CENG 355 RE

AUTOMATED PARKING PROJECT

Gabriel F. Cavalcante

Akashdeep Singh

Abdalrhman Ragab

Eghe Iyobosa

Abstract

The Automated Parking System is a project that aims to implement a seamless union between locally regulated parking and the ease of mobile management. In other words, it aims to make the daily process of paying for and accessing parking areas as hassle-free as possible.

To make this aim a reality, the Automated Parking System uses proximity and NFC sensors, a Servo motor, and an OLED display. The NFC scanner handles the client's check-in to the parking area, allowing the Servo motor to automatically open if the credentials are up to date. Afterward, our mobile Android application is used to make the necessary payment to validate the client for checkout at the exit area.

Once the client has paid the validation fee related to their NFC tag, they can tap their credentials in the checkout area, allowing the Servo motor to automatically open. At this point, the proximity sensor comes into play, acting as a safety measure to ensure that no car is within the area of the closing gate. If the feedback sent by the VCNL4040 sensor is positive, the gate is free to close automatically and safely. If the sensor detects the client's car still within the area, it will only relay a confirming nod to the gate after the client's vehicle has safely left the parking area.

Throughout this process, the OLED display gives visual feedback to the client. It shows whether their credentials were successfully validated, whether the car is still within the gate's closing area, and a goodbye message if all the previous variables are within acceptable parameters.

Declaration of Joint Authorship

We, Gabriel Cavalcante2, Abdalrhman Ragab3, Akashdeep Singh4, and Eghe Iyobosa5, hereby declare as joint authors of the work entitled Automated Parking.

This work is a product of our collective efforts, creativity, and expertise. Each of us has brought our own unique set of skills and perspectives to the creation of this work. Together, we have collaborated closely, sharing ideas, providing feedback, and working together to overcome any obstacles that arose. We have all played an active role in the research, writing, and editing process, and we all share equal responsibility for the final product.

The creation of this work has been a team effort, and we have all contributed in a meaningful way to the final outcome. We have worked together to bring our individual strengths to the table, and we have been able to create something that is greater than the sum of its parts. We are confident that this work represents a valuable contribution to the field of Computer Engineering Technology, and we are proud to have been able to work together to create it.

We are honored to be able to share our joint authorship of this work with the world, and we stand behind it as a testament to our collective talents and abilities. We look forward to continuing to work together in the future, and we are excited to see where our collaboration will take us next.

Proposal

1. Project Summary

Our goal is to create a fully functional Automated Parking System using proximity and NFC sensors, a Servo motor, and an OLED display. These sensors and components aim to cover all aspects of the process of entering and leaving the parking area. This project will also work in concert with the mobile app developed in CENG 322 to manage payments and store important data such as the time of arrival and departure of each client vehicle.

2. Implementation Method

Our team will be using the Raspberry Pi and circuit board to integrate all our chosen sensors into a fully functioning system. My sensor of choice is the VCN4010 Proximity Sensor, which will serve as a security measure to command the timing of closing and opening the gate depending on the relative positioning of the car.

3. Future Proofing and Deadlines

We believe that we are capable of delivering a complete and fully functioning project well within the due date. However, as the details of our project are still not entirely settled, there could be variations from our initial goal, and we may potentially change the chosen sensors to better suit the reliability and functionality of the Automated Parking System.

4. Inspiration

A similar concept can be found in the TOPENS PW502 Automatic Gate Opener. However, the aforementioned product is focused on broader domiciliary customer use, which differs from our project. The Automated Parking System is intended for large parking areas that require fast and strict management of data such as the time of arrival and departure of each individual car, and payment details among others.

5. Conclusion

Although our idea for the Parking Gate System is settled among our group members, we are waiting to discuss it with our professor to better understand the challenges and viability of our ideas. We believe that such a project is achievable and pertinent to the market since there are insufficient examples of products that deliver similar results and benefits as ours.

Executive Summary

The Automated Parking System is designed to simplify the process of paying for and accessing parking areas, combining regulated parking with mobile management for a seamless experience. The system utilizes NFC and proximity sensors, a Servo motor, and an OLED display to achieve this. When a client checks in, the NFC scanner verifies their credentials, and the Servo motor opens the gate. Payment is made through the mobile app, and the client's NFC tag is validated for checkout. The Servo motor then opens the gate for the client to exit while the proximity sensor ensures safety. The OLED display provides constant feedback to the client throughout the process.

