Ryerson University
Department of Electrical, Computer & Biomedical Engineering
Faculty of Engineering & Architectural Science

Automated Vehicle Parking Management System

Computer/Electrical Engineering Capstone Design Project

Toronto Metropolitan University, W2024

by

Taskin Abdur-Rahman
Zohraan Badar
Kisoban Rajendran
Yadu Krishnan Madhu

Acknowledgement

We would like to express my sincere gratitude to all those who contributed to the successful completion of this engineering design report. The completion of this project would not have been possible without the support and collaboration of various individuals and organizations.
We extend our appreciation to:

Dr. Alagan Anpalagan: The Faculty Lab Coordinator (FLC) for this project, who is recognized for his guidance and assistance for students throughout their design project.
ECB Department: The Department of Electrical, Computer, and Biomedical Engineering for facilitating the capstone design project experience for students and for their continued support for the learning opportunities available for students.

Furthermore, we want to acknowledge the contributions of my peers and colleagues who provided valuable feedback and assistance at various stages of the project. Lastly, we appreciate the resources and facilities provided by the Faculty of Engineering and Architectural Science.Their collective efforts have significantly enriched the quality of this engineering design report. Thank you all for your support.

Certification of Authorship

"I hereby certify that I/we are the author(s) of this document and that any assistance I/we received in its preparation is fully acknowledged and disclosed in the document I/we have also cited all sources from which I/we obtained data, ideas, or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I/we also certify that this paper was prepared by me/us for this purpose."

	Student 1	Student 2	Student 3	Student 4
Name	Taskin Abdur-Rahman	Zohraan Badar	Kisoban Rajendran	Yadu Krishnan Madhu
Email	navid.rahman@torontomu.ca	z1badar@torontomu.ca	kisoban1.rajendran@torontomu.ca	ymadhu@torontomu.ca
Phone	647-920-6501	647-395-9386	647-939-2069	437-213-9830
Signature	[illegible]	Z.B.	K.R	Y.M

Table of Contents

Abstract

Parking lots, especially in areas like airports, malls, and downtown, often face issues of congestion and limited space due to unorganized traffic flow. This report proposes an Automated Parking Management System (APMS) to address these challenges. The APMS aims to streamline parking processes, enhance user experience, and improve overall efficiency through touch-free automation, real-time monitoring, and predictive analytics.
The system utilizes IoT technology and sensors to automate parking spot occupancy monitoring and authentication processes. Users can book parking spots through a user-friendly web application and receive recommendations based on availability prediction algorithms. Touch-free authentication is facilitated through QR code authentication and automatic license plate recognition. Backend servers manage all activities, including entrance and exit authentication, payment processing, and data collection.
To enhance cost-efficiency, usability, and security, the system leverages advanced software architecture and integrates features like real-time data analysis, pre-booking options, and automated verification. Additionally, alternative authentication methods, such as RFID technology, are explored to offer faster and more reliable authentication.
Performance testing reveals challenges in QR code detection efficiency, prompting consideration for alternative authentication methods or improvements in QR code detection processes. Overall, the APMS presents a promising solution to alleviate parking congestion, improve user experience, and optimize parking lot management.

Introduction & Background

Parking lots can be very crowded and have very limited parking spaces (ex. at Airports, malls, downtown) due to unorganized traffic and pedestrians that might get in the way. Because of this it can cause accidents that may harm pedestrians or other drivers without organizing the traffic of a parking lot. The aim of this project is to design an automated parking management which is touch-free, fully automated, time efficient, user friendly. With the rising shortage of real estate and increasing population the need for efficient parking management systems has also increased. This parking system should reduce traffic on public roads and congestion in parking lots through the use of availability prediction algorithms from collected data. The parking lot is made touch-free with the help of QR code authentication, vehicle detection sensors, and automatic license plate recognition with cameras. A backend server automates all activities in the parking lot, such as entrance authentication, parking space occupancy monitoring, exit authentication, payment, and data collection. The remote backend server controls devices in the parking lot such as cameras, sensors and lights, through IOT devices across a secure network connection. A user-friendly interface web-application allows users to easily register for a parking spot, manage parking spot bookings, and access the parking lot using a generated QR code. This system removes the need for users to pay through a kiosk and place the ticket in their car, making for a time efficient and easy to use system.

Another goal of this design is to improve on the previous solution to this problem. The improvements shall be done with respect to cost efficiency, usability, and security and reliability.

Our design aims to leverage an advanced software design architecture to improve efficiency of cameras and image processing, reducing the cost of the system. Additionally, including automated parking verification in our design strengthens the security system, removing the need for manual parking inspection. Enforcing a pre-booking system removes the need for payment verification, although it restricts the usability of the parking system for users. Our design allows users to enter without a booking, this allows anyone to drop-in and park while maintaining a user-friendly payment system.

Objectives

Some of the features, requirements and performance metrics this project aims to achieve are documented in the section below.

Main high-level features

- ☐ The APMS shall have a parking lot automation subsystem that automates various processes in a parking lot.
- ☐ The parking lot automation subsystem shall be driven by IOT technology based architecture.
- ☐ The parking lot automation subsystem shall make use of sensors to monitor parking occupancy.
- ☐ The parking lot automation subsystem shall be driven by a central server that communicates with a device with Wi-Fi capabilities.
- ☐ The central server shall collect long term parking spot occupancy data to be later analyzed (data analysis).
- ☐ The system shall provide a GUI subsystem that allows users to use the parking lot easily and efficiently.
- ☐ The GUI subsystem shall provide a parking spot booking interface for users.
- ☐ The GUI subsystem shall provide recommendations to the user for parking spots using availability prediction algorithms.

Parking lot automation subsystem

- ☐ Users must book a parking spot to enter the parking lot.
- ☐ Users shall be able to enter only within the time of a valid booking.
- ☐ Users must exit within their booked time.
- ☐ System shall authenticate users with a booking one at a time.

- ☐ Regular parking users shall be able to park at any “regular” spot.
- ☐ Premium parking users shall be able to park at a specific premium spot.
- ☐ Disabled parking users shall be able to park at any disabled spot.
- ☐ System shall have a physical indicator in the parking lot to indicate that a spot is occupied.
- ☐ System shall track the occupancy of parking spots.
- ☐ Vehicle length and width shall not exceed parking spot length and width.

- ☐ System shall use Internet of Things (IOT) technology to implement tracking software for parking spots.
- ☐ System shall perform real-time monitoring of parking spots.
- ☐ System shall use sensors to detect the presence of a vehicle in a parking spot.
- ☐ System shall authenticate users for “special” (premium , disabled, etc.) parking spots.

Data analysis

- ☐ System shall track the real-time parking occupancy in a remote server.
- ☐ System shall collect and store parking occupancy data in a remote database.

- ☐ System shall use prediction algorithms using collected data to predict parking occupancy at certain times at specific locations.
- ☐ System shall track peak hours at specific parking spots.
- ☐ System shall track the most popular parking spots.

Web-Application GUI

- ☐ Users must have a registered account to login to the application.
- ☐ Users shall be able to register an account using the application.
- ☐ Users shall be to login to use the application features and services.
- ☐ Users shall be able to store personal information associated with their account profile.
- ☐ Users shall be able to logout of the application.

- ☐ Users shall be able to book parking spots for available times.
- ☐ Booking information shall be used to authenticate users at the parking lot.
- ☐ Users shall be able to cancel their bookings at any time.
- ☐ Users shall be refunded if booking is canceled prior to the booked time slot.
- ☐ Users are required to pay at the time of booking.

- ☐ Users shall be able to see real-time parking occupancy of spots.
- ☐ Users shall be able to see peak hours at specific spots.
- ☐ Users shall be able to see occupancy rate of each spot.
- ☐ Users shall receive recommended booking times based on predicted availability of parking spots.

Non-functional/Performance requirements

- ☐ The APMS parking lot automation system shall be time efficient compared to modern traditional methods.
- ☐ The APMS parking lot payment system shall be more user friendly and secure than traditional payment methods.
- ☐ The APMS shall automate authorization and security to eliminate the need for a security team.
- ☐ The APMS shall reduce traffic congestion and improve process efficiency using availability prediction algorithms.

- ☐ The APMS shall provide users a touch-free experience at the parking lot.
- ☐ The APMS shall provide a user-friendly and effortless experience.

Theory and Design

Parking Lot Authentication

QR-Codes are used to authenticate users entering the parking-lot through the entrance. User shows the QR-Code to the camera at the gate. The camera reads the QR-code and asks the system to validate the QR-code and open the gate. A unique QR-Code is generated for each user at each unique parking lot.

