# Automatic Parking Monitoring and Assistance for city of Champaign

By

Bo Wang

Christopher Santoso

Ximin Lin

Final Report for ECE 445, Senior Design, Spring 2019

TA: John Kan

May 2019

Project No. 17

## Abstract

Parking meters are amongst some of the most antique and overlooked technology today. These systems are rarely updated, and when they are, they’ve proven to frequently obfuscate the parking process with unclear messages and strict fees. Our parking meter seeks to add simplicity to the process by removing as much worry on the driver as possible. With the ability to automatically recognize a driver’s license plate and charge their associated balance accordingly, we can easily eliminate the need to carry around spare change or a credit card. Furthermore, strict fees often discourages drivers from parking in certain spots, especially when they are ruthlessly enforced by previous failed attempts at automated parking. We seek to solve this by integrating more lenient parking rules on our devices, by only charging a driver once they have been parked for a set amount of time, and by allowing a time gap between when their balance runs out and when parking officials should be notified. With these considerations in mind, our parking meter will automatically detect license plates with high accuracy, and remove the hassle of the driver having to monitor their parking time themselves.

## Contents

## 1. Introduction

When people forget to pay the parking meter due to carelessness or some emergencies, people could be charged for $15-50. Also, sometimes because of the bad schedule plannings, people ran late to the parking lot to pick up their car. In case of these scenarios, people usually pay extra money to safely cover the time they will use, which in turns cause people losing money because they left early.

Also, parking spots on campus are numerous, and there is constantly a need for multiple people to constantly monitor parking spots to ensure rules are not broken. It is very easy for an individual to make an honest mistake however, such as parking too close to the curb, forgetting the time limit they paid for, or parking in a spot they are not allowed to park in.

We propose to solve those problem with an enhanced parking device that can be either mounted on a pole or a wall that will monitor cars coming in and out of the parking space, and notify the car-owner and the officials if there is a violation. Our device will also be able to calculate the parking fee depends on the exact time for a driver parks and charge his or her prepaid account based on the plate number, which would be a more reasonable method compared with current inflexible one.

## 2. Design

### 2.1 Design Procedure

#### 2.1.1 General Design Considerations

With consideration to the flow below, there are four jobs that must be accomplished: incoming car detection, plate recognition, parked-time counting, and WiFi communications. With respect to these requirements, we decided to include a Raspberry Pi in our overall design for its WiFi communication capabilities and its speed in terms of processing power over the standard microcontroller, since plate recognition and communication would have to happen in a very short amount of time (during a car’s arrival and departure), there was a necessity for sufficient processing power in addition to our standard microcontroller.

Another one of our goals was to preserve power as much as possible. While the entire workflow is possible with the Raspberry Pi alone, there exist concerns over its lifetime if it were to run 24/7; plus, it would consume considerably more power in doing so (calculations in 3.1 Power Verifications section). Thus, we chose to have the Raspberry Pi sleep during periods when it was not needed, and allow the microcontroller to wake the Pi instead. While the Pi is asleep, the microcontroller will then perform the simpler tasks such as calculating the balance while a car is parked or awaiting a parked car while there isn’t one.

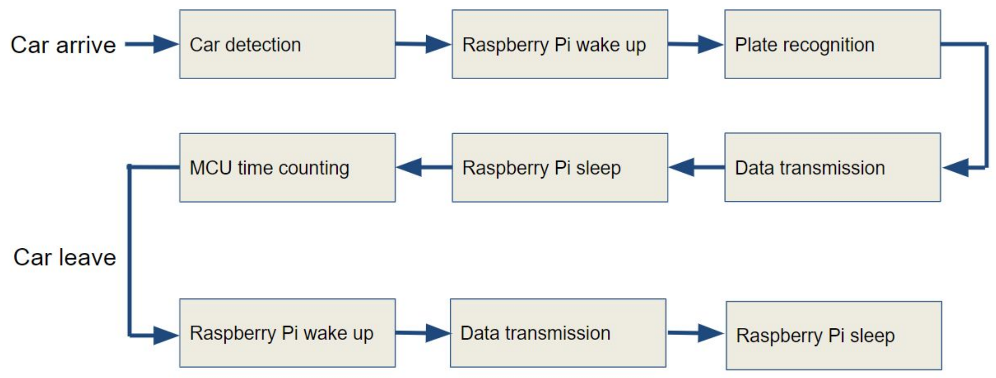


*Figure 1: The designed working flow of our parking assistance device, shown as one round of a car park in and leave.*

### 2.1.2 Hardware design

Based on the flow shown in Figure 1, our front-facing interface requires three main parts: an LCD screen to display parking-related messages, an ultrasonic sensor to detect motion, and a camera for license plate recognition. We decided to allow a Raspberry Pi (B+) handle the WiFi communications with the main server and the plate recognition because these processes can potentially consume a lot of processing power, we chose to delegate these tasks to a device that has proven itself in these areas.