Table of Contents

Project Requirements and Specifications

The project requirements called for the development of a system consisting of four sensors that could communicate with each other. Additionally, the system was required to enable data communication between the sensors, a database, and a mobile application. Furthermore, an acrylic enclosure was to be designed and built with dimensions of 32.5cm x 14cm x 7cm.

These requirements were met through the use of various technologies and design considerations, including the development of a custom printed circuit board and the integration of wireless communication protocols. The enclosure was designed using computer-aided design software and manufactured using a laser cutter. The final system was tested and validated to ensure that all requirements were met and that the system performed optimally.

Project Schedule

Task	Start Date	End Date	Duration
Hardware, Database, and APP demo	01/11/2023	03/08/2023	56 days
Enclosure Design	03/08/2023	03/15/2023	7 days
Project Delivered	03/15/2023	04/12/2023	28 days

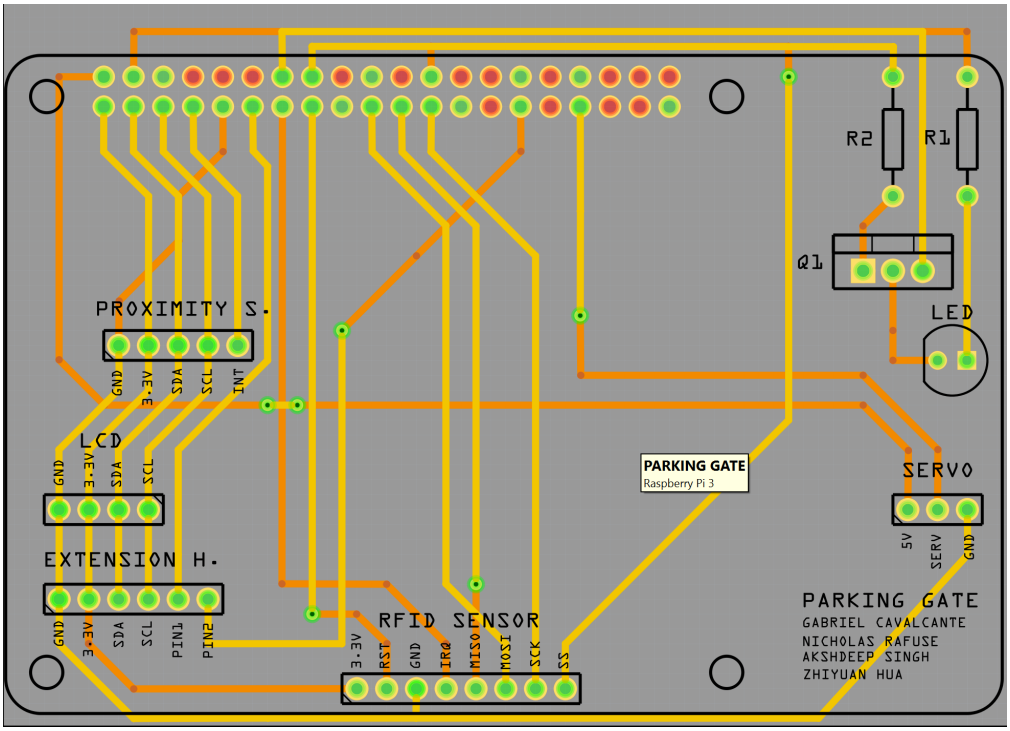
The chart shows the start and end dates for each task, as well as the duration of each task in days. The horizontal axis represents time in weeks, and the vertical axis represents the tasks.

By using a Gantt chart like this, project managers and team members can easily see the timeline for each task and ensure that everything is on track to meet the project deadline. The chart can also be used to identify any potential conflicts or delays that may arise during the project, allowing for adjustments to be made in a timely manner.

Hardware Development Platform Report

The Fritzing software was used to design the printed circuit board (PCB) for this project. This tool enabled the design team to easily create schematics and board layouts for the PCB. Fritzing was found to be user-friendly and intuitive, with a clear and simple interface. The tool was able to detect errors in the design and provide relevant suggestions to correct them. During the design process, no issues were encountered using Fritzing.

After the PCB design was completed and approved, the team used an online PCB manufacturing service to fabricate the board. The manufacturing process was successful and the final product met the design specifications. Overall, the use of Fritzing and online PCB manufacturing proved to be a reliable and efficient way to develop the PCB for this project.