Components

1. SG90 Servo (x2)
2. Wooden popsicle sticks (x2)
3. SG90 Servo Arm + screws
4. ESP32-WROOM-32 (x1)
5. ESP32-CAM (x2)

Need a picture of ESP32CAM here + gates

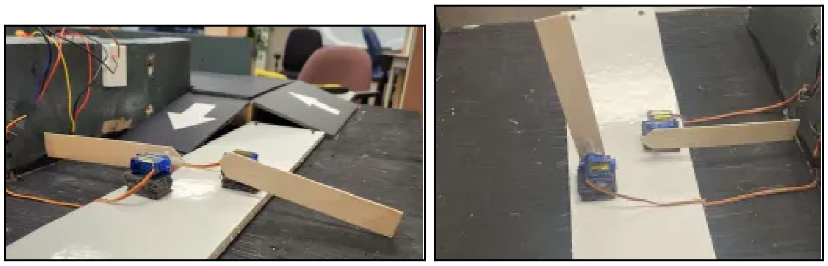
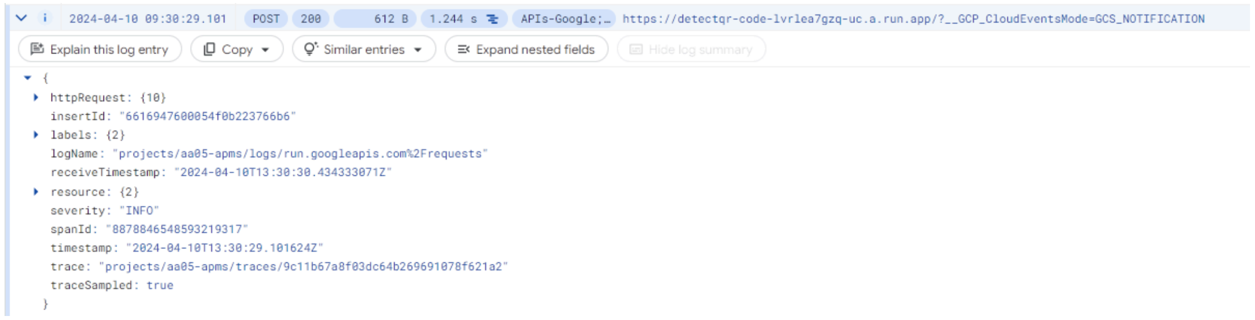


Figure 1: Gate operation at the entrance.

How does it work?

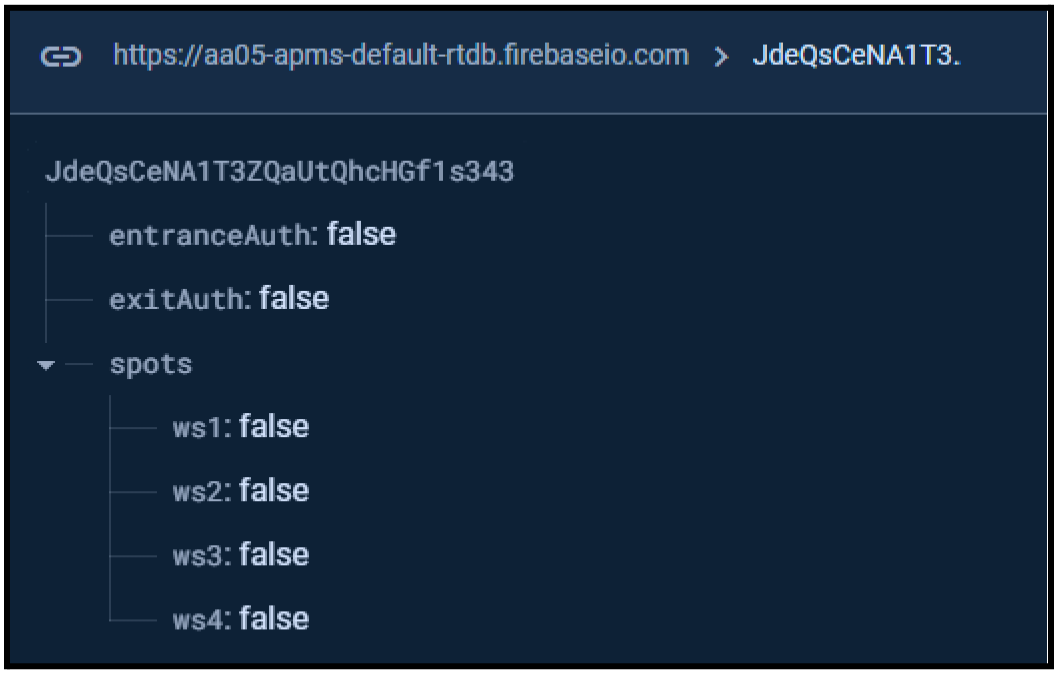
1. A ESP32-CAM microcontroller is used to constantly upload images to the backend database.



```
2024-04-10 09:30:29.101  POST  200  612 B  1.244 s  APIs-Google;…  https://detectqr-code-lvrlea7gzq-uc.a.run.app/?__GCP_CloudEventsMode=GCS_NOTIFICATION
{
  httpRequest: {10}
  insertId: "6616947600054f0b223766b6"
  labels: {2}
  logName: "projects/aa05-apms/logs/run.googleapis.com%2Frequests"
  receiveTimestamp: "2024-04-10T13:30:30.434333071Z"
  resource: {2}
  severity: "INFO"
  spanId: "8878846548593219317"
  timestamp: "2024-04-10T13:30:29.101624Z"
  trace: "projects/aa05-apms/traces/9c11b67a8f03dc64b269691078f621a2"
  traceSampled: true
}
```

Figure 2: Google Cloud log of cloud function being triggered

2. User arrives at the entrance or exit gate, then holds their authentication QR-Code up to the entrance/exit camera view.
3. Image being uploaded triggers a cloud event which runs a function that reads the newly uploaded image and attempts to authenticate and open the gate. (Refer to Appendix section aa05-apms-frontend→functions).
4. If QR-Code is valid, the cloud function sets database value to true.



```
https://aa05-apms-default-rtdb.firebaseio.com > JdeQsCeNA1T3.

JdeQsCeNA1T3ZQaUtQhcHGf1s343
├── entranceAuth: false
├── exitAuth: false
└── spots
    ├── ws1: false
    ├── ws2: false
    ├── ws3: false
    └── ws4: false
```

Figure 3: Firebase real-time database to control parking-lot gates.

5. Gate opens by periodically reading the boolean value of the database. If true it opens the gate, waits for a set delay, closes the gate. Then sets the database value back to false (Refer to Appendix section aa05-parking-lot).
6. Users must show the QR-Code to the camera and hold it for a moment while the camera uploads the image to the database.

Quick Response (QR)-Code

A QR code (Quick Response code) is a two-dimensional barcode that can store various types of data, such as URLs, text, numbers, and even binary data. These little square patterns have become ubiquitous, appearing on product packaging, advertisements, and event tickets.

1. **Data Encoding**

   A QR code encodes information in a grid of black squares on a white background. The arrangement of these black squares (called data modules) represents the data to be stored. Numeric and alphanumeric characters, bytes, and even kanji characters convert into this unique arrangement.

2. **Scanning and Decoding**

   When you scan a QR code using a QR code scanner (usually your smartphone camera), it captures the code's pattern. The scanner then decodes the arrangement of squares back into human-readable data. In seconds, the code translates from visual elements to its original form (e.g., a URL or a piece of text).

3. **Special Features of QR Codes**

   QR codes can hold a significant amount of data compared to traditional barcodes. They are read quickly by scanners and cameras. Virtually all smartphones can instantly and easily scan QR codes.

