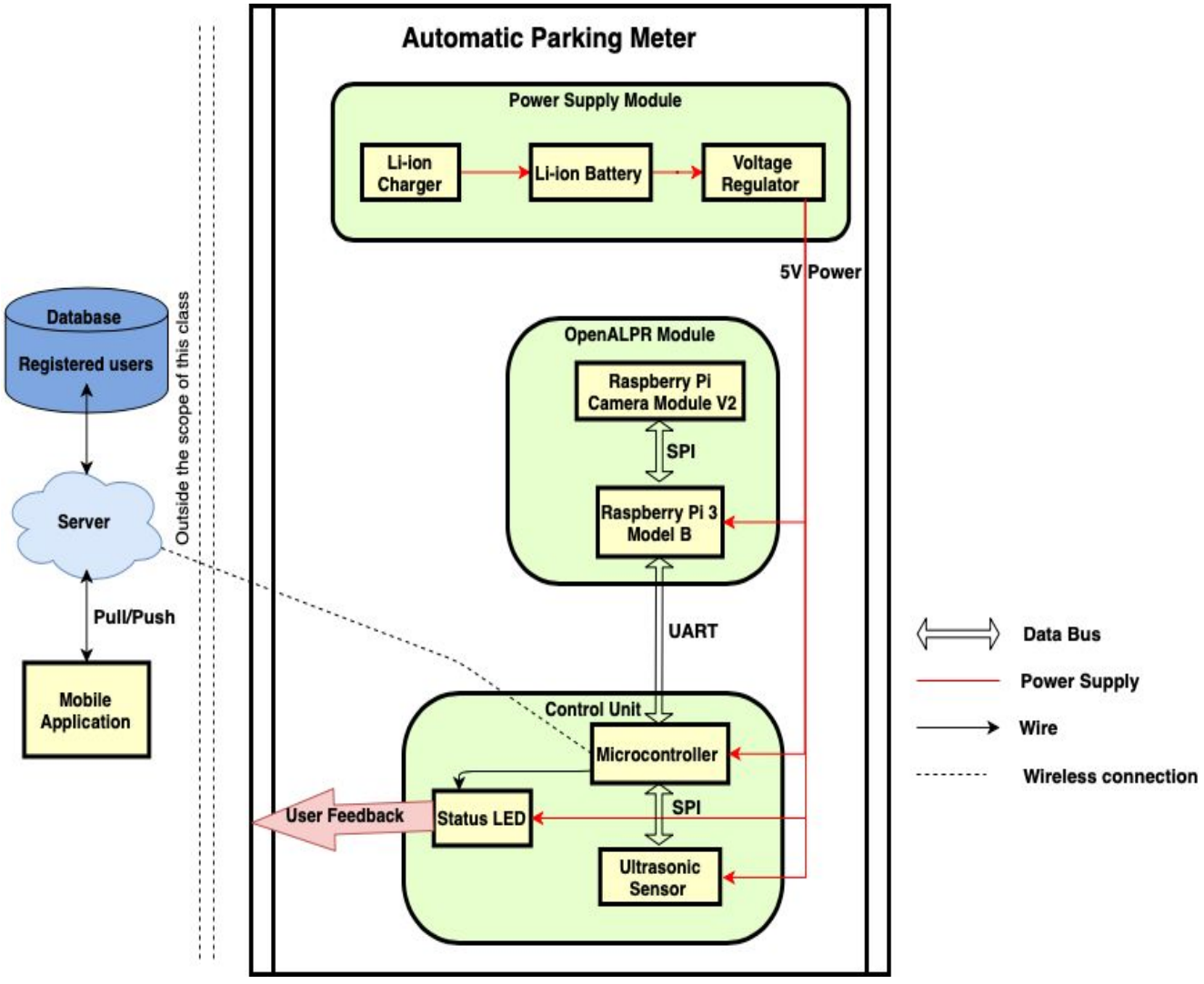

Figure 2: Automatic Parking Meter Block Diagram

2.2 Physical Design

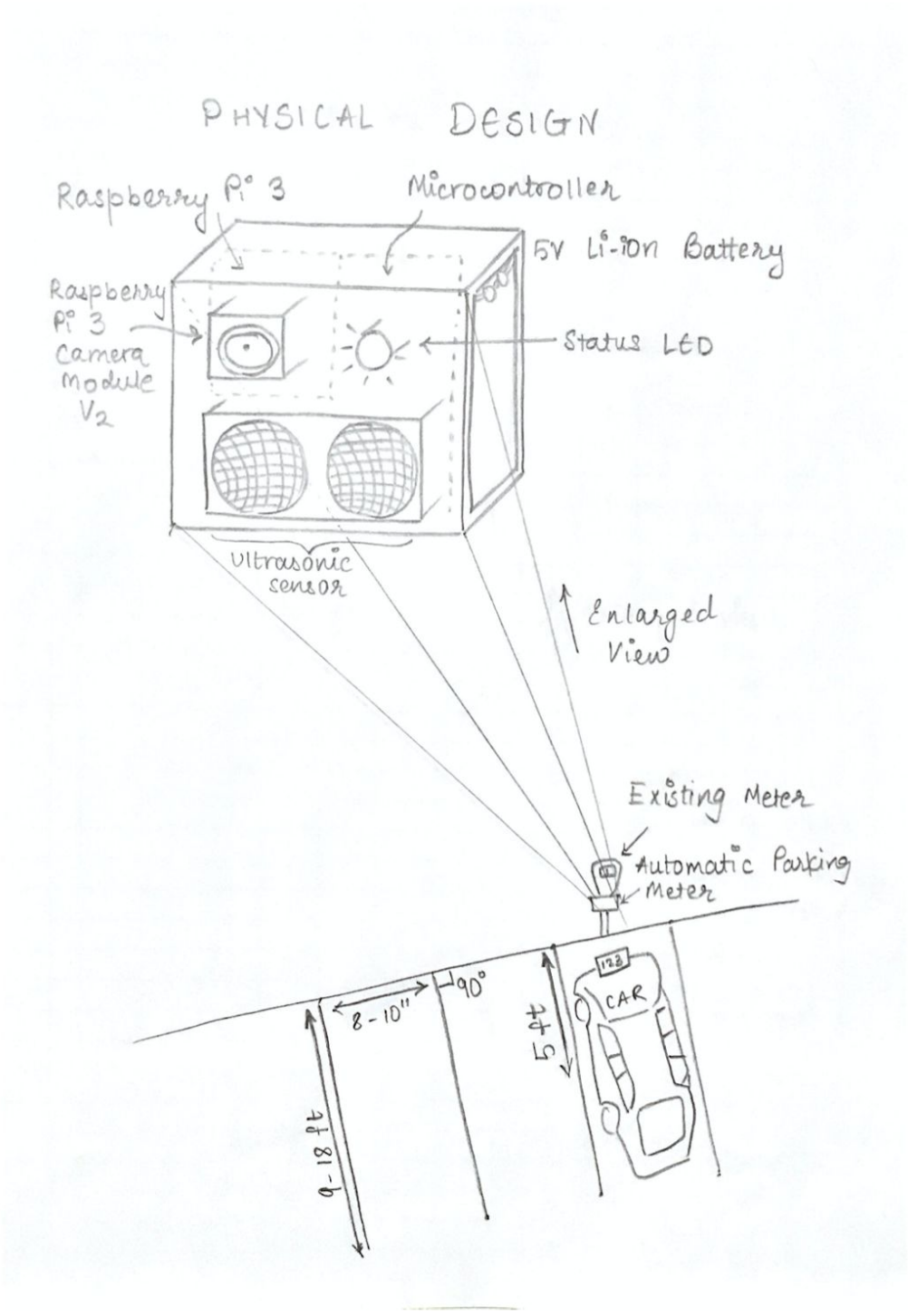

Figure 3: Physical Design

2.3 Power Supply

We need a power supply module which is responsible for providing power to the control unit, and OpenALPR unit. The following are components of this system.

2.3.1 Li-ion Battery: We will have one rechargeable 5V Li-ion battery that will be used to supply power to each component of the project; with power to the multicontroller and Raspberry Pi 3. In particular, we plan to use USB Battery Pack for Raspberry Pi - 10000mAh - 2 * 5V outputs [17]. We will need a 5V power supply for the Raspberry Pi 3 B, and multicontroller.

2.3.2 Voltage Regulator: We will use a voltage regulator to regulate the voltage that is sent to different components such as Raspberry Pi 3, ultrasonic sensor, and microcontroller. Battery provides 5V, however, we have a microcontroller which needs 2.5V - 3.6V, as well as LED that requires 2.6V - 3.8V. Hence, for different voltage input requirements, we need a Voltage Regulator LM1117 to power each component with correct voltage.

Power Supply Subsystem Requirement	Verification
The Li-ion battery must be able to power the system at 4.5-5.5V.	1. Connect a fully charged Li-ion Battery to a constant-current test circuit. 2. Use a voltmeter across the battery to ensure that the voltage is within 4.5V-5.5V range.
Voltage Regulator must be able to provide 2.6V-3.8V for Bi-color LED and 800mA current. Illuminate LED with color red at ~ 2.6 V ± 5%, and color green at ~3.8V ± 5%.	1. Connect a voltage regulator in between the battery and the LED. 2. Use an oscilloscope to measure the open-circuit voltage where we would place the LED. 3. Ensure that the voltage and current gets in the correct range by inserting a

	resistor with the proper resistance value.