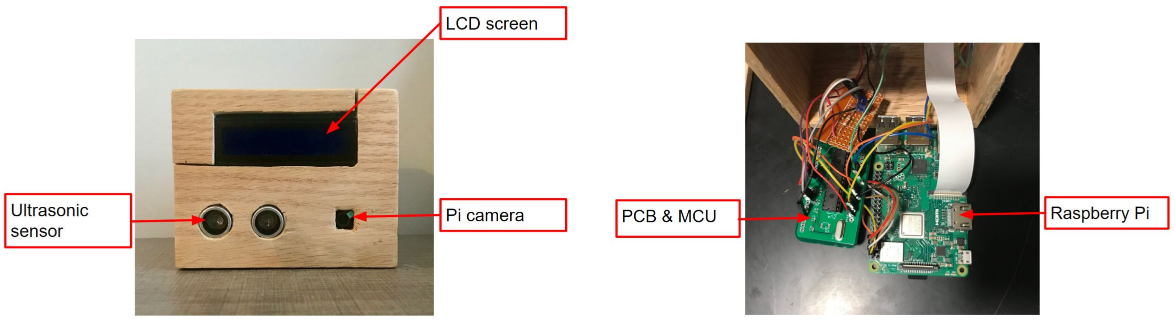


*Figure 2: The final combination of our hardware components. Ultrasonic sensor for detection, camera to take a photo including the car plate, LCD for information displaying. Inside the box is our PCB with microcontroller and Raspberry Pi.*

Since the Raspberry Pi is only needed when there is an incoming car and a departing car, the vast majority of the time will be spent either awaiting a new car, or deducting from the balance of a currently parked car. These simple tasks can be left to the microcontroller. In the case of awaiting an incoming car, the motion sensor need only be polled for a (decreasing) change in distance, and in the case of calculating the balance of a parked car, the balance need only be deducted by a set rate every couple minutes; thus, during this time, the Pi is put to sleep.

### 2.1.3 Software design

To recognize the plate number on the car, we started working on the recognition algorithm. The image would be taken by Raspberry Pi camera and then processed by Raspberry Pi. The next part of our software design are the communication between microcontroller and Raspberry Pi, data transmission between Raspberry Pi and central server, which store the information of users accounts.

## 2.2 Design Details

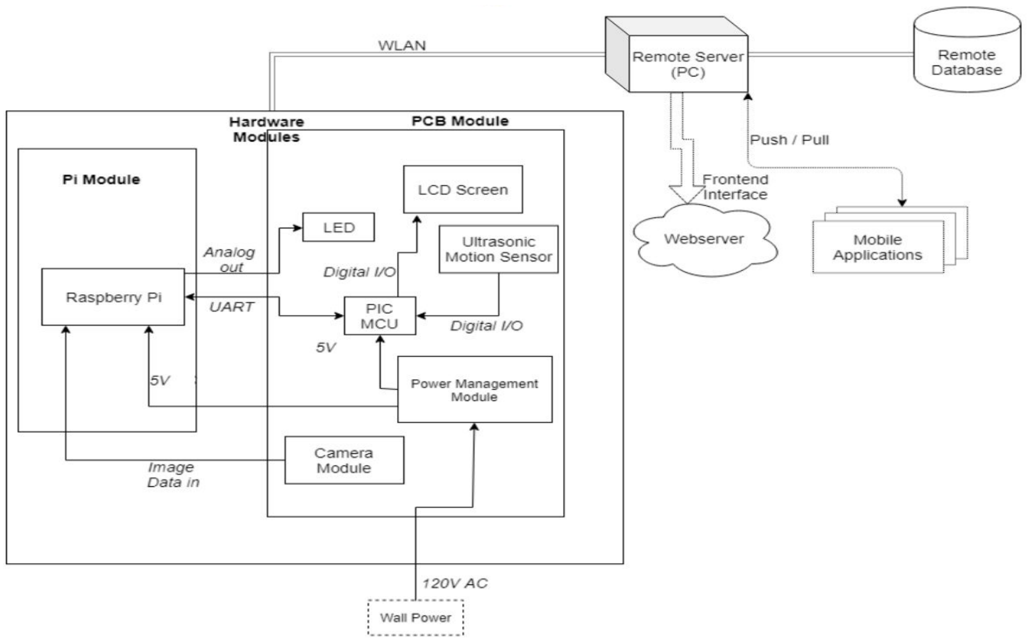


*Figure 3: Top-Level block diagram of the entire system. The hardware section is separated into a PCB module, which will consist of the modules to be placed on the PCB, as well as the Pi Module, which consists of only the Raspberry Pi.*