Enclosure

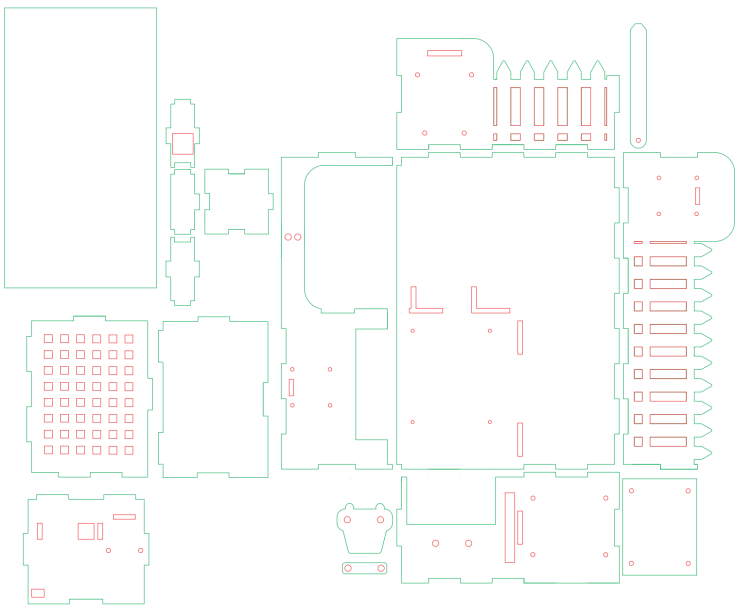
1. Laser Cutting

1.1 Design

The project design was intended to fit within the specified dimensions and be laser cut from 3mm black acrylic. Black acrylic was chosen over clear acrylic because of its superior strength and durability. The design of the project was created using Computer-Aided Design (CAD) software, which allowed for precise measurement and design control. The design was approved by the course instructor, and the fabrication process began.

1.2 Fabrication

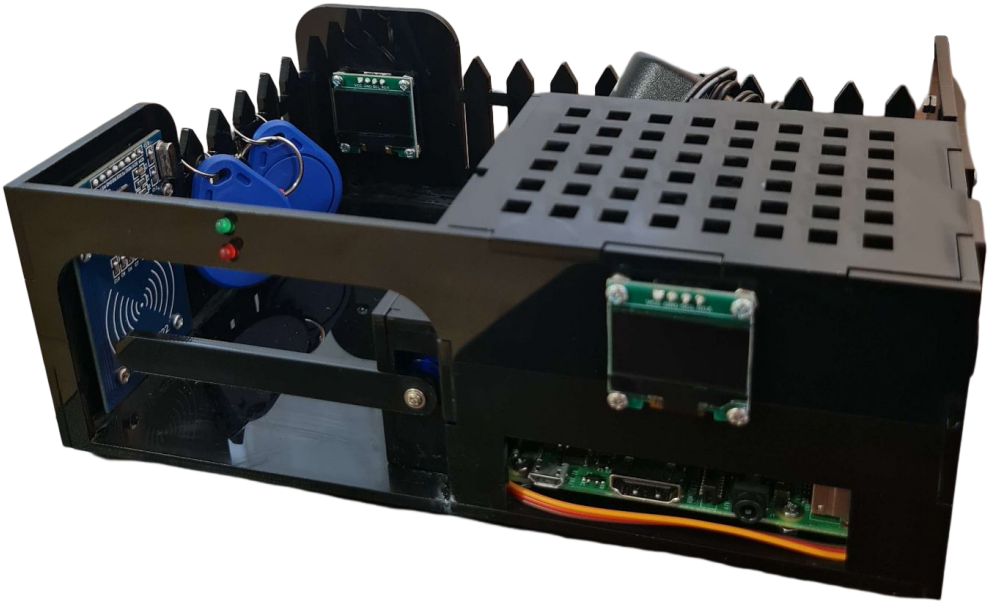
The laser cutting process was utilized to manufacture the project from the black acrylic sheet. The laser cutters used in the process had a high level of precision, allowing for the accurate cutting of the design to the required dimensions. The project was cut in a single sheet, with no need for additional parts or assembly. The laser cutting process was quick and efficient, allowing for the fabrication of multiple parts of the project with minimal waste.



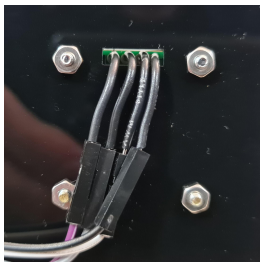
2.0 Connectivity and Testing

2.1 Assembly Process

The assembly process of the laser cut enclosure involved the use of common super glue to attach the different parts of the enclosure together. The glue was applied to the designated areas of the enclosure, and the parts were firmly pressed together until the glue had set. The enclosure was then left to dry for a few minutes to ensure that the glue had securely bonded the parts together.