2.4 Control Unit

Control Unit is responsible for managing the inputs from sensors, communicating with OpenALPR unit and at the end giving feedback to users via Status LED.

2.4.1 Microcontroller: We plan to use an ESP32 microcontroller. ESP32 has WiFi functionality, which we are particularly interested in for our microcontroller to interact with the database for registered user verification [10]. We believe it should be useful in a project of our size because it is known to be used in the Arduino Uno, which is a similar size to the multicontroller we plan to build. Microcontroller will be responsible for taking inputs from the ultrasonic sensor, trigger the OpenALPR unit; specifically Raspberry Pi 3 Model B, to take camera inputs, and wait for the OpenALPR unit output through SPI (Serial Peripheral Interface). Microcontroller will then output the user feedback with Status LED.

2.4.2 Ultrasonic Sensor: For the purpose of triggering the system, we need to detect the distance between our system and the car entering or leaving the parking spot, for which we will use an Ultrasonic sensor [4], since they are more reliable than IR sensors during day time due to the influence of sunlight. The Ultrasonic sensor will measure the distance of the car to our system, which would be set for 5 feet, thereby communicating with the microcontroller to initiate the camera input and begin processing.

2.4.3 Status LED: For the purpose of displaying to the user if our system recognizes the car parked or not, we will use a bi-color standard Green and Red LED, that consumes 20mA current and 2.6-3.8 V Voltage [4]. The input to the LED is from the microcontroller regarding the OpenALPR unit.

Control Unit Subsystem Requirement	Verification
• Must be able to take ultrasonic sensor input when any car is within 5 feets.	1. Program the microcontroller with Arduino code putting ultrasonic threshold range at about ~5 feets ± 5%. 2. Connect the microcontroller to the laptop using USB, and have the code check if the sensor detects an object within 5 feets ± 5%. 3. Take any object of about size 5*5 feet and bring it close to the ultrasonic sensor in the range of 5 feets ± 5%. 4. Confirm by printing some output if the sensor was able to detect the object successfully.
• Must be able to communicate with the Raspberry Pi Model 3 at speed greater than 1.5 Mbps.	1. Connect microcontroller to the Raspberry Pi Model 3 using USB. 2. Send random data from the microcontroller to the Pi using the USB bridge and start recording time. 3. Ensure that the data received on the Pi matches the data that was sent from the microcontroller within range of speed.
• Must be able to communicate with the database from the WiFi module in less than 20 seconds.	1. Send any dummy data input to the server, via WiFi module to store into the database.

	2. Check the data in the database to see if the data write occurred. 3. Program the microcontroller about parsing the database for the desired data and ensure that the dummy data is verified in the database as expected in the time range.
● Status LED must be properly illuminated when instructed from the microcontroller when biased with specified voltages in range of 2.6 - 3.8V.	1. Repeat the verification steps from above. 2. If verified, program microcontroller to provide specific voltage across LED either green or red, based on color voltage specification. 3. Using a voltmeter ensure that the correct LED color illuminates within 2.6 - 3.8V.

2.5 OpenALPR Module

OpenALPR (Automatic License Plate Recognition) module is considered as the main brain of our system as we use OpenALPR open source image recognition library for extracting license number from license plate image. This module communicates with the microcontroller for its input and output. When the ultrasonic sensor on the multicontroller finds there is a car moving into the spot within 5 feet of the system, we will send a signal to a Raspberry Pi 3 Model B via microcontroller for the Pi to trigger its camera module and start computing the license plate number.

2.5.1 Raspberry Pi 3 Model B: We will use a Raspberry Pi 3 to run computer vision algorithms on the picture captured by its attached camera. The attached Raspberry Pi Camera Module V2

will be triggered to take a picture by input from the control unit i.e microcontroller. Once the picture is stored on the RP3, we will run OpenALPR (Automatic License Plate Recognition) algorithms on the image to extract the car's license numbers. We will use the built in WiFi module of the Raspberry Pi 3 to send the extracted plate number to a database/server, where it will check if a user with such license number has registered through the mobile app. The Raspberry Pi would further communicate the information received from the database to the microcontroller.

2.5.2 Raspberry Pi Camera Module V2: We will attach a 8 MP Raspberry Pi Camera Module V2 to our Raspberry Pi 3 B that will take a picture of the car's license plate when it is notified by the microcontroller in the control unit. The camera will be attached on the front side of our system .