### 2.2.1 PCB & LCD screen and Ultrasonic sensor

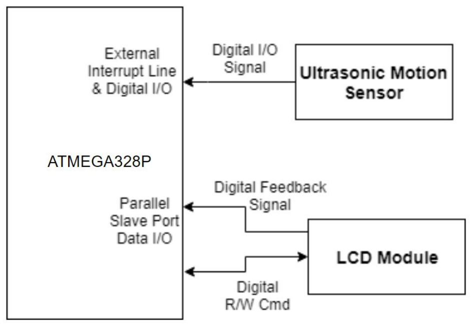


*Figure 4: PIC Microcontroller to LCD Module and Motion Sensor Module connections.*

To detect whether a car is parking, we are using the ultrasonic sensor. The basic principle of ultrasonic sensor is that one hole of the sensor send the sound wave, and then the other one receive the wave reflection. The distance is calculated by:

$$\text{distance} = \text{time interval of sending and receiving} \ * \ \text{speed of sound} / 2 \qquad (1)$$

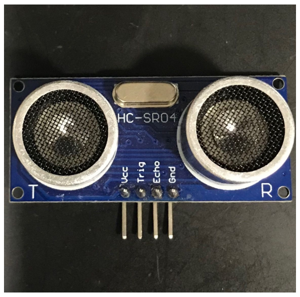


*Figure 5: The Ultrasonic sensor. pin Trig is the input from MCU and Echo is the output to the MCU. Those two pins connect and communicate with microcontroller based on the formula described above.*

### 2.2.2 Microcontroller to Raspberry Pi Module

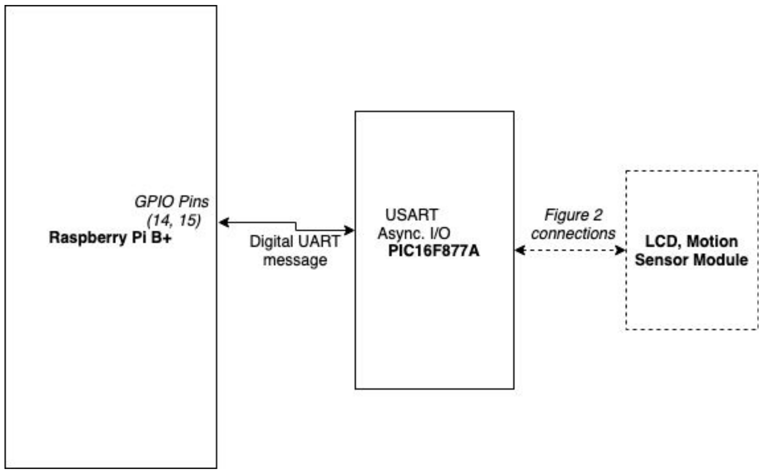


*Figure 6. Raspberry Pi to Microcontroller connection. The dotted module is the module depicted in Figure 2 (connections between the microcontroller, the LCD screen, and the ultrasonic motion sensor).*

### 2.2.3 Raspberry Control & Power distribution and saving

Our device use the ground power. For Raspberry Pi, we used the power adapter of Raspberry Pi. All other components, including microcontroller are powered by 5V pin on Raspberry Pi.

Another significant feature of our design is Raspberry wake up & halt control. Raspberry would sleep twice and wake up twice in one round of parking, our goal of this module is to save the power as much as possible.

At first, what we trying to do is using transistor to modify the connection between 5V dc power and raspberry pi micro usb power input, simulating the plug in and plug out operation, but we find out that any resistor or the resistance of transistor would affect the voltage applied on Raspberry pi since the working resistance of Raspberry pi is only at 5 ohms to 10 ohms.

But then we find another way to achieve the similar feature: We keep Raspberry PI connected with 5v dc power. If we code a command in our program,The raspberry would stay in halt mode,  and if we wanna wake it up,  we have to short short pin 5 and 6, which is GPIO 3 and GND. We complete this control module still by the feature of transistor(BJT), a digital signal would be sent from MCU, the other two pin would act like connected.