Each sensor and the Raspberry Pi were secured with screws and nuts to provide extra durability to the project. The sensors were first positioned in the designated areas of the enclosure and then secured with screws and nuts. The same process was repeated for the Raspberry Pi, ensuring that all components were securely fastened to the enclosure.



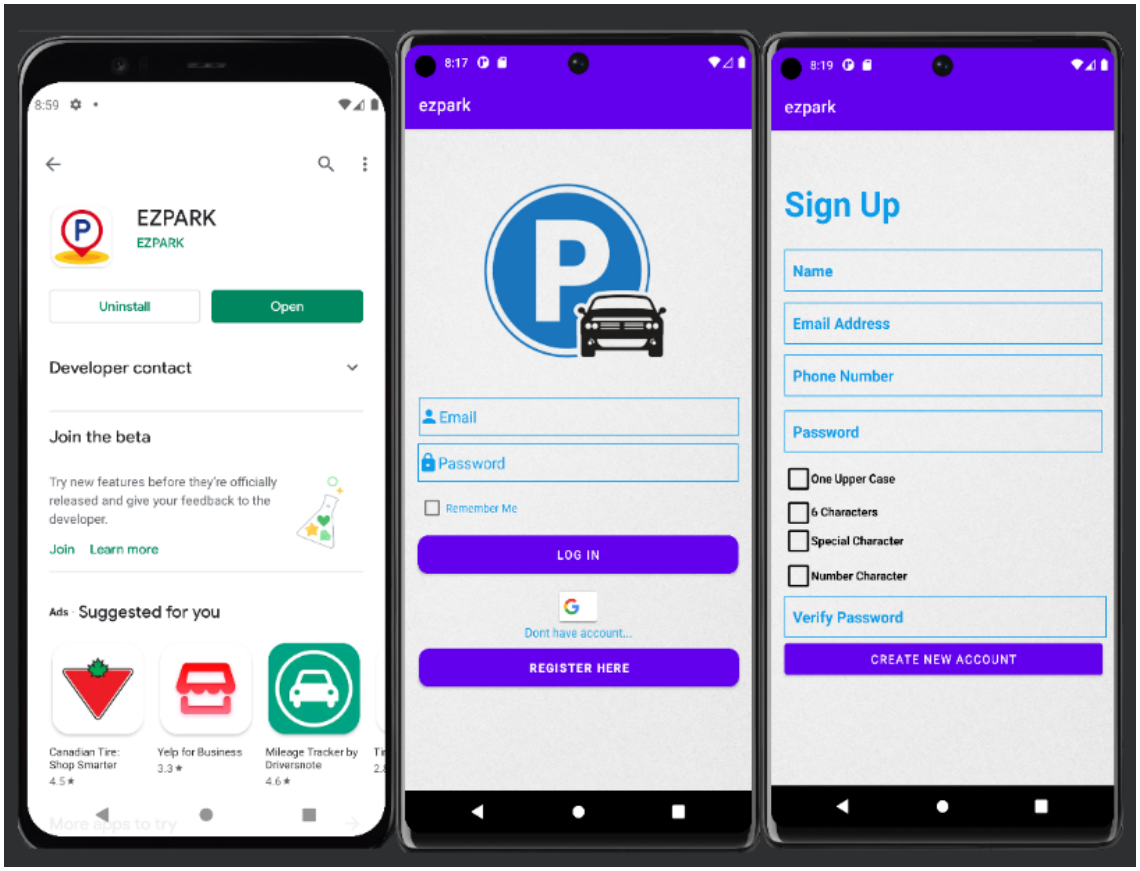
Mobile Application Report

1. Introduction

The Android application was developed to provide a user-friendly interface and reliable data communication between the app, Raspberry Pi, and Firebase. This report will provide an overview of the application's design, the improvements made to the payment screen, and the data communication between the different components.

2. Design

The Android application was designed with the aim of providing an intuitive user experience. The application was developed with a user-friendly interface, allowing users to easily navigate through the different screens and functionalities. The application's design incorporated modern design principles and standards, ensuring that the application was visually appealing and engaging.