OpenALPR Module Subsystem Requirement	Verification
● Must be able to take microcontroller input to trigger the Raspberry Pi Camera Module V2.	1. Connect the Raspberry Pi 3 along with the Camera Module V2 to any laptop. 2. Confirm that Raspberry Pi 3 triggers the Camera Module V2, after keypress. [13] 3. Confirm the camera takes a picture in any picture formats stored on the laptop.
● Camera Module V2 must be able to take a clear picture when notified by the Raspberry Pi 3 that is good enough quality of an image that it can be properly analyzed by the OpenALPR system.	1. Connect the Raspberry Pi 3 along with the Camera Module V2 to any laptop using a USB cable. 2. Run the Raspberry Pi script for taking a picture via camera module.

	3. Confirm the camera takes a picture of good quality stored on the laptop.
● Must be able to perform OpenALPR accurately extracting the license number from the license image, precision of character by character.	1. Visually confirm the output from the OpenALPR algorithm returns the same number as you can visually see on the license plate.

2.6 Mobile Application: We will utilize a mobile application associated with our parking meter. This will give a more detailed feedback to the user, and they will be able to see the duration they have been parked in a spot, how much they would get charged accordingly, other car information details, and also the user's QR code that can be used for scanning purposes. The QR code will provide as an alternative if the Camera Module cannot detect the vehicle's license plate for some reason, and the user will be able to scan this to secure their parking spot and begin their time parked. In addition, we will have the user register their information within the application prior to parking, this is necessary for license plate recognition to be associated with an existing account.

Mobile Application Requirement	Verification
● Users can register their information in the application and can be updated in the database.	1. Using the log-in page of the application, create a new user and refresh the database immediately after. 2. Check if the database has now changed to reflect the addition of a new user.

2.7 Server/Database: In order to have users register their information within our mobile application, we must have some sort of database that can retain this information, and can be

easily accessible with our app. We have chosen to use Firebase for our backend, this will integrate nicely and we can access it quickly and efficiently.

Server/Database Requirement	Verification
● We are able to retain information in the database, and make efficient access within one second to it when checking if an account exists under a license plate.	1. Program the code to perform a read and write to the database 2. Measure the latency of both the read and write using a network capture and ensure that it is under one second.
● We are able to create new users in the database from the mobile application.	1. Parse the database to see if its been updated with new user data when registered from the app.