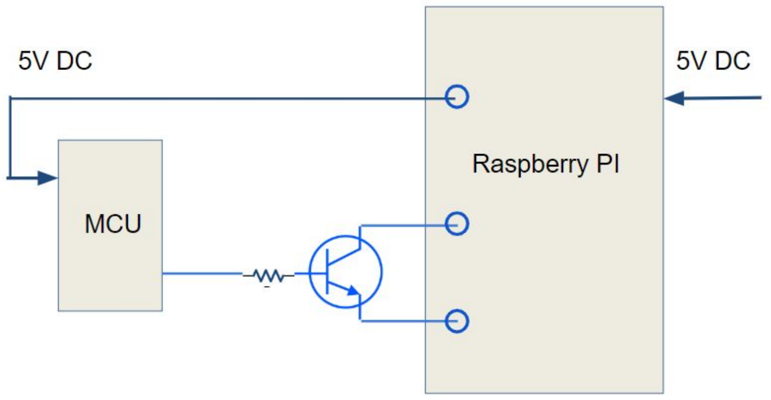


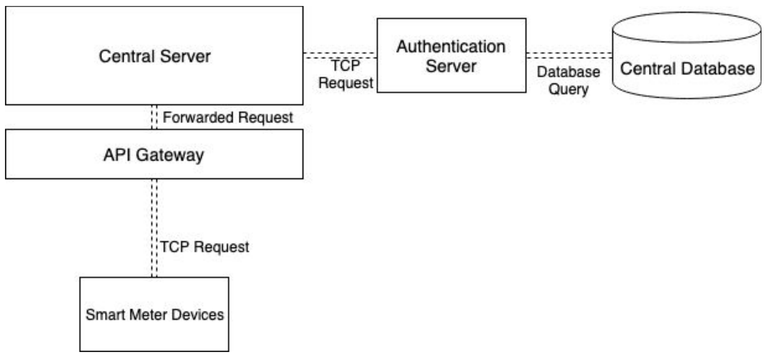
*Figure 7. Power supply and the virtualization of Raspberry Pi power control module.*

### 2.2.5 Plate Recognition Model

The plate recognition model will consist of multiple models. The first model consists of detecting objects with the shape of a license plate, which means this model will be trained to recognize a rectangle of a specific size, edge, and of corner shape (since license plates have distinctly rounded corners). The second portion of the model consists of optical character recognition. Once a license plate-shaped object is detected, there needs to be a model to recognize text on the license plate itself, and associate this text with ASCII characters in order to perform lookups based on the license plate.

We implement the major parts ourselves, including plate segmentation and character segmentation. For the remaining character recognition, we use Tesseract, developed by Google, to recognize the characters. Tesseract is actually powerful to directly segment characters and recognize them. However, after several trials, we found that the results of Tesseract's recognition are inconsistent and prone to noise. That's why we only use Tesseract model for single character recognition. With the help from [5], we are able to do plate segmentation and character segmentation effectively.

### 2.2.6 Software Flowcharts



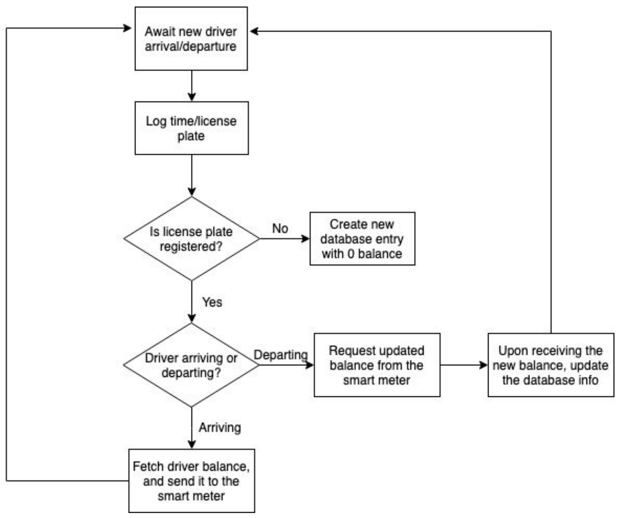

Await new driver arrival/departure
Log time/license plate
Is license plate registered?
No
Create new database entry with 0 balance
Yes
Driver arriving or departing?
Departing
Request updated balance from the smart meter
Upon receiving the new balance, update the database info
Arriving
Fetch driver balance, and send it to the smart meter

*Figure 9: Top-level data flow of the entire system.*

*Figure 10: Flow of data on the server side.*

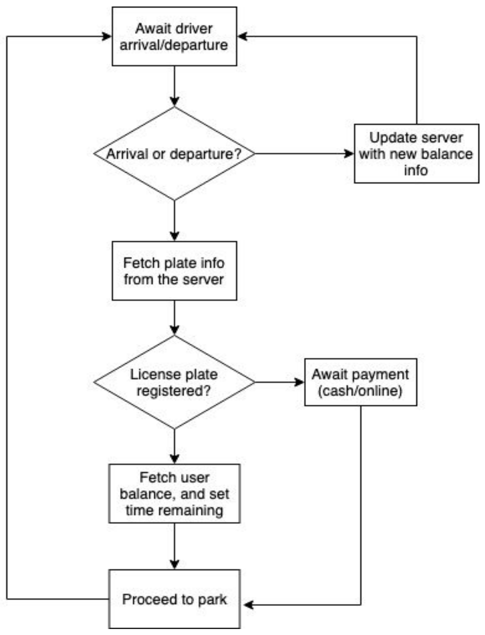

Await driver arrival/departure
Arrival or departure?
Update server with new balance info
Fetch plate info from the server
License plate registered?
Await payment (cash/online)
Fetch user balance, and set time remaining
Proceed to park