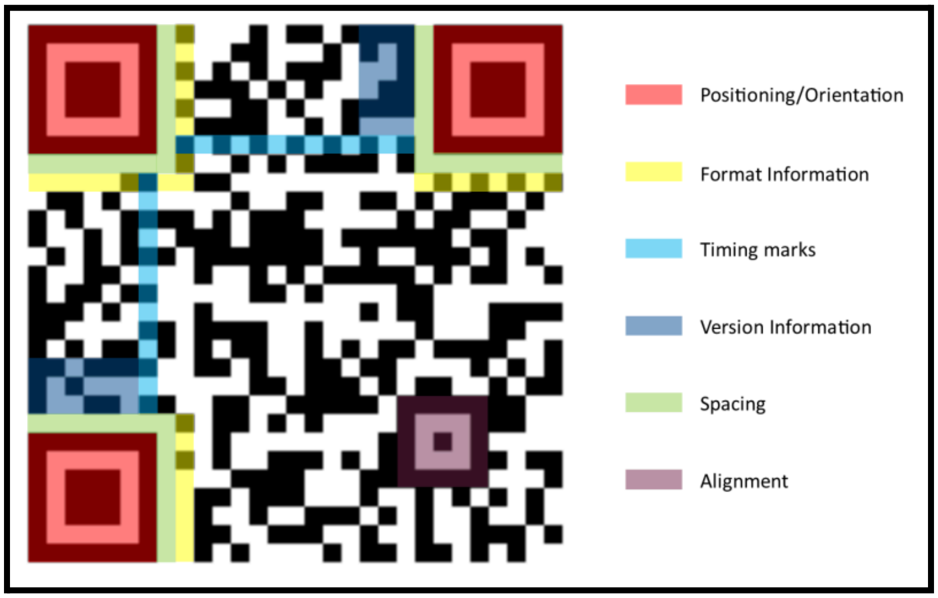


Figure 4: QR-Code components highlighted.

Advantages and disadvantages

Touchless

This system allows users to enter and use the parking lot through the use of their smartphones without having to interact with a parking machine or use a physical RFID device. Over time, with less user interaction the system requires less maintenance. A touchless system is also more hygenic.

Cost Effectiveness

This system is cost effective as it does not require use of payment machines and or RFID authentication systems. It only uses QR-Code reader software libraries which perform computer vision functions on images.

Reliance on WiFi connectivity

This system requires a robust and reliable Wifi connection to be able to perform in day to day usage of the parking lot. Having a loss of packets can result in gate opening delays or failure to authenticate users.

Parking Presence Detection

A weight sensor apparatus is used to detect the weight of a vehicle and notify the system of a car's presence. Each parking spot is equipped with a weight sensor, when a car is detected a light turns on indicating that the spot is occupied, and the database is updated with the real-time spot occupation data.

Components

1. Weight sensor apparatus (x4) (Figure 5)
   a. 1kg strain gauge load cell (x1)
   b. 3D Printed rectangular platform with rectangular platform with dimensions (120mm x 70mm) [base platform + 20mm each side] (x2)
   c. HX711 module (x1)
2. ESP32-WROOM-32 (x1)

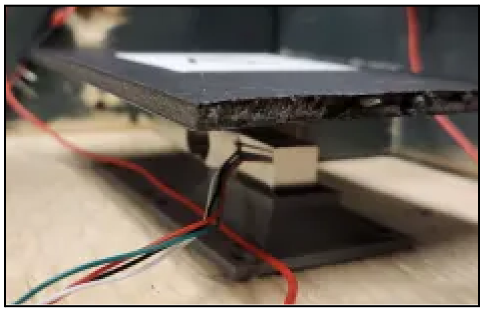


Figure 5: Weight sensor apparatus for parking spots.

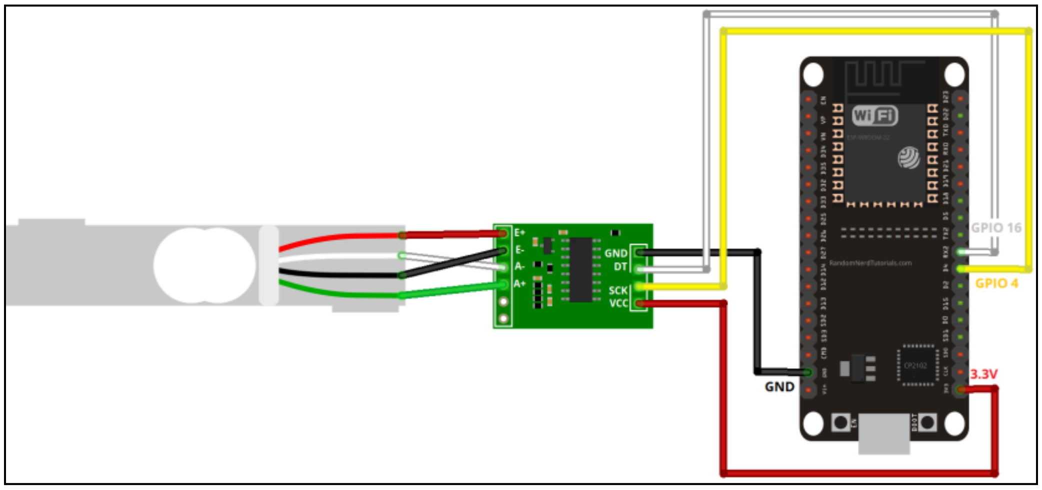


Figure 6: Load Cell strain gauge wiring diagram.

How does it work?

1. A weight sensor apparatus is placed under each parking spot in the parking lot.
2. When a car parks on the weight sensor platform the weight detected by the HX711 module is provided to ESP-32.
3. The ESP32 code then reads the digital integer values coming from the HX711 module and compares it to a static integer threshold.
4. If the threshold is exceeded, The ESP32 will:
    a. Send a HTTP request to the database to update the occupation value of the parking spot to *true*.
    b. Set the LED associated with the parking spot to *HIGH*.

Strain-Gauge Load Cell

Load cells are sensors that convert force (weight) into an electrical signal. These load cells consist of a metal bar with attached strain gauges. A strain gauge is an electrical sensor that measures force or strain on an object. When an external force is applied (e.g., an object's weight), the strain gauge's resistance changes due to deformation of the metal bar. The strain gauge resistance varies proportionally to the load applied, allowing us to calculate weight. The strain gauge consists of a conductive metal wire that when stretched, will become longer and skinnier resulting in an increase of electrical resistance. Conversely, if it is placed under compressive force it will broaden and shorten decreasing the electrical resistance. The diagram below shows the diagram of the weight sensor apparatus, along with the strain strain gauge circuit. The HX711 module converts analog signals into digital data for microcontrollers to use to recognize values coming from the strain gauge.

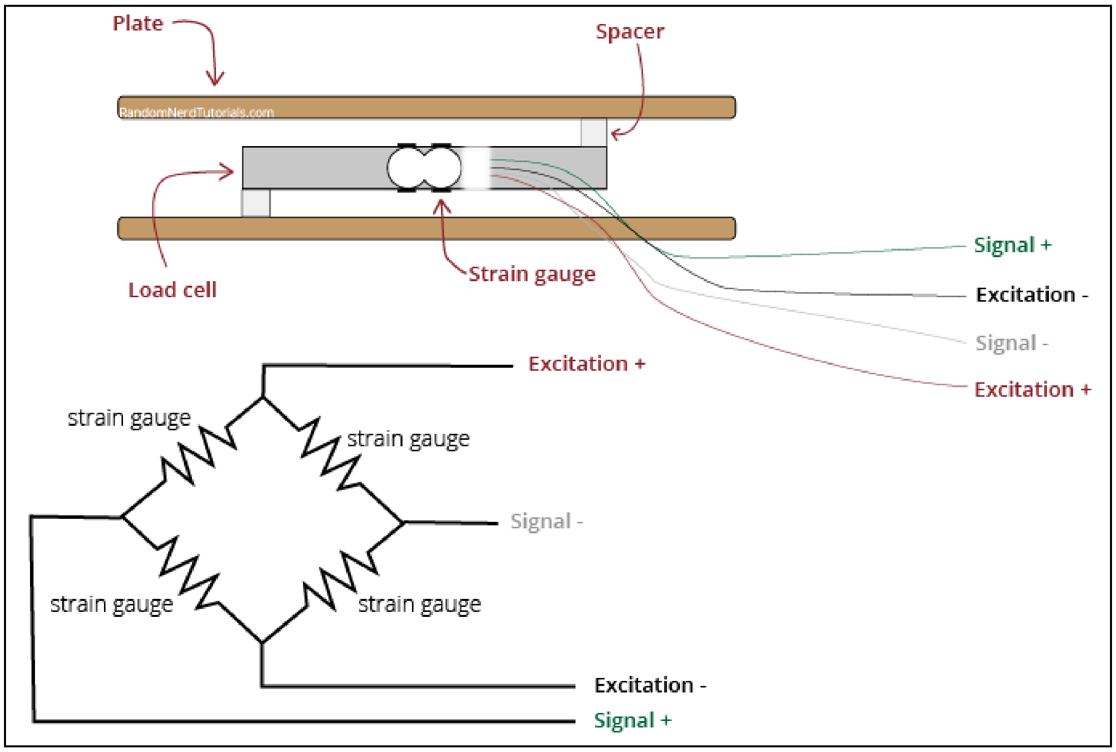


Figure 7: Strain Gauge Load Cell diagram.