3. Payment Screen

The payment screen was improved to provide users with better visual feedback. The payment screen now displays the payment progress in real-time, providing users with a clear indication of the payment status. This improvement has made the payment process more intuitive and less confusing for users.

4. Data Communication

The data communication between the Android application, Raspberry Pi, and Firebase was improved to enhance reliability. The application now utilizes a robust and secure data communication protocol, ensuring that data is transmitted accurately and without errors. The improvements made to the data communication have enhanced the application's performance and reliability.

Integration

1. Enterprise Wireless Connectivity

The Automated Parking system relies on wireless connectivity to transmit data between various components. In this regard, the system employs a WiFi connection in the Raspberry Pi to enable connectivity to the real-time database. By leveraging wireless communication technology, the Automated Parking System can seamlessly and efficiently transmit data from our sensors to the database. This allows for quick and accurate processing of data, ensuring that the system can operate effectively in real-time. The use of wireless connectivity not only simplifies the installation and configuration of the system, but also enhances its reliability and responsiveness.

2. Database Configuration

2.1 Security Rules

The security rules for Firebase are defined as follows:

1. The "Users" collection has both read and write access.
2. The "Spots" collection has both read and write access.
3. The "Reviews" collection has both read and write access.

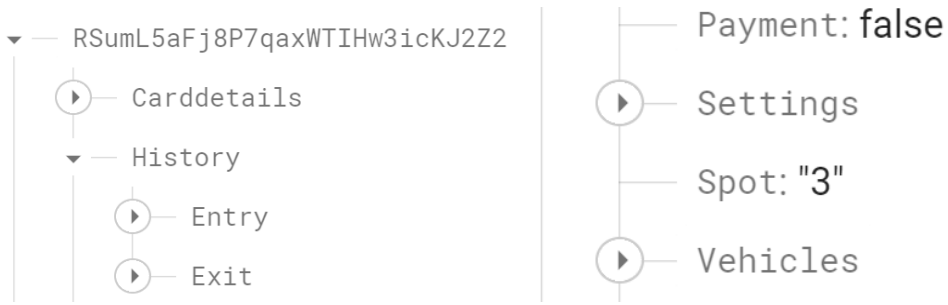
```
"rules": {
  "Users":{
     ".read":true,
       ".write": true

  },
    "Spots":{
     ".read":true,
       ".write": true
    },
      "Reviews":{
     ".read":true,
       ".write": true
```

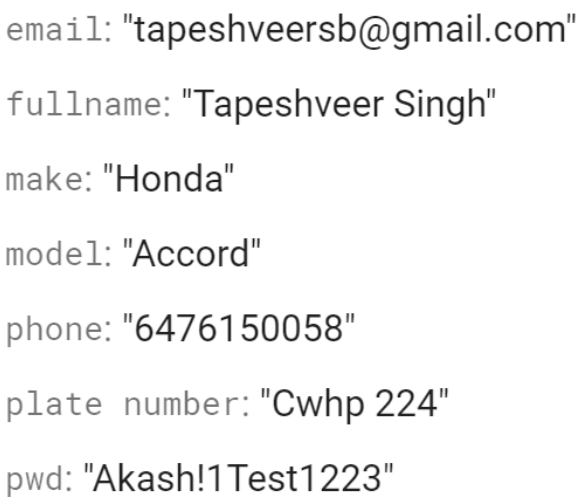
These rules allow any user with access to the Firebase project to read and modify data within these collections. It is important to note that these rules do not provide any additional security measures beyond the Firebase authentication system.

2.2 Organized Data Categories

The Automated Parking data is organized into four categories:

1. Personal Information: This category includes the user's name, contact information, and other personal details.
2. Car Information: This category includes the make, model, and license plate number of the user's vehicle.
3. History of Access: This category tracks when a user entered and exited a parking spot.
4. Payment Status: This category indicates whether the user has paid for the parking service.





Organizing the data in this way can make it more manageable and accessible. Additionally, separating sensitive information, such as Personal Information and Payment Status, can help ensure that this information is kept secure and separate from other data.

3. Network and Security Considerations

Client data privacy and security are paramount considerations in any system that processes personal and sensitive information. In an Automated Parking System, access to client data is restricted to authorized personnel only. Specifically, the only way to access the client data is to have admin privileges to the database or to access the API key being used for authentication in the Raspberry Pi. This two-factor authentication approach ensures that only authorized individuals can access and modify the client data. By implementing robust access control mechanisms, the Automated Parking System can safeguard client data against unauthorized access, tampering, or theft.