*Figure 11: Software logic flow on the meter/client side.*

### 2.2.7 Design Alternatives

One alternative we considered was in the sleep/wake module of the Raspberry Pi. In our first design, we supplied constant power to the microcontroller, and would use a relay to cut power to the Raspberry Pi to put it to sleep. To wake it up, the microcontroller would operate the relay to allow wall power to flow to the Pi. This design originally worked, but it’s unsafe to shut down the Pi by abruptly cutting power to it. Instead, we took advantage of the fact that earlier versions of the Pi used to have a power button, and even though this functionality was removed, the capability to wake the Pi by simply shorting two of its GPIO pins remained.

# 3. Design Verification

## 3.1 Raspberry Pi Power Saving Verification

We recorded the current drawn by out Raspberry Pi, which could reflect the total power consumption of our design.

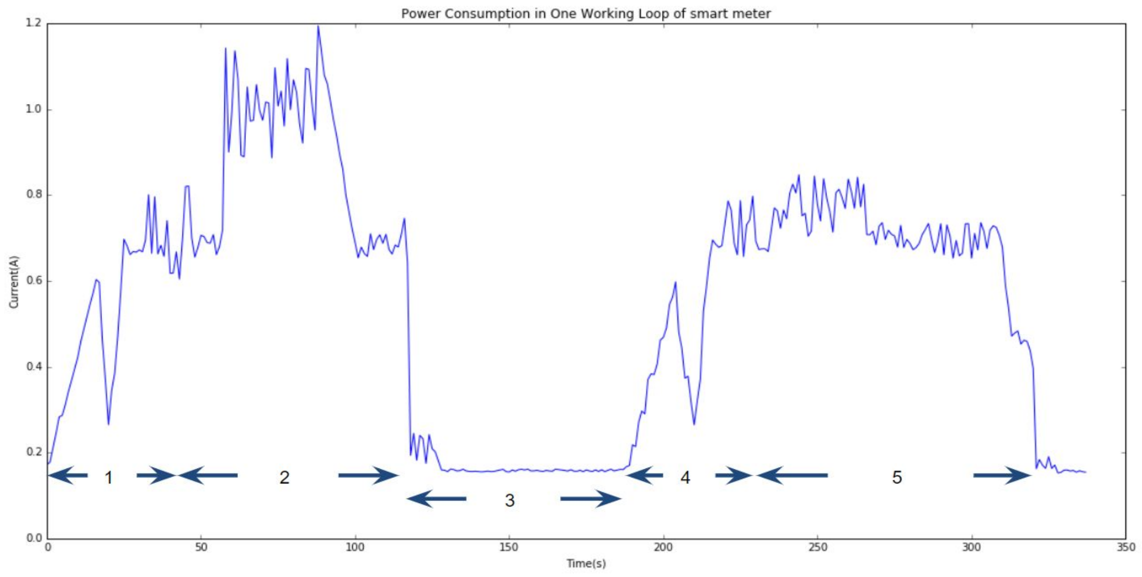


*Figure 12: Current value of the parking assistance device.*
*area1: Raspberry wake up (car arrives)*
*area2: Plate recognition and data transmission*
*area3: Raspberry sleep*
*area4: Raspberry wake up (car leaves)*
*area5: Data transmission while parking ending*

From the above data, we could easily calculated the power consumption of one round of parking by applying basic formula:

$$\text{Total Power} = \text{average current value} * \text{DC voltage applied} \qquad (2)$$

From our data, we could get that average working current is 0.63 when Raspberry Pi is on,  then we would consume around 3W. However, if the Raspberry Pi is in halt state, we would only use 0.2A because of backlight of LCD,, which is 1W power. If we assume the average parking time is one hour, and overall usage rate of parking spot is 70%, the average power consumption for one device during the day time(8am - 5pm) would be 1.112W.
 We could turn the LCD off to save the power further. Then the power would be 0.2W for microcontroller and ultrasonic sensors. Considering the numerous street parking meters, which depends on the official website's record[6], there are over 2000 University of illinois meters on campus. we still assume the average parking time is 1 hour, and the overall usage rate is 70% from 8am to 5pm. For one week the campus parking meter would consume 47628kW, which is 13.23 kWh. However, if we let Raspberry Pi keep awaking, the power consumption would reach 45 kWh.