Advantages and disadvantages

Reliability and Accuracy

By detecting the weight of parking spots we can have consistent and reliable results. The weight sensors measure weight accurately preventing over-weight/under weight objects from using parking spots.

Reliance on temperature

Weight sensors can become inaccurate due to temperature, for accurate results the temperature needs to be maintained in the parking lot. Which adds several restrictions and can be an undesirable choice for a parking lot.

Cost

This strategy increases the cost of the system as installing and maintaining weight sensors into a parking lot can be very expensive. Maintaining a certain temperature is also necessary for accurate results and this can also increase the overall cost.

Firebase (BaaS)

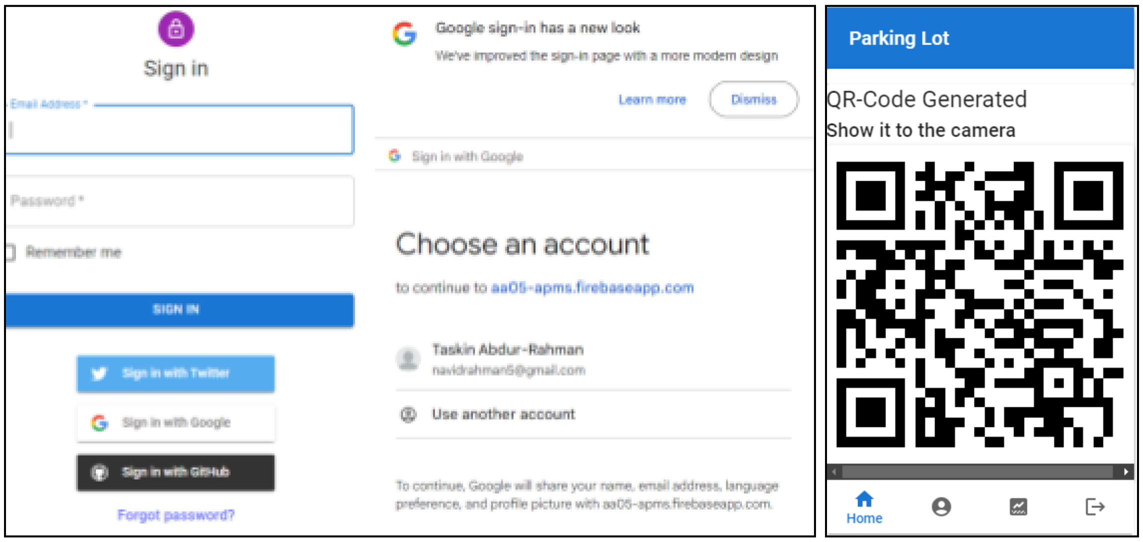
Firebase is a cloud service which falls under the category of Backend as a Service (BaaS) as it provides a range of services that is dedicated to backend systems involving, Firestore (large scalable database), Real-time Database (small quick-read-write database for real-time data), Authentication (Users database), Functions (Web API functions responding to http requests that run on cloud), Storage (file and data storage on cloud).

Authentication
An end-to-end user identity solution in under 10 lines of code
Cloud Firestore
Realtime updates, powerful queries, and automatic scaling
Realtime Database
Store and sync data in realtime
Storage
Store & retrieve user generated content
Functions
Extend and connect Firebase features
Hosting
Deploy web apps in seconds

Figure 8: Backend services provided by Firebase.

How does it work?

1. User accounts are created and managed using Firebase Authentication. An account is created when users sign up or login to the web-application through a third party (Google, Facebook, Github).



Google sign-in has a new look

We've improved the sign-in page with a more modern design

to continue to aa05-apms.firebaseapp.com

To continue, Google will share your name, email address, language preference, and profile picture with aa05-apms.firebaseapp.com.

Figure 9: User authentication and QR-Code generation process.

2. To use the parking lot users must either book a parking spot or request a drop-in parking QR-Code. Once the QR-Code is generated users can present it to the camera at the gate.

3. The ESP32-CAM shall capture an image of the QR-Code and upload it to Firebase Storage, which triggers a cloud function that detects QR-Code images and attempts to decode and authenticate users using a specific decoding and verification algorithm.

4. If the QR-Code is valid, the cloud function sets the value of *entranceAuth* or *exitAuth* in the Firebase Real-time Database to *true*.

5. Finally the ESP32-WROOM-32 detects the change in value in the Real-time Database and opens the gate by sending a voltage signal through the servo pin.

Advantages and disadvantages

Cost effectiveness

Using Firebase as a backend removes the need for in-house database and server solutions, which would be costly for scalability. The human resources needed for designing, developing and maintaining in-house solutions can be unnecessarily expensive.

Reliability

Outsourcing to an expert, and it reduces the complexity of the automated parking system. Outsourcing the backend job to an expert, trusted third party allows for reliable performance.

Reduced Complexity

Removing the need for in-house solutions reduces the complexity of a system, which requires less experts for backend solutions. Firebase is also designed for ease of use compared to other backend services.

Alternative Designs

RFID Authentication

A different approach to authentication at the gate is by the use of RFID technology. Instead of a camera reading the QR-Code of a user at the gate, an RFID reader is stationed at the gate and reads the UID of the tag. If the UID matches the one stored in the Firestore database for any of the users from the list of users allowed for the given parking lot, then the gate shall be opened.

The following example shows how the UID from an RFID tag can be read with an ESP32 + RFIDRC522 reader. In Figure 10 the circuit is shown and in Figure 11 the serial monitor from the Arduino IDE shows the UID that was read.

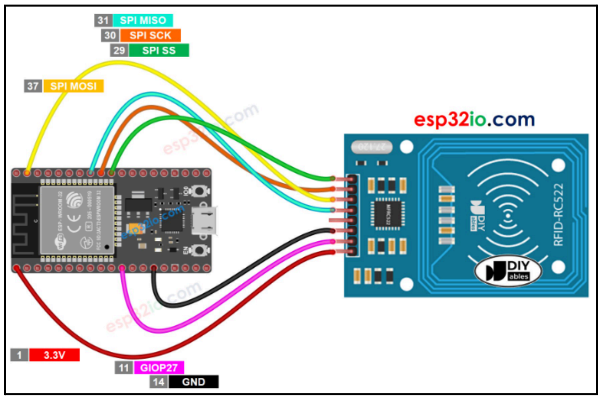
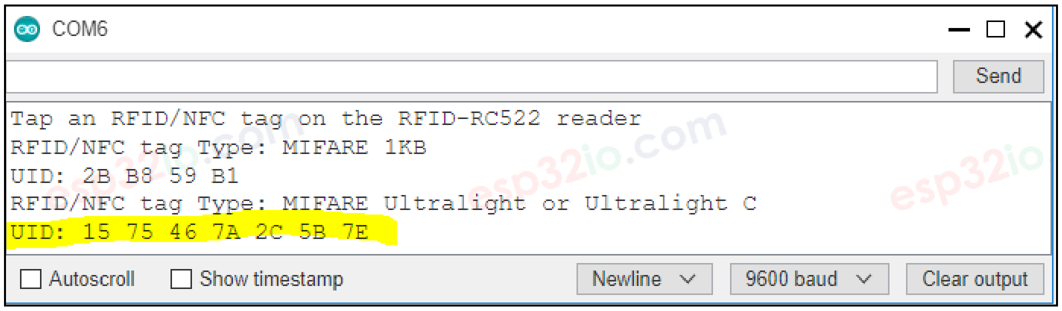


Figure 10: ESP32-WROOM-32 + RFIDRC522 Reader circuit diagram [8].



```
Tap an RFID/NFC tag on the RFID-RC522 reader
RFID/NFC tag Type: MIFARE 1KB
UID: 2B B8 59 B1
RFID/NFC tag Type: MIFARE Ultralight or Ultralight C
UID: 15 75 46 7A 2C 5B 7E
```

Figure 11: Arduino IDE Serial Monitor output from ESP32 [8].

Radio-Frequency Identification (RFID) Technology

This system uses RFID technology to authenticate users using passive RFID. Passive RFID tags are devices with embedded circuits that do not have an internal power source. The circuit has an induced current from the reading antenna of the RFID reader. The RFID reader constantly emits an electromagnetic wave, which induces a current in any nearby RFID tag. When a current is induced in an RFID tag it emits its own unique electromagnetic wave radio-frequency signal which the RFID reader can detect with its own receiving antenna [].