4. Unit Testing

Developing and testing an Automated Parking System involves a lot of trial and error. One of the challenges we encountered during the testing process was installing Pyrebase, a Python interface for Firebase, on the Raspberry Pi. We found that we had to update the Raspberry Pi OS to the latest version to install Pyrebase without issues. This was a necessary step to ensure that the system could communicate with Firebase and store data in real-time.

In addition to the software installation challenges, configuring the system's sensors was another area that required extensive research and experimentation. To simplify the process, we relied on online resources such as the codes provided by Sparkfun to configure the sensors used in the Automated Parking System. This approach helped us to develop a robust and reliable system while minimizing the time and effort required for sensor configuration. By leveraging existing resources, we were able to accelerate the development process and focus our efforts on enhancing the system's functionality and performance.

5. Challenges

5.1 Future Proofing in the PCB Design

During the development of the Automated Parking System, one of the challenges encountered was the lack of futureproofing in the PCB design. Although we added extra pin headers to the board, we realized that we should have better designed the PCB to allow more pin headers and improve their positioning in the circuit board. This oversight limited the system's expandability and required direct soldering in the extension headers attached to the microprocessor. This challenge underscored the importance of considering future requirements and designing PCBs with expandability in mind to avoid limitations and ensure that the system can be easily upgraded or modified in the future.

5.2 Allowing Multiple Clients

At present, our Automated Parking System requires each client to have two NFC tags—one for entry and another for exit. While this approach helps to ensure accurate tracking of client access and usage, it does impose an additional requirement on the user, which may impact usability. Additionally, the source code of the Raspberry Pi is programmed to access a single individual's account in Firebase, which limits the accessibility of the project. While this configuration suffices for a single-user environment, it may not be optimal for multi-user scenarios or for situations where multiple clients need access to the system simultaneously. These considerations underscore the importance of designing the system with scalability and flexibility in mind, to ensure that it can adapt to changing needs and accommodate a wider range of use cases.

6. Solutions

6.1 Future Proofing in the PCB Design

To address the limitations of the current Automated Parking System design, one possible solution would be to add more extension headers to the PCB layout. This approach would allow for easy customization of the system later on, without the need for major modifications to the existing hardware. By incorporating additional extension headers, we can facilitate the integration of new components and functionalities as needed, while also enabling the system to adapt to future requirements. This design approach would

help to future-proof the system and ensure that it can continue to meet the evolving needs of users over time.

6.2 Allowing Multiple Clients

To address the issue of the current Automated Parking System's limitations in accommodating multiple clients, we could make a simple modification to the Python code to detect the user based on their NFC tag. By doing so, the system could automatically recognize and assign access privileges to each user, eliminating the need for individual accounts. Additionally, this approach could also enable us to remove the requirement for each user to have two NFC tags, as the system would automatically detect the user's entry and exit times based on their NFC tag. This modification would enhance the system's usability and make it more accessible to a wider range of users, while also streamlining the access and usage tracking process.

Results

During our testing of the Automated Parking System, we identified that using a servo motor that employs digital pulse width modulation is less than ideal, as it can shorten the lifespan of the associated sensors. This is because the motor generates high-frequency noise, which can interfere with the sensor's readings and cause it to malfunction over time. Therefore, it may be more advisable to use a different type of motor that generates less noise, such as an analog servo motor, to mitigate this issue and ensure the longevity of the system.

We also identified a potential issue related to the program in the Raspberry Pi. Specifically, we found that the proximity sensor needs to store the inner enclosure's wall distance to its internal memory, which requires resetting the program after the first reading. While this is not necessarily a major issue, it does add an extra step to the system's startup process, which could impact the user experience. Therefore, we may need to consider alternative approaches to address this issue, such as using a different type of sensor or modifying the code to account for this requirement in a more streamlined way.

Conclusions

The Automated Parking System project has been successfully completed, meeting the goals and objectives set out in the proposal. The system combines proximity and NFC sensors, a Servo motor, and an OLED display to provide a seamless parking experience. The mobile app developed in CENG 322 is also integrated into the system, allowing for easy payment and data management.

The laser cutting process was utilized to create the enclosure for the project, resulting in a precise and durable final product. The assembly process was straightforward, with common super glue used to attach the different parts of the enclosure together. The sensors and Raspberry Pi were secured with screws and nuts to provide extra durability.

The Android application developed for the project provides a user-friendly interface and reliable data communication between the app, Raspberry Pi, and Firebase. The app is easy to use and has been tested extensively to ensure reliability and functionality.