## 3.2 Other Verification

Table 1. Requirements and Verifications for Overall Flow

| Requirement | Verification | Result (Functional/Non-Functional) |
|---|---|---|
| The module must be able to communicate with the main server to send and retrieve license plate data. | A. Create a database entry with an associated predefined license plate on the server side.<br>B. From the client, send a (debug) request to retrieve the plate entry and its associated balance. | Functional<br><br>The parking module is able to look up and create entries in the database for existing and new license plates, respectively, in under 1 second. |
| The parking module must be able to identify a license plate when there is one present within 0.5 to 1 meter, of the camera. | A. Position a license plate 0.5 meters from the module's camera.<br>B. Check to see that the license plate number displayed on the LCD screen is the same as the license plate number held in front of the parking module. | Functional<br><br>The parking module is able to recognize a license plate at less than 0.3 and more than 1 meter away from the main camera. |

Table 2. Requirements and Verifications for Microcontroller to Peripheral Functionality

| Requirement | Verification | Result (Functional/Non-Functional) |
|---|---|---|
| The microcontroller must be able to display a 16x2 character message on the LCD screen. | A. Program the microcontroller to send a predefined 32-character ASCII message to the LCD display peripheral.<br>B. Verify that the corresponding message appears on the display. | Functional |
| The microcontroller must be able to read a distance from the ultrasonic sensor. | A. Command the microcontroller to display the distance read from the ultrasonic sensor.<br>B. Position an object in front of the ultrasonic sensor. Verify that the distance displayed changes in correspondence with the position of the object. | Functional |

Table 3. Requirements and Verifications for Microcontroller to Pi Module

| Requirement | Verification | Result (Functional/Non-Functional) |
|---|---|---|
| The microcontroller must be able to send and receive data over UART with a baud rate of 9.6kbaud. | A. Using PySerial, open a serial connection between the Raspberry Pi and the microcontroller (with 9.6kbaud transmission speed).<br>B. Send a command to the microcontroller to give it a predefined message.<br>C. Command the microcontroller to send back the predefined message.<br>D. Verify that the message received is the same as the message sent. | Functional |
| The microcontroller must be able to wake the Raspberry Pi from sleep mode via. | A. Manually put the Raspberry Pi to into sleep mode by entering “sudo shutdown -h now” into the console.<br>B. From the microcontroller, program it to manually send the wake up command upon a debug button (physical debug button) being pressed.<br>C. Verify that the Raspberry Pi is able to wake up within 30 seconds of the button being pressed. | Functional<br><br>The microcontroller is able to wake the Raspberry Pi from sleep mode, but the Raspberry Pi requires ~30 |

| | | |
|---|---|---|
| | | seconds to boot up before it can recognize plates. |
| The Raspberry Pi must be able to send and receive a balance from the microcontroller, and the microcontroller must be able to update this balance (while a car is parked). | A. From the Raspberry Pi, send the command to update the microcontroller’s stored license plate.<br>B. From the Raspberry Pi, send the command to display the license plate on the LCD screen.<br>C. Verify that the plate number displayed on the LCD screen is the same as the one sent. | Functional |

## 4. Costs

### 4.1 Parts

**Table 1 Parts Costs**

| Part | Manufacturer | Retail Cost ($) | Bulk Purchase Cost ($) | Actual Cost ($) |
|---|---|---|---|---|
| Raspberry Pi Model 3 B+ | Raspberry pi | 35 | 35 | 35 |
| Raspberry Pi Camera Module V2-8 Megapixel,1080p | Raspberry pi | 24 | 24 | 24 |
| SanDisk Ultra 16GB Ultra Micro SDHC UHS-I/Class 10 Card | SanDisk | 7 | 7 | 7 |
| 5V Standard 16 x 2 Character Blue Backlight LCD Display Module | uxcell | 17 | 17 | 17 |
| Ultrasonic sensor hc-sr04 (3942) | Adafruit | 3.95 | 3.95 | 3.95 |
| Atmega 328P | Microchip Technology | 1.96 | 1.6274 | 1.96 |
| **Total** | | 88.91 | 88.54 | 88.91 |

### 4.2 Labor

Rate: $50/hr

Bo Wang: 50 * 2.5 * (10 * 3) = $3750

Christopher Santoso: 50 * 2.5 * (10 * 3) = $3750

Ximin Lin: 50 * 2.5 * (10 * 3) = $3750

# 5. Conclusion

## 5.1 Accomplishments

From our final demo, the overall flow depicted in Figure 1 is completely working. Upon arrival of a (mock) car, the microcontroller is able to wake the Raspberry Pi from sleep mode, and the Raspberry Pi is able to take a picture of the license plate, look it up in the database, and fetch its corresponding balance. In the case when a new license plate is seen, a new entry with 0 balance is created on the server. The Pi is able to successfully communicate with the microcontroller and send the corresponding balance and to go sleep. The microcontroller is then able to deduct a set fee while the car is parked, and upon the car leaving, successfully send the balance back to the Raspberry Pi after waking it and update the final balance.