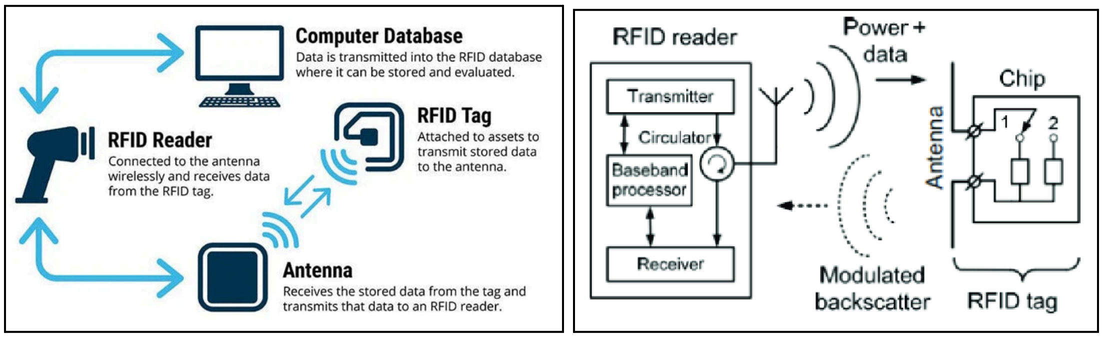

Computer Database
Data is transmitted into the RFID database where it can be stored and evaluated.
RFID Tag
Attached to assets to transmit stored data to the antenna.
RFID Reader
Connected to the antenna wirelessly and receives data from the RFID tag.
Antenna
Receives the stored data from the tag and transmits that data to an RFID reader.
RFID reader
Transmitter
Circulator
Baseband processor
Receiver
Power + data
Modulated backscatter
Antenna
Chip
1
2
RFID tag

Figure 12: High-level and low-level overview of passive RFID technology.

Advantages and disadvantages

Speed

This system of authentication is quick and easy. Radio-frequency identification is faster as computer vision requires computationally heavy algorithms. It is also less prone to error as lighting and motion blur can hinder the authentication process of QR-Codes.

Reliability

Overall this system is more reliable than QR-Code detection since the decoding of the code is being done locally with the RFIDRC522 module and the ESP32, whereas the ESP32-CAM is constantly uploading images to the database which can be computationally heavy and time consuming. It relies on higher bandwidth on the Wifi connection which makes it less reliable. If something obstructs the view of the camera it is no longer able to authenticate the users at the gate making it rely heavily on having a clear view.

Ease of Access

While on one hand the QR-Code detection allows users to authenticate themselves through the use of a smartphone, requiring very little overhead and additional parts, the RFID tags require unique identification tags to be produced for each user. This can add more complexity to the system, as there needs to be a system for creating unique tags for a new user and a tag replacement process.

Infrared Sensor (IR) Sensor

This design attempts to provide a solution to the detection of a vehicle presence at a parking spot, or a design to track parking spot occupation. The Infrared Object Avoidance Sensor constantly emits infrared radiation, if an object gets close enough IR light reflects off of it which is then picked up by a neighboring photodiode as seen in the diagram below (Figure 13)

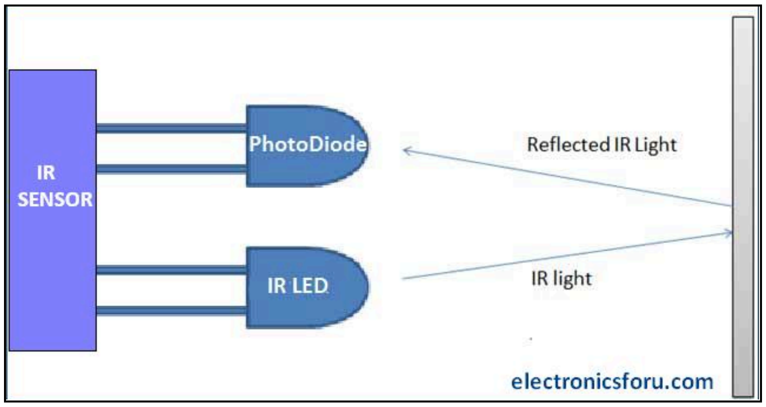


Figure 13: High-level diagram of IR sensor.

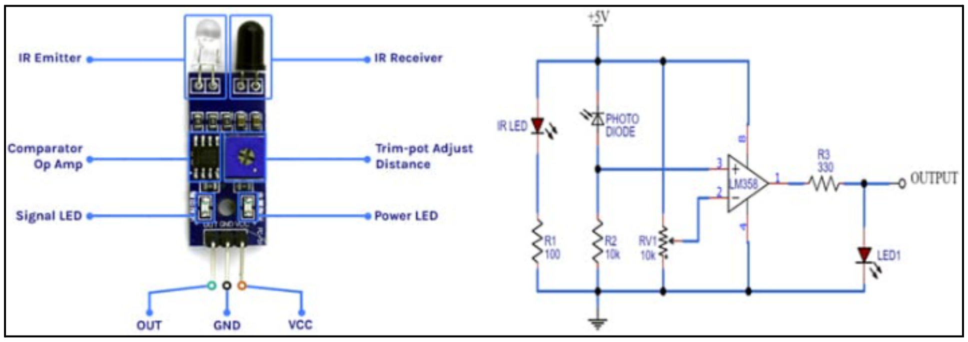


Figure 14: IR sensor low-level component diagram.

Advantages and disadvantages

Efficient

The IR sensor is efficient at detecting the presence of a vehicle in the parking lot. It is fast, responsive and inexpensive. It requires little maintenance and installation resources.

Reliable

It is reliable as it does not require much calibration and maintenance.

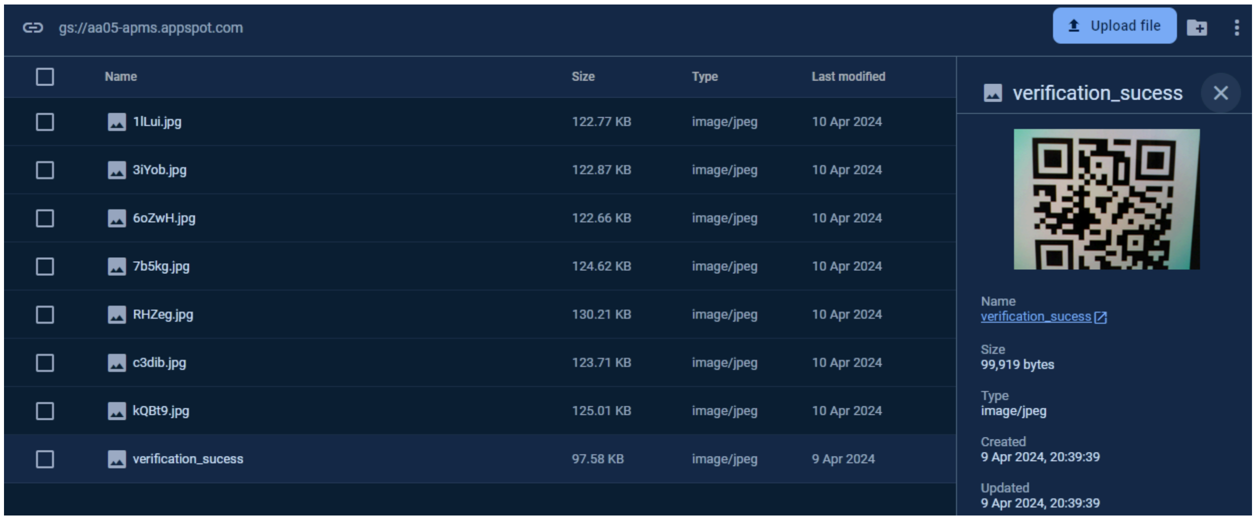
Inaccurate

The IR sensor will detect any object in its vicinity. There may need to be a system in place to detect if the object is a momentary disturbance or a vehicle that is parked for a longer period of time.