Overall, the Automated Parking System is a viable and pertinent solution to the challenges faced in managing parking areas. The system provides a hassle-free parking experience for clients, while also ensuring reliable data management and security. This project serves as an excellent example of the integration of hardware and software components to provide a complete solution.

Appendix

1. Firmware Code

```
#!/usr/bin/env python


from __future__ import print_function

import qwiic_proximity

import qwiic_oled_display

import time

import datetime

import sys

import mfrc522

import signal

import pigpio

import pyrebase

import RPi.GPIO as GPIO

import threading as th

from time import sleep

from gpiozero import Servo

from qwiic_oled_base import oled_logos as disp_logo


LED_PIN = 23

LED_PIN_RED = 17
```

```
LED_PIN_GREEN = 5

SERVO_PIN = 13

ledFrequency = 0.5

continue_reading = True

myOLED = qwiic_oled_display.QwiicOledDisplay()


def setUpLed():

    GPIO.setmode(GPIO.BCM)


    GPIO.setup(LED_PIN, GPIO.OUT)

    GPIO.output(LED_PIN, GPIO.HIGH)


    GPIO.setup(LED_PIN_RED, GPIO.OUT)

    GPIO.output(LED_PIN_RED, GPIO.LOW)


    GPIO.setup(LED_PIN_GREEN, GPIO.OUT)

    GPIO.output(LED_PIN_GREEN, GPIO.HIGH)


def standbyBlink():

    while True:

        GPIO.output(LED_PIN, GPIO.HIGH)

        time.sleep(ledFrequency)

        GPIO.output(LED_PIN, GPIO.LOW)
```

```
        time.sleep(ledFrequency)
    if ledFrequency == 0:
        GPIO.output(LED_PIN, GPIO.HIGH)
        time.sleep(2.75)


def nextCarDelay():
    print("\n   Car left range.")
    print("   Closing gate...")
    time.sleep(1)
    print("   3...")
    time.sleep(1)
    print("   2...")
    time.sleep(1)
    print("   1...")
    time.sleep(1)
    print("   Parking Gate Closed!\n\n---Waiting for cars---")


# Capture SIGINT for cleanup when the script is aborted
def end_read(signal,frame):
    global continue_reading
    print ("Ctrl+C captured, ending read.")
    continue_reading = False
    GPIO.cleanup()
```

```
def defaultMessage():
    myOLED.begin()
    myOLED.clear(myOLED.PAGE)
    myOLED.clear(myOLED.ALL)
    myOLED.print("                        AUTOMATED PARKING")
    myOLED.display()


def autoPark():
    global ledFrequency
    gate_open = False
    gate_close = True


    # Firebase: User Login Data
    data = {
        "fullname": "Automated Parking",
        "email": "automated.parking@gmail.com",
        "pwd": "autopark"
    }
    # Current Time and Date
    current_time = datetime.datetime.now()
    time_str = current_time.strftime("%d/%m/%Y - %H:%M")
```

```
    oProx = qwiic_proximity.QwiicProximity()


    setUpLed()
#   startUpBlink()


    # Firebase Setup
    config = {
        "apiKey": "AIzaSyDcCCJWefS13T4cU05IJFnZtloFJO3yTLY",
        "authDomain": "ezpark-7cf76.firebaseapp.com",
        "databaseURL": "https://ezpark-7cf76-default-rtdb.firebaseio.com",
        "projectId": "ezpark-7cf76",
        "storageBucket": "ezpark-7cf76.appspot.com",
        "messagingSenderId": "103330481504",
        "appId": "1:103330481504:web:22007a21929ea345838a05",
        "measurementId": "G-JYWZVBK079"
    }
    firebase = pyrebase.initialize_app(config)
    auth = firebase.auth()
    db = firebase.database()


    # NFC Setup
    signal.signal(signal.SIGINT, end_read)
    GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(27,GPIO.OUT)
MIFAREReader = mfrc522.MFRC522()


# Servo Setup
GPIO.setup(SERVO_PIN, GPIO.OUT)
servo = Servo(SERVO_PIN)
servo.max()


# Display Setup
defaultMessage()


# Proximity Sensor Setup
oProx.begin()
oProx.set_led_current(200)
oProx.set_prox_integration_time(8) # 1 to 8 is valid
startingProxValue=0 # Take 8 readings and average them
for x in range(8):
    startingProxValue += oProx.get_proximity()
startingProxValue /= 8
deltaNeeded = startingProxValue * 0.05 # Look for %5 change
if deltaNeeded < 5:
    deltaNeeded = 5   # set a min value
```

```
    # Threading Led
#      thread = th.Timer(1,standbyBlink)
#      thread.start()

    while continue_reading:

        # Scan for cards
                                                (status,TagType)       =
MIFAREReader.MFRC522_Request(MIFAREReader.PICC_REQIDL)

        # If a card is found
        if status == MIFAREReader.MI_OK:
            print ("Card detected")

        # Get the UID of the card