## 5.2 Uncertainties

a. In the design of Raspberry control, we let GPIO3 on Pi keep receiving floating data, which would cause potential problem to while device is running.
b. We did not include state information while plate recognition, so our device could not work correctly  if there are two cars from different state but have same plate number.
c. Since it would take some time to end the parking, if one car parks in the same parking spot just after other car leaves, we cannot ensure our device would respond.

## 5.3 Ethical considerations

One of the primary concerns to be aware of is the fact that user data and their credit card account information is being stored and charged while utilizing our service. For this reason, it's imperative that user information is properly encrypted and transmitted in a safe and secure manner. This will require us to research our options for when it comes to storing and transferring data.

An issue brought up during our discussions revealed that a similar automated parking collection meter was installed in Palisades Park (1). As a result, user complaints were numerous, and stores lost customers because they simply didn't want to deal with the complications of these new meters. Furthermore, the city was able to raise a lot of money from drivers going over time, since tickets would be automatically billed to their address. This practice seems to clearly violate IEEE ethics code #5: "to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems." Because the use of their meters was confusing and unclear, many users suffered fines, which only seems to exacerbate the concern of automated processes. We seek to improve user parking experience with a more fully-featured automated driver parking assist.

Our meter will provide clear signals to the driver when a violation has occurred, and one of our biggest goals is to make the user interface as simple as possible. To reap benefits from users' confusion is unethical, so it's very important that our design choices make sense to the driver and that our signals and instructions are easy to follow.

IEEE ethics rule #6 ("to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations") brings up another concern about storing and transmitting user data. Only one of us has experience with storing and retrieving encrypted user data for public services, so in doing so, we acknowledge that our knowledge in the field is limited, so in order to prevent unsafe practices, we should consult online resources and utilize existing tools for secure transfer and storage of user data.

## 5.4 Future work

One modification we could make is to change the Raspberry Pi's camera to a night-vision camera. This comes at a low additional cost (~36 USD each), and enables the camera to see at night. Another additional feature that could be implemented are smarter ways to detect whether a car is incoming or leaving as opposed to a random object passing by. One way this can be done is to have an average of samples taken from the ultrasonic sensor and apply a prior to eliminate outliers from sampling error. More complex methods can be used as well to detect more complex parking violations; for example, when a picture of the license plate is taken, based on the position of the license plate and the distance (in pixels) from the center of the image, assuming the parking meter is centered in the parking spot, one can determine whether a car is parked too far or too close to the edge of its parking space.

Another edge case we can potentially account for is during a busy day when another car is detected as soon as a car leaves. In this instance, since it requires ~30 seconds for the Pi to boot up and ~20 seconds for the Pi to go to sleep, we can simply leave the Raspberry Pi on for ~2 minutes after a car has left to ensure that no additional license plate is detected within that time.

Aside from these changes to the original design, an app and website for user sign-up and University Parking notification can be built on top of the web API we've created.

## References

[1] northjersey. (2019). Palisades Park: Digital parking meters a chance at a camera windfall. [online] Available at: https://www.northjersey.com/story/news/bergen/palisades-park/2018/06/19/palisades-park-digital-parking-meters-camera-windfall/710805002/ [Accessed 3 Feb. 2019].

[2] Learn.adafruit.com. (2019). How PIRs Work | PIR Motion Sensor | Adafruit Learning System. [online] Available at: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work [Accessed 4 Feb. 2019].

[3] MaxBotix Inc. (2019). Ultrasonic vs Infrared (IR) Sensors - Which is better? - MaxBotix Inc.. [online] Available at: https://www.maxbotix.com/articles/ultrasonic-or-infrared-sensors.htm [Accessed 4 Feb. 2019].

[4] Officer.com (2018). OpenALPR Software Upgrade: Brings Accuracy Up To 99.02%. [online] Available at: https://www.officer.com/command-hq/technology/traffic/lpr-license-plate-recognition/press-release/21031077/openalpr-technology-inc-openalpr-software-upgrade-brings-accuracy-to-9902 [Accessed 18 Feb. 2019].

[5] github.com. (2019). OpenCV 3 License Plate Recognition Python. [online] Available at: https://github.com/MicrocontrollersAndMore/OpenCV_3_License_Plate_Recognition_Python [Accessed 25 Mar. 2019].