QR-Code Detection

Design 1: Firebase Storage upload

One of the primary designs that were considered include ESPCAM capturing images indefinitely and uploading to the Firebase Storage. This is a simple design with a straightforward approach which remains secure and well integrated with the application. This approach however includes latency issues as it relies on upload latency and cloud function trigger latency to add on top. Due to the unreliable nature of internet connectivity, cloud function triggers tend to be unstable under stress as a result.



gs://aa05-apms.appspot.com

Name	Size	Type	Last modified
1lLui.jpg	122.77 KB	image/jpeg	10 Apr 2024
3iYob.jpg	122.87 KB	image/jpeg	10 Apr 2024
6oZwH.jpg	122.66 KB	image/jpeg	10 Apr 2024
7b5kg.jpg	124.62 KB	image/jpeg	10 Apr 2024
RHZeg.jpg	130.21 KB	image/jpeg	10 Apr 2024
c3dib.jpg	123.71 KB	image/jpeg	10 Apr 2024
kQBt9.jpg	125.01 KB	image/jpeg	10 Apr 2024
verification_sucess	97.58 KB	image/jpeg	9 Apr 2024

verification_sucess

Name: verification_sucess
Size: 99,919 bytes
Type: image/jpeg
Created: 9 Apr 2024, 20:39:39
Updated: 9 Apr 2024, 20:39:39

Figure 15: Image of a QR-Code which was successfully decoded based on Design 1.

Design 2: Image Capture Server

Another design that was discussed has the ESPCAM serve as an Image capture Server where it serves as an endpoint for an image capture. Although Design 2 involves several layers of added complexity to achieve the end goal, being QR-Code decoding, on-site implementation shows greater stability and more promising results than Design 1. Design 2 however, relies on a local network.

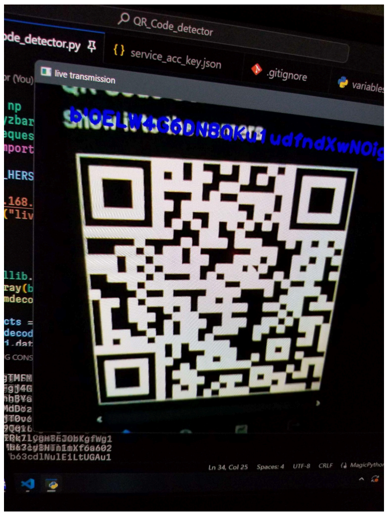


Figure 16: A successful capture and decode based on Design 2

Design 1 offers an elegant solution, one that fits in like a perfect puzzle piece if the application were to be a jigsaw puzzle. However, on-site implementation of Design 1 proves to be much of a challenge. The overall process of capturing, uploading and decoding takes up a significant amount of time resulting in a poorer user experience. There are neither on-site phone-position indicators (QR-Code that is to be scanned is generated on the web app through the phone) nor led indicators to indicate users whether the image capture was successful or not, other than the apparent app notification. Hence, a bad image capture will have to wait for the upload and decode latency which can be significant. Users will have to try until the QR-Code is captured in a way where the cloud function is able to decode it, and verify. Design 2 offers an alternative that tackles this key issue. It requires a local device, one preferable with a screen (such as laptops) where users are able to view the ESP-CAM's point of view. The user is then able to hold their phone appropriately for the CAM to capture the QR-Code. An onscreen confirmation is also visible within the screen when the QR-Code is successfully captured. This solution however lacks much of the modularity that Design 1 offers, hence a combination of both solutions will provide the optimal outcome.

Material/Component list (New)

Table 1.00 - List of materials and components

Description	Part #	Quantity	Unit Cost	Subtotal	Total
2PC ESP32-CAM-MB	2_XF01022_US	2	34.99	34.99	39.54
ELEGOO 3pcs Breadboard 830 Point Solderless Prototype PCB Board Kit for Arduino Proto Shield Distribution Connecting Blocks	BB830	1	16.99	16.99	19.20
Freenove Breakout Board for ESP32 / ESP32-S3 WROVER WROOM, Terminal Block Shield with Pin Header, 5V 3.3V Power Outputs, GPIO Status LED	FNK0091	2	35.9	35.9	40.56
1KG Digital Load Cell Weight Sensor + HX711 Weighing Sensors Ad Module for Arduino 1kg 5kg 10kg 20kg	N/A	6	3.32	19.92	22.51
5mm Red LED – Super Bright – 624nm	C503B-RCN-CW0Z0AA1	4	In-house	In-house	In-house
Mini Breadboard - White	C503B-RCN-CW0Z0AA1	1	4.00	4.00	4.52
22 AWG Wire 100 ft	N/A	1	In-house	In-house	In-house
14mm M4 Screw (4 pack)	YSCRE-414144	1	1.50	1.50	1.67

5V 1A Switching Power Supply	ADADC-400505	1	12.00	12.00	13.56
Basic Screwdriver Set (6pcs)	TOOSC-322027	1	4.50	4.50	5.08
3PC Screw Terminal Block	TERMB-008235	6	12.00	12.00	13.56
2.1mm DC Barrel (F) to Terminal block	ADADC-010288	1	2.5	2.5	2.8
Solid Rosin Flux	SOLDF-511001	1	3.50	3.50	3.95
Leadfree Solder - 100g	SOLDR-009325	1	13.00	13.00	14.69
M5 Screw	N/A	4	In-house	In-house	In-house
White Screw Terminal Block Dual Row Electric Barrier Block 12-Position Terminal Strip, 10 Amp 380 V	N/A	2	3.29	6.58	7.43
Black electrical tape	N/A	1	In-house	In-house	In-house

Measurement and Testing Procedures (New)

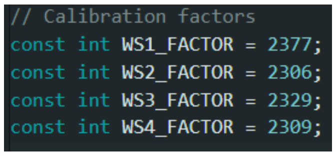
Weight Sensor Calibration Procedure

To prepare the weight sensors for detecting the presence of vehicles in the parking lot, their weight measurements must be calibrated using a known weight. For each load cell calibration must be performed in order to calculate its calibration factor, which is then used in the weight measurement calculation in the code. The steps are as follows:

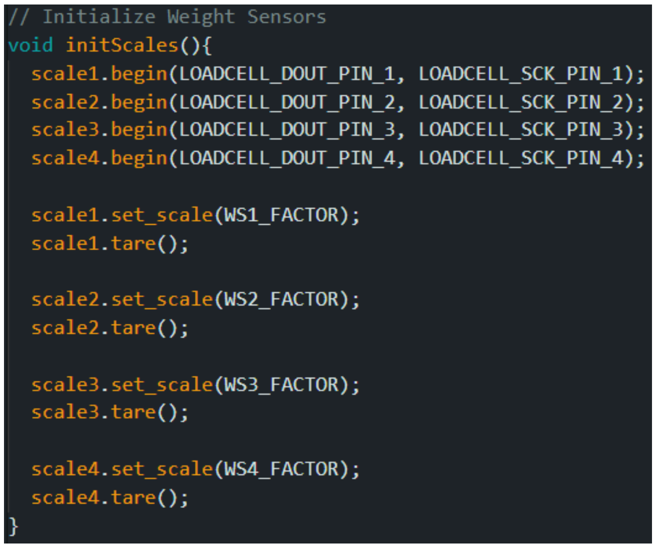
1. Find an object with a known weight that is less than 1kg and is small enough to sit on the platform.
2. For each weight sensor, run the LoadCellCalibration.ino file in Arduino IDE with the serial monitor enabled. Ensure that the wiring is set up correctly before running the program.
3. Note down the readings when no weight is measured for 6-10 readings.
4. Note down the weight readings for your object for 6-10 readings.
5. Find the average of the values for no weight and for your object.
6. Perform the following calculation to obtain the *calibration factor*:

$$Calibration\ Factor = (ObjectWeight_{avg} - NoWeight_{avg})/Actual\ Weight\ of\ Object$$

7. In Figure 17 below the calculated calibration factor is used in the weight measurement code.



```
// Calibration factors
const int WS1_FACTOR = 2377;
const int WS2_FACTOR = 2306;
const int WS3_FACTOR = 2329;
const int WS4_FACTOR = 2309;
```



```
// Initialize Weight Sensors
void initScales(){
  scale1.begin(LOADCELL_DOUT_PIN_1, LOADCELL_SCK_PIN_1);
  scale2.begin(LOADCELL_DOUT_PIN_2, LOADCELL_SCK_PIN_2);
  scale3.begin(LOADCELL_DOUT_PIN_3, LOADCELL_SCK_PIN_3);
  scale4.begin(LOADCELL_DOUT_PIN_4, LOADCELL_SCK_PIN_4);

  scale1.set_scale(WS1_FACTOR);
  scale1.tare();

  scale2.set_scale(WS2_FACTOR);
  scale2.tare();

  scale3.set_scale(WS3_FACTOR);
  scale3.tare();

  scale4.set_scale(WS4_FACTOR);
  scale4.tare();
}
```

Figure 17: Calibration factor usage in ParkingLot.ino

Performance Measurement Results

QR-Code Reading

The capabilities of the QR-Code detection functions were measured by running the functions on test images of QR-Codes. The first category of images were digitally generated images that are clear and big. The second category of images were real-time images captured through an ESP32-CAM, and the third category was real-time use case performance. To help us analyze the results we implemented logic to save images that were decoded successfully in Firebase Storage.