        (status,uid) = MIFAREReader.MFRC522_Anticoll()

        # If we have the UID, continue
        if status == MIFAREReader.MI_OK:

            # Print UID
                                            print    ("Card    read    UID:
"+str(uid[0])+","+str(uid[1])+","+str(uid[2])+","+str(uid[3])+','+str(uid[
4]))
```

```
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]

            # Select the scanned tag
            MIFAREReader.MFRC522_SelectTag(uid)

            #ENTER Your Card UID here
            my_uid = [195,140,167,29,245]
            exit_uid = [83,75,216,27,219]
            #Configure LED Output Pin

            #Check to see if card UID read matches your card UID
            if uid == my_uid:
                myOLED.begin()
                myOLED.clear(myOLED.PAGE)
                    myOLED.clear(myOLED.ALL)   # clear the display's memory
buffer
                  myOLED.print("                                WELCOME")   #
print the text
                myOLED.display()

db.child("Users").child("RSumL5aFj8P7qaxWTIHw3icKJ2Z2").child("History").c
hild("Entry").push(time_str)
                if gate_open == False:
                    servo.mid()
```

```
        gate_open = True
        gate_close = False
    time.sleep(5)
     # Begin operation loop
    nothingThere = True
    while True:
        proxValue = oProx.get_proximity()


        if proxValue > startingProxValue + deltaNeeded:
            nothingThere = False
            print("   Car within range: %d" % proxValue)
            ledFrequency = 0


        elif not nothingThere:
            nextCarDelay()
            if gate_close == False:
                servo.max()
                gate_close = True
                gate_open = False
            ledFrequency = 0.5
            nothingThere=True
            break
    time.sleep(2)
```

```
                defaultMessage()
                uid = 123
            elif uid == exit_uid:
                                                                payment    =
db.child("Users").child("RSumL5aFj8P7qaxWTIHw3icKJ2Z2").child("Payment").g
et()
                validation = payment.val()
                myOLED.begin()
                myOLED.clear(myOLED.PAGE)
                    myOLED.clear(myOLED.ALL)   # clear the display's memory
buffer

                if validation == True:
                    myOLED.print("                              GOODBYE")   #
print the text
                    myOLED.display()

db.child("Users").child("RSumL5aFj8P7qaxWTIHw3icKJ2Z2").child("History").c
hild("Exit").push(time_str)
                    if gate_open == False:
                        servo.mid()
                        gate_open = True
                        gate_close = False
                    # Begin operation loop
                    nothingThere = True
```

```
            while True:
                proxValue = oProx.get_proximity()


                if proxValue > startingProxValue + deltaNeeded:
                    nothingThere = False
                    print("   Car within range: %d" % proxValue)
                    ledFrequency = 0


                elif not nothingThere:
                    nextCarDelay()
                    if gate_close == False:
                        servo.max()
                        gate_close = True
                        gate_open = False
                    ledFrequency = 0.5
                    nothingThere=True
                    break
            time.sleep(2)
            defaultMessage()
            uid = 123


        else:
```

```
                    myOLED.print("  PLEASE PAY BEFORE         LEAVING")   #
print the text

                    myOLED.display()

                    GPIO.output(LED_PIN_RED, GPIO.HIGH)

                    GPIO.output(LED_PIN_GREEN, GPIO.LOW)

                    time.sleep(5)

                    defaultMessage()

                    uid = 123

                    GPIO.output(LED_PIN_RED, GPIO.LOW)

                    GPIO.output(LED_PIN_GREEN, GPIO.HIGH)

    time.sleep(.4)

    GPIO.cleanup()

if __name__ == '__main__':

    try:

        autoPark()

    except (KeyboardInterrupt, SystemExit) as exErr:

        print("\nSystem Shutdown")

        sys.exit(0)
```

2. Mobile Application Code

https://github.com/AkashSingh8137/ezpark/tree/main/ezpark