2.8 Schematics

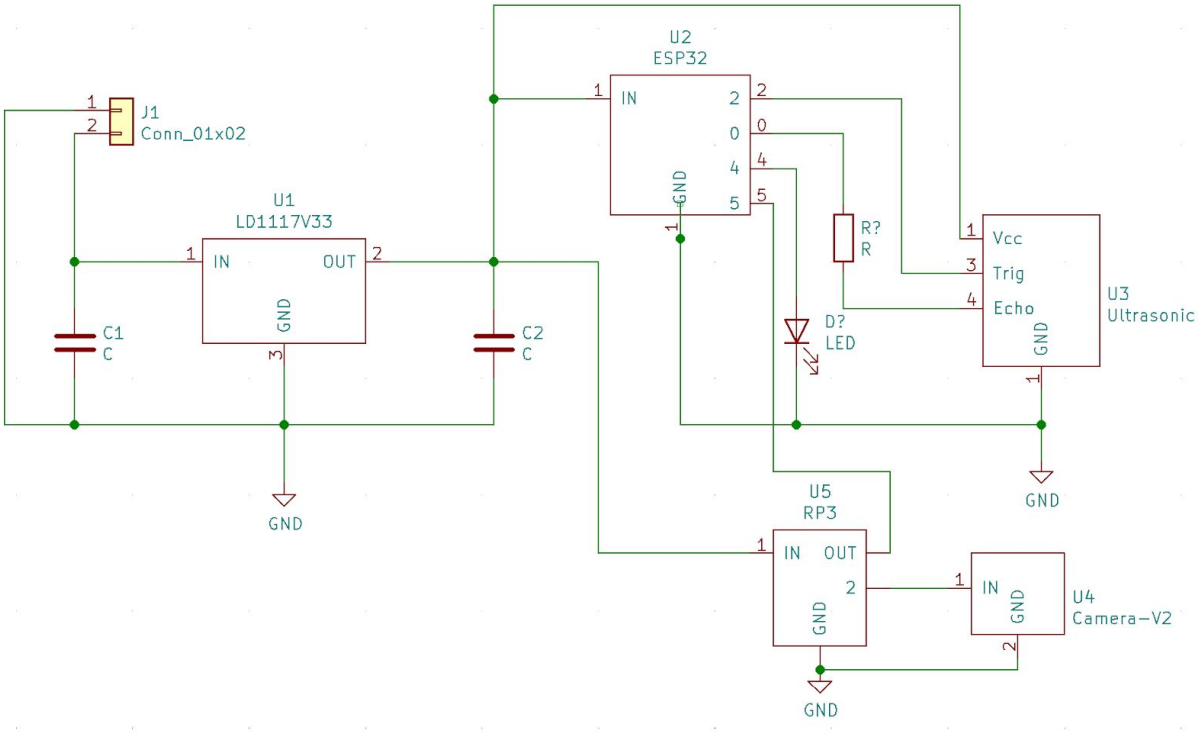

Figure 4: Circuit Diagram

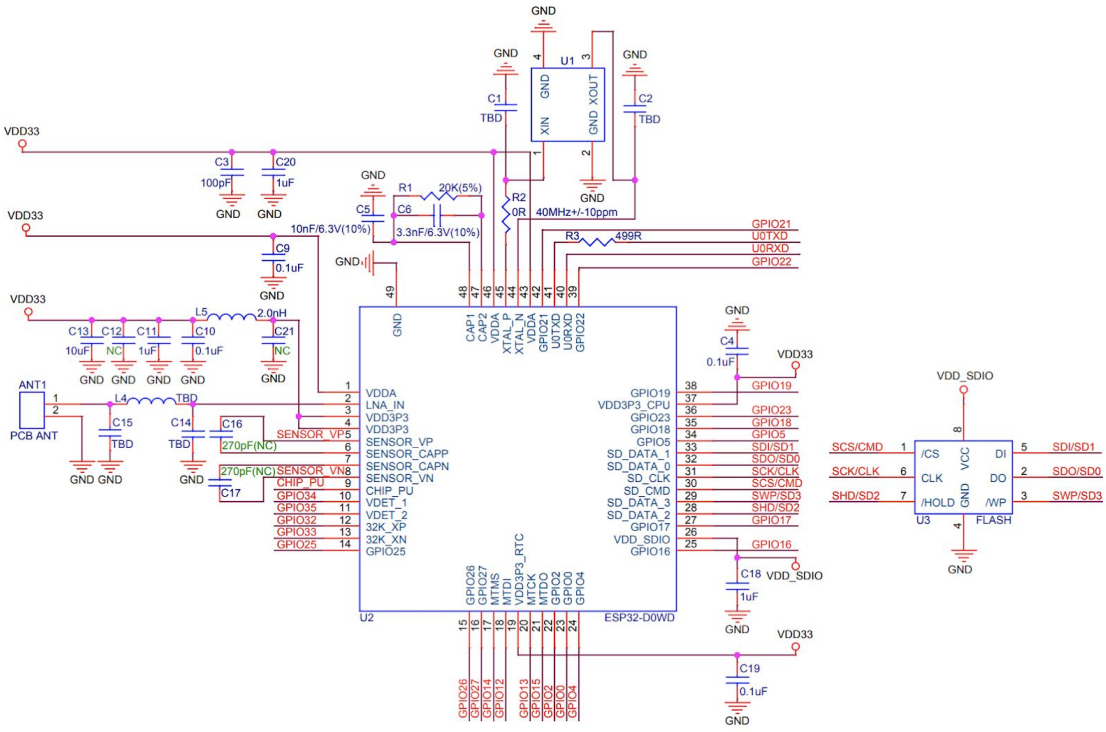

Figure 5: Detailed ESP32-WROOM-32D schematic

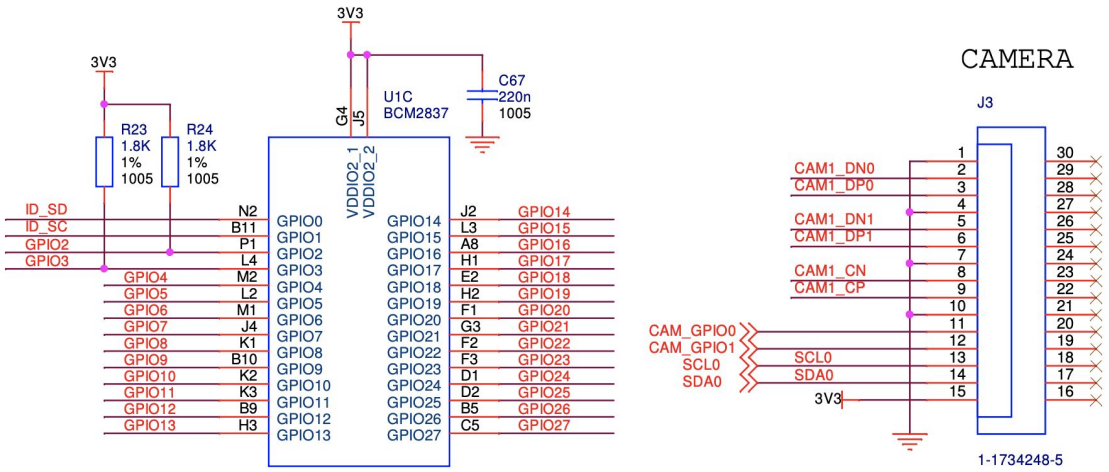

Figure 6: Detailed Raspberry Pi 3 B and Raspberry Pi Camera Module V2

2.9 Tolerance Analysis

A critical component of our project that will pose the biggest challenge is our Control Unit block. There are several requirements that this block must pass in order for our design to be successful: 1) The microcontroller needs to be supplied with approximately 3.3V. 2) The ultrasonic sensor must be able to detect a large object at a distance of 5 feet away. 3) The LED will have to display red or green depending on the status of the plate registration.

In order to fulfill our first requirement, we will need to use some sort of level shifter because our power supply input will be about 5V. There are two ways to approach this solution: a voltage divider or a linear voltage regulator. A voltage divider is a simple circuit that aids in reducing a voltage source by adding resistors. The formula for calculating this is below [8]:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

Here, if we substitute 3.3V for V_{out}, 5V for V_{in}, and 1kΩ resistor for R_1, we get that our second resistor will be about twice the value of R_1. The circuit would then look like this [8]:

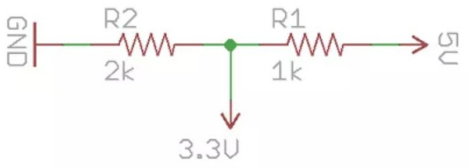

Figure 7: Circuit diagram of the voltage divider

However, this method is not quite reliable, and typically works for slow signals. With this in mind, we have decided to take the approach of the linear voltage regulator.