[6] illinoi.edu. (2019). number of University parking meters on campus: https://parking.illinois.edu/parking_items/meters  [Accessed 25 Mar. 2019].

## Appendix A Requirement and Verification Table

**Table 2 System Requirements and Verifications**

| Requirement | Verification | Verification status |
|---|---|---|
| 1. Requirement<br><br>The module must be able to communicate with the main server to send and retrieve license plate data. | 1. Verification<br>a. Create a database entry with an associated predefined license plate on the server side.<br>b. From the client, send a (debug) request to retrieve the plate entry and its associated balance.<br>c. Verify that the client contains the same entry information as what was created in (A). | Y |
| 2. Requirement<br><br>The parking module must be able to identify a license plate when there is one present within 0.5 to 1 meter, of the camera. | 2. Verification<br>a. Position a license plate 0.5 meters from the module's camera.<br>b. Check to see that the license plate number displayed on the LCD screen is the same as the license plate number held in front of the parking module. | Y |
| 3. Requirement<br><br>The parking module must be able to fetch an associated balance from the server within 1 minute of identifying the license plate. | 3. Verification<br>a. Position the plate until it is identified by the meter.<br>b. Verify that the debug message on the LCD screen correctly displays the database entry associated with the licence plate. | Y |
| 4. Requirement<br><br>The parking module must be able to detect a car leaving the parking space, which is defined as the car moving away more than 1 meter from the meter, and no identical license plate being identified within the next minute. | 4. Verification<br>a. Position the license plate 0.5 meters from the parking module, and hold it in place for five minutes to signal the meter that a car is parked.<br>b. Move the plate away from the module to a distance of 1 meter (or greater), and keep it out of range for two minutes. | Y |

| | | |
|---|---|---|
| 5. Requirement<br><br>The microcontroller must be able to display a 16x2 character message on the LCD screen. | 5. Verification<br><br>a. Program the microcontroller to send a predefined 32-character ASCII message to the LCD display peripheral.<br>b. Verify that the corresponding message appears on the display. | Y |
| 6. Requirement<br><br>The microcontroller must be able to read a distance from the ultrasonic sensor | 6. Verification<br><br>a. Command the microcontroller to display the distance read from the ultrasonic sensor.<br>b. Position an object in front of the ultrasonic sensor. Verify that the distance displayed changes in correspondence with the position of the object. | Y |
| 7. Requirement<br><br>In displaying a status message (1. Empty Balance and 2. Obstructed Spot or Plate, 3. Restricted Hours), the LCD module will take no more than 3 seconds to display the correct status message upon the correct conditions being met. | 7. Verification<br><br>a. Load the parking module with an nearly-empty balance (0.40 USD).<br>b. Allow the balance to empty by keeping a parked car in the same spot for more than 5 minutes.<br>c. Verify that the LCD display reflects the "Empty Balance" message.<br>d. Place an object within 3 meters, 1 meter, and 1cm of the parking module with no license plate in view of the<br>e. Verify that for each of the 3 distances, the "Obstructed Parking Spot" message displays on the LCD.<br>f. Set the meter's restricted hours to -1 and +1 hours of the current time.<br>g. Place an object with a license plate within 0.5 meters of the parking meter camera view.<br>h. Verify that the LCD displays the "Restricted Parking Hours" message. | Y |

| | | |
|---|---|---|
| 8. Requirement<br><br>The microcontroller must be able to send and receive data over UART with a baud rate of 9.6kbaud. | 8. Verification<br><br>a. Using PySerial, open a serial connection between the Raspberry Pi and the microcontroller (with 9.6kbaud transmission speed).<br>b. Send a command to the microcontroller to give it a predefined message.<br>c. Command the microcontroller to send back the predefined message.<br>d. Verify that the message received is the same as the message sent. | Y |
| 9. Requirement<br><br>The microcontroller must be able to wake the Raspberry Pi from sleep mode via. | 9. Verification<br><br>a. Manually put the Raspberry Pi to into sleep mode by entering "sudo shutdown -h now" into the console.<br>b. From the microcontroller, program it to manually send the wake up command upon a debug button (physical debug button) being pressed.<br>c. Verify that the Raspberry Pi is able to wake up within 30 seconds of the button being pressed. | Y |
| 10. Requirement<br><br>The Raspberry Pi must be able to send and receive a balance from the microcontroller, and the microcontroller must be able to update this balance (while a car is parked). | 10. Verification<br><br>a. From the Raspberry Pi, send the command to update the microcontroller's stored license plate.<br>b. From the Raspberry Pi, send the command to display the license plate on the LCD screen.<br>c. Verify that the plate number displayed on the LCD screen is the same as the one sent. | Y |