The first category

The images digitally generated were clear and easy to decode; there were no issues with reading the QR-Code in this category. The success rate was 100%.

The second category

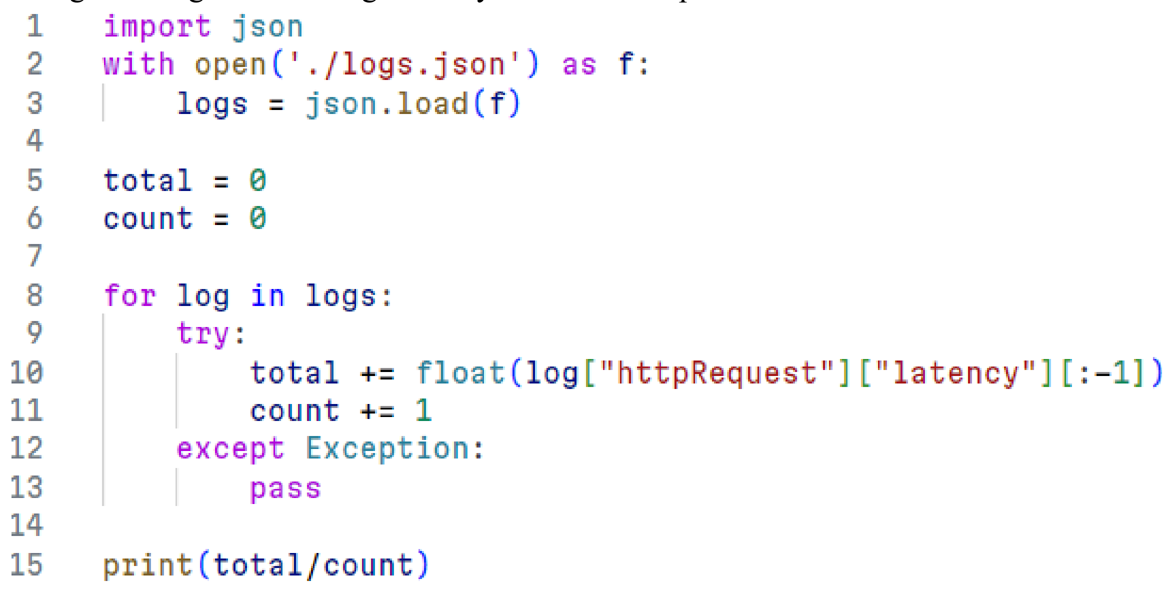
The images captured through the ESP32-CAM were not very successful with some of the images failing to be read by the Firebase Cloud Functions.

The third category

These images were captured and tested continuously in real-time simulating a use-case at the gate entrance. Most of the images failed to be read as the functions require them to be clear and undistorted. In a real use case, the camera uploads images very slowly relative to what is expected for satisfactory results.

QR-Code Design 1

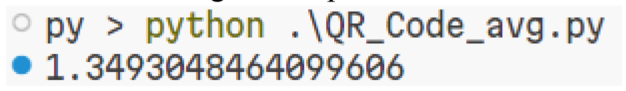
Design 1 which involves capturing the image, uploading to Firebase storage and cloud function decoding the image. The average latency for the whole process is around 1.35 seconds.



```python
import json
with open('./logs.json') as f:
    logs = json.load(f)

total = 0
count = 0

for log in logs:
    try:
        total += float(log["httpRequest"]["latency"][:-1])
        count += 1
    except Exception:
        pass

print(total/count)
```

Figure 18: A small script computing the average latency based on the logs.

Result of running the script:



```
○ py > python .\QR_Code_avg.py
● 1.3493048464099606
```

1.35 seconds is a significant amount of time to take for the whole process.

QR-Code Design 2 Analysis

Design 2 is more difficult to empirically analyse the performance in terms of latency. Hence no additional performance metrics were computed as part of the analysis.

Analysis of Performance

QR-Code Reading

From the results of the performance it was concluded that the strategy or approach to this problem would not lead to desirable performance. On balance of advantages and disadvantages it is not reasonable to constantly upload images to Firebase Storage at the fastest rate to detect the presence of QR-Code in the view of the camera. This is because it is inefficient and relies heavily on strong bandwidth and Wifi connection. It may also be beyond the capabilities of the ESP32-CAM to constantly upload images rapidly over a longer period of time.

Another strategy that should have better performance is by use of a local server that accesses the camera feed of the ESP32-CAM using http get requests. Another approach is by using the cloud function to access the ESP32-CAM's video feed, although this requires setting up a static ip which needs to be exposed through port-forwarding. This approach is very difficult and may not be possible due to security policies in place at the school.

The alternative approach is to abandon computer vision and QR-Codes for authentication, and instead to use RFID authentication technology.

Conclusion

In summary, this report includes the design for all the high-level system requirements for the APMS. The design fulfills the functional requirements for each of the subsystems, as well as the performance requirements. This solution is more reliable and user-friendly than alternative designs as it leverages machine learning to recognize patterns within a select area, providing automated security inspection on multiple spots at the same time. Using license plate verification allows for a more secure payment system, as users can be reported to city authorities if they fail to pay. Other designs limit users to pre-booked access to the parking lot, our design allows users to drop-in and pay in regular parking, reserving special spots for booking only.

Some issues that have not been addressed is the business model for the parking system, describing what payment options are available and at which times. Another issue being the need for sensors in special parking spaces, as it still needs to be discussed if camera verification is sufficient for monitoring special parking spaces. The main difficulty of this project is the lack of progress in terms of low-level system design involving detailed schematics, parts, measurements, software flowcharts and architectures. This is because more time was spent perfecting high-level design decisions as it would later impact lower level design.

The next steps would be to prepare an implementation plan and prepare all necessary technical design documents. This would involve the completion of all low-level drawings, schematics, parts and specifications list and an execution plan for the parking lot subsystem. For the software, the diagrams and flowcharts describing the Web-application api, parking lot functions and data-collection functions need to be created. These steps refer to the first phase of the Implementation Plan, which if done correctly should ensure the completion of the project.

References

[1] *The Collaborative Interface Design Tool*. Figma. (n.d.). https://www.figma.com/

[2] S. Gören, D. F. Óncevarlk, K. D. Yldz and T. Z. Hakyemez, "On-Street Parking Spot Detection for Smart Cities," 2019 IEEE International Smart Cities Conference (ISC2), Casablanca, Morocco, 2019, pp. 292-295, doi: 10.1109/ISC246665.2019.9071760.

[3] Keras, "Keras: The Python Deep Learning library," [online] Available: https://keras.io/

[4] Google. (n.d.). Google. https://firebase.google.com/

[5] React. (n.d.). https://react.dev/

[6] Tiwari, Sumit. (2016). An Introduction to QR Code Technology. 39-44. 10.1109/ICIT.2016.021

[7] Kuphaldt, T. R. (2015, February 12). *Strain gauges: Electrical Instrumentation Signals: Electronics textbook*. All About Circuits. https://www.allaboutcircuits.com/textbook/direct-current/chpt-9/strain-gauges/

[8] *ESP32 - RFID/NFC*. ESP32 Tutorial. (n.d.). https://esp32io.com/tutorials/esp32-rfid-nfc
[9] Smiley, S. (2022, May 13). *What is passive RFID?*. atlasRFIDstore. https://www.atlasrfidstore.com/rfid-insider/what-is-passive-rfid/
[10] CD-Tonics For You. https://www.electronicsforu.com/technology-trends/learn-electronics/ir-led-infrared-sensor-basics