The linear voltage level converter is more ideal for signals with a higher baud rate, which will work in our favor. The schematic for this converter is shown below [18]. This regulator requires the use of two 10µF capacitors for the input and output for stability. It has up to 800 mA capability, and our microcontroller requires at least 500 mA so it will be able to power it well. There are heat-sinking factors that we need to be aware of, but as long as we supply a steady 5V, our dissipated heat is only 0.85 W, so no heat-sink would be required.

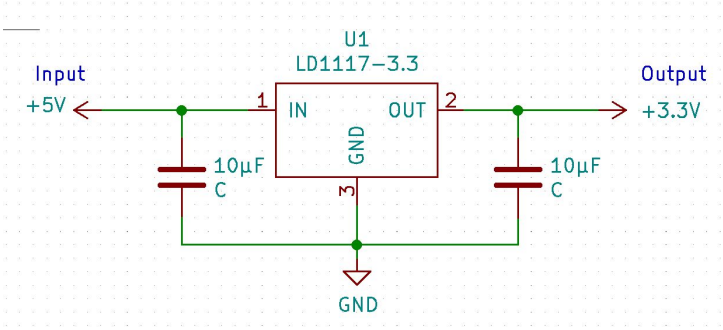

Figure 8: Circuit diagram for linear voltage regulator

Our second requirement is that the ultrasonic sensor should be able to detect a large object at a range of 5 feet. A problem that can arise is when a person may be walking in front of the parking meter and accidentally triggers it. To accomplish our requirement and combat this issue, we will program the ultrasonic sensor to only detect an object within 5 to 7 feet for several consistent cycles. During some testing, we found that this was an optimal distance to check for a vehicle. This will pick up the car when it is entering the spot and will then signal to our Raspberry Pi module to capture an image of the license plate. The sensor's maximum limit for

distance checking is roughly 4 meters, with the ranging accuracy reaching to about ±3 mm. Below is the timing diagram for the HC-SR04 Ultrasonic Sensor [27]:

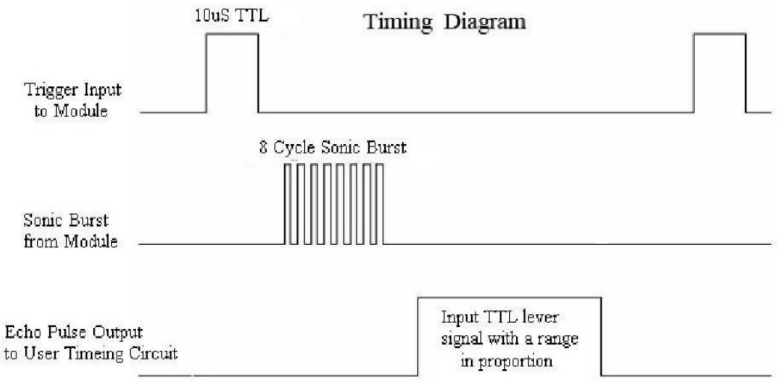

Figure 9: Timing Diagram for Ultrasonic Sensor

In order to begin range detection, we will first need to send a 10μS pulse. This will then send a cycle of 8 bursts of ultrasound at 40 kHz, which raises the echo signal. This echo signal will return towards the sensor itself, letting it know whether it detected an object.

The last requirement that must be satisfied in our Control Unit block is that the LED must display either green or red, while operating at a voltage of 2.1 V for green and 2.0 V for red. It requires at least 20 mA of forward current to be operational, as shown in the below diagrams [28]. This can be accomplished easily, as our input power supply will be 5V, so we need to add the proper resistor value before the LED in order to obtain that ideal voltage needed.

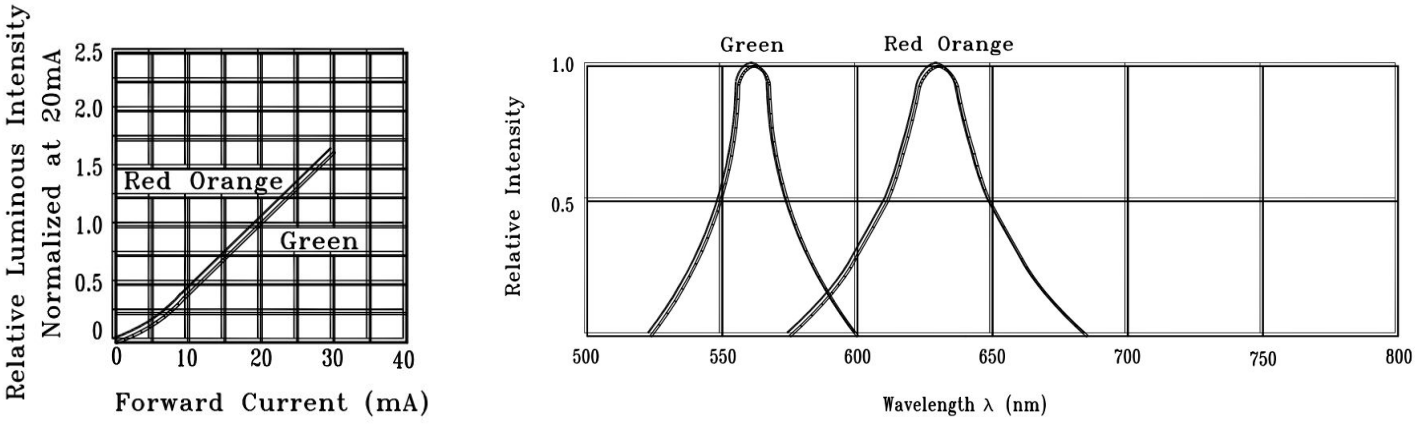

Figure 10: LED color relative to Current

3 Cost and Schedule

3.1 Cost Analysis

3.1.1 Labor Cost

We are a team of three, all Computer Engineering students. The average starting salary for an Illinois Undergraduate Computer Engineering graduate was $96,518 in 2016. This comes out $96,518/52 weeks = $1,856.11 per week. $1,856.11/40 hours = $46.40/hour. There are about 10 week left of the semester. If we all plan to work 15 hours a week on the project, that puts the total labor cost to (46.40*15*8*3*2.5) = $41,760.

3.1.2 Parts Cost

Part Name	Qty.	Part Description	Cost
Raspberry Pi 3 Model B	1	Computer Vision processing device	$35.00
Raspberry Pi Camera Module V2	1	Take license plate picture	$28.20
Espressif ESP32-WROOM 32D	1	Microcontroller	$4.50
Digikey 3.3 V 800 mA Regulator - LD1117-3.3	1	Brings voltage from 5V power source down to 3.3V for ESP32	$0.55
Digikey Ultrasonic Sensor - HC-SR04	1	Sense an approaching car	$3.95
Digikey Bi-Color LED	1	Depending on the input voltage displays Red or Green LED	$0.55
Adafruit USB Battery Pack for Raspberry Pi - 10000mAh - 2 x 5V outputs	1	To power overall system	$39.95