Appendices

aa05-apms-frontend

functions

```
// Listens for any stuff that gets added to storage and then runs the call back
exports.detectQR_code = onObjectFinalized(async (event) => {
  const bucket = admin.storage().bucket(event.data.bucket);
  const file = bucket.file(event.data.name);
  let qrDetected = false;

  if (event.data.contentType.startsWith("image/")) { // Checking if whatevr was
added was an image
    const tempFilePath = `/tmp/${event.data.name}`;
    await file.download({ destination: tempFilePath });
    const image = await Jimp.read(tempFilePath);
          const  qrCodeValue  =  jsQR(image.bitmap.data,  image.bitmap.width,
image.bitmap.height); // qr code obj


    if (qrCodeValue !== null) {

      let verify = false;
      qrDetected = true;

      const decoded_qr_code = unshuffle(qrCodeValue.data);
      const users = await admin_app.auth().listUsers(1000);

      for (let user of users.users) {
        if (decoded_qr_code.includes(user.uid)) verify = true;
      }

      if (verify) {
        const qrCodeDataRef = db.ref(`${secret_key}/entranceAuth`);

        // Sets the entrance auth to true
        await qrCodeDataRef.set(true)
        .catch(e => {
          logger.error(e);
        })

      }

    } else {
      console.log('No QR code found in the image.');
    }

  } else {
    logger.log("Object is not an image.");
  }

  if (!qrDetected) {   // Trying to delete images that were not detected as qr
codes
    try {
      await file.delete();
    } catch (error) {
      logger.log(error);
    }
  }

});
```

aa05-apms-parking-lot

ParkingLot.ino

```
#include <Arduino.h>

/******************** Firebase RTDB Imports ********************/
#include <WiFi.h>
#include <Firebase_ESP_Client.h>

// Provide the token generation process info.
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"

/******************** Weight Sensor Imports ********************/
#include "HX711.h"
#include "soc/rtc.h"
```

```
/******************** Servo Imports ********************/
#include <ESP32Servo.h>

/******************** Firebase RTDB Definitions ********************/
// Insert your network credentials
#define WIFI_SSID "Taskin-Hotspot"
#define WIFI_PASSWORD "dbmw6047"

// Insert Firebase project API Key
#define API_KEY "AIzaSyC129F4OIPEJUjGMbrEu9rUYMeKoo_zBY8"

// Insert RTDB URLefine the RTDB URL */
#define DATABASE_URL "https://aa05-apms-default-rtdb.firebaseio.com/"

// Insert Authorized Email and Corresponding Password
#define USER_EMAIL "navidrahman5@gmail.com"
#define USER_PASSWORD "vrselbhdbbfclsro"

/******************** Firebase variables ********************/
// Define Firebase objects
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

String uid;

unsigned long sendDataPrevMillis = 0;
unsigned long timerDelay = 15000;

/******************** Servo variables ********************/
static Servo entranceGate;  // Servo object
static Servo exitGate;  // Servo object

const int entrancePin = 13;
const int exitPin = 27;

bool entranceAuth = false;
bool exitAuth = false;
```

```
String entrancePath;
String exitPath;

const int maxAngle = 110;

/******************** Weight Sensor variables ********************/
// WS1 Wiring
const int LOADCELL_DOUT_PIN_1 = 16;
const int LOADCELL_SCK_PIN_1 = 4;
// WS2 Wiring
const int LOADCELL_DOUT_PIN_2 = 17;
const int LOADCELL_SCK_PIN_2 = 5;
// WS3 Wiring
const int LOADCELL_DOUT_PIN_3 = 15;
const int LOADCELL_SCK_PIN_3 = 2;
// WS4 Wiring
const int LOADCELL_DOUT_PIN_4 = 14;
const int LOADCELL_SCK_PIN_4 = 12;

// LED Boards
const int LED_1 = 25;
const int LED_2 = 26;
const int LED_3 = 0;
const int LED_4 = 21;

// Calibration factors
const int WS1_FACTOR = 2377;
const int WS2_FACTOR = 2306;
const int WS3_FACTOR = 2329;
const int WS4_FACTOR = 2309;

const int CAR_WEIGHT = 79;

// HX711 scale init
HX711 scale1;
HX711 scale2;
HX711 scale3;
HX711 scale4;

// Weight sensor values
```

```
bool ws1;
bool ws2;
bool ws3;
bool ws4;

String spotsPath;

// Initialize Servo
void initServo() {
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);

  entranceGate.setPeriodHertz(50);    // standard 50 hz servo
   entranceGate.attach(entrancePin, 500, 2400); // attaches the servo on
pin 18 to the servo object
  exitGate.setPeriodHertz(50);    // standard 50 hz servo
   exitGate.attach(exitPin, 500, 2400); // attaches the servo on pin 18 to
the servo object
}

// Initialize Weight Sensors
void initScales(){
  scale1.begin(LOADCELL_DOUT_PIN_1, LOADCELL_SCK_PIN_1);
  scale2.begin(LOADCELL_DOUT_PIN_2, LOADCELL_SCK_PIN_2);
  scale3.begin(LOADCELL_DOUT_PIN_3, LOADCELL_SCK_PIN_3);
  scale4.begin(LOADCELL_DOUT_PIN_4, LOADCELL_SCK_PIN_4);

  scale1.set_scale(WS1_FACTOR);
  scale1.tare();

  scale2.set_scale(WS2_FACTOR);
  scale2.tare();

  scale3.set_scale(WS3_FACTOR);
  scale3.tare();

  scale4.set_scale(WS4_FACTOR);
  scale4.tare();
```

```
}

// Initialize WiFi
void initWiFi() {
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println(WiFi.localIP());
  Serial.println();
}

// Write bool values to the database
bool sendBool(String path, bool value){
  if (Firebase.RTDB.setBool(&fbdo, path.c_str(), value)){
    Serial.print("Set " + path + " to ");
    Serial.println(value);
    return true;
  }
  else {
    Serial.println("FAILED");
    Serial.println("REASON: " + fbdo.errorReason());
    return false;
  }
}

void setup() {
  Serial.begin(115200);

           /******************************      Setting      up      Servos
****************************/
  initServo();

       /******************************    Setting    up    Weight    Sensors
*****************************/

  // LED boards initiatlization
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);
  pinMode(LED_4, OUTPUT);

  rtc_cpu_freq_config_t rtc_config;
  rtc_clk_cpu_freq_get_config(&rtc_config);
  rtc_clk_cpu_freq_to_config(RTC_CPU_FREQ_80M, &rtc_config);
  rtc_clk_cpu_freq_set_config_fast(&rtc_config);
  Serial.println("Parking Lot Demo");

  Serial.println("Initializing the scale");
  initScales();

          /****************************      Setting      up      Firebase
***************************/
  initWiFi();

  // Assign the api key (required)
  config.api_key = API_KEY;

  /* Assign the RTDB URL (required) */
  config.database_url = DATABASE_URL;

  // Assign the user sign in credentials
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

  Firebase.reconnectNetwork(true);
  fbdo.setResponseSize(4096);

   //  Assign  the  callback  function  for  the  long  running  token  generation
task
        config.token_status_callback    =    tokenStatusCallback;    //see
addons/TokenHelper.h

  // Assign the maximum retry of token generation
  config.max_token_generation_retry = 5;

  // Initialize the library with the Firebase authen and config
  Firebase.begin(&config, &auth);
  // Getting the user UID might take a few seconds
  Serial.println("Getting User UID");
  while ((auth.token.uid) == "") {
    Serial.print('.');
    delay(1000);
  }
  // Print user UID
  uid = auth.token.uid.c_str();
  Serial.print("User UID: ");
  Serial.println(uid);

  // Update paths
  spotsPath = "/" + uid + "/spots";
  entrancePath = "/" + uid + "/entranceAuth";
  exitPath = "/" + uid + "/exitAuth";
  sendBool(entrancePath, false);
  sendBool(exitPath, false);
}

void openEntrance() {
  entranceGate.write(0);
  for (int pos = 0; pos <= maxAngle; pos++) {
    entranceGate.write(pos);
    delay(15);
  }
}

void closeEntrance() {
  entranceGate.write(maxAngle);
  for (int pos = maxAngle; pos >= 0; pos--) {
    entranceGate.write(pos);
    delay(15);
  }
}

void openExit() {
  exitGate.write(0);
  for (int pos = 0; pos <= maxAngle; pos++) {
    exitGate.write(pos);
    delay(15);
  }
}

void closeExit() {
  exitGate.write(maxAngle);
  for (int pos = maxAngle; pos >= 0; pos--) {
    exitGate.write(pos);
    delay(15);
  }
}

void updateAuthState() {

  if (Firebase.ready()) {
    if (Firebase.RTDB.getBool(&fbdo, entrancePath)) {
      entranceAuth = fbdo.boolData();
    }
  }
  if (Firebase.ready()) {
    if (Firebase.RTDB.getBool(&fbdo, exitPath)) {
      exitAuth = fbdo.boolData();
    }
  }
}

void updateWeight() {
  if (scale1.get_units() > CAR_WEIGHT) {
    ws1 = true;