Total: $112.07

3.2 Schedule

Week	Elliot	Kishan	Rutu	Team Goal
2/24	Work on Design Document	Work on Design Document	Initiate PCB design in KiCad from planned circuit schematic/Work on design document	Complete Design Document
3/2	Buy project parts	Create test cases and check circuit schematic prone of bugs	Make initial conversation with ECE shop to put our system in box	Create PCB design and correct schematics if needed
3/9	Program microcontroller	Test design	Finalize PCB for Early bird PCB order	Finalize PCB design, order parts and send in design
3/16	Work on WiFi module of the microcontroller to send query request to database	Implement database for registered user	Implement openALPR algorithm to work with Pi	Start working on software applications (microcontroller and mobile application backend)
3/23	Continue working on data transmission protocols	Continue working on data transmission protocols	Continue working on data transmission protocols	Continue working on software applications
3/30	Debug control component	Implement the Mobile Application	Create front end design of Mobile Application	Connect software applications to PCB design (if we have received PCB by now)
4/6	Debug control component	Continue with mobile application	Test end to end flow of the system	Begin debugging overall design, creating test cases
4/13	Create ways to test overall design	Test mobile app	Fix any bugs or errors and finalize product	Finalize and fix bugs/flaw in design, continue

				testing
4/20	Prepare for demo	Prepare for demo	Prepare for demo	Final touches on design, prepare for demo, start up final report for project
4/27	Work on final paper	Work on final paper	Work on final paper	Work on and finish up final report for project

4 Discussion of Ethics and Safety

One of the few concerns regarding our project will be the power supply. The parking meter should theoretically be placed outside and should be able to endure various weather conditions, including rain. We must ensure that our power supply is covered properly so that rain or snow cannot enter. This could potentially cause a hazard to the user if they are touching the parking meter and it has been exposed to water leakage in some manner. This applies to the IEEE Code of Ethics #1 [1]. We must make sure that our finished design complies with an IP65 rating, which means that it will be "dust tight" and protects against water that is shot from a nozzle.

Another ethical issue that can arise is keeping a user's personal information safe and secure. We will only ask for their license plate, phone number, and full name. We will not disclose any of this information, and our system will not retain any of this information for our benefit, nor will we keep track of a user's location. We will only display accurate information on our parking meter and within the mobile application. This aligns with the IEEE Code of Ethics #3 [1]. We want to ensure that a user who is using our application can trust us with their data.

5 Citations:

[1] IEEE, “IEEE Code of Ethics,” *IEEE Policies*, Section 7, no. 8, June 2019. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed Mar. 1, 2020].

[2] Openalpr, “Camera Configuration.” Camera Configuration - Openalpr 2.7.102 Documentation. [Online]. Available: doc.openalpr.com/camera_placement.html. [Accessed Mar. 1, 2020].

[3] Atascadero Municipal Code, 9-4.117 Parking Design Standards. [Online], Available: qcode.us/codes/atascadero/view.php?topic=9-4-9_4_117. [Accessed Mar. 1, 2020].

[4] Pendergast, Robert L., et al. “Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino,” Random Nerd Tutorials, 2 Apr. 2019. [Online]. Available: randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/. [Accessed Mar. 1, 2020].

[5] Inrix, “Searching for Parking Costs Americans $73 Billion a Year.” PR Newswire: Press Release Distribution, Targeting, Monitoring and Marketing, 26 June 2018, [Online]. Available: www.prnewswire.com/news-releases/searching-for-parking-costs-americans-73-billion-a-year-300486543.html. [Accessed Mar. 1, 2020].

[6] Davis Law Firm, “Traffic Ticket Statistics.” Davis Law Firm, 1 Oct. 2019, [Online]. Available: jeffdavislawfirm.com/traffic-ticket-statistics/. [Accessed Mar. 1, 2020].

[7] Arduino. “ArduinoToBreadboard.” *Arduino*, [Online]. Available: www.arduino.cc/en/Tutorial/ArduinoToBreadboard. [Accessed Mar. 1, 2020].

[8] Connolly, David. “How to Level Shift 5V to 3.3V.” *Random Nerd Tutorials*, 2 Apr. 2019. [Online]. Available: randomnerdtutorials.com/how-to-level-shift-5v-to-3-3v/. [Accessed Mar. 1, 2020].

[9] Fuchs. “Ultrasonic Sensors Knowledge (Part 4): Influences on Measurement Accuracy.” *Pepperl+Fuchs*, 20 July 2018. [Online]. Available: www.pepperl-fuchs.com/usa/en/25518.htm. [Accessed Mar. 1, 2020]

[10] Explore Embedded. “Overview of ESP32 Features. What Do They Practically Mean?” *Tutorials*, 17 December 2016. [Online]. Available: www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F. [Accessed Mar. 1, 2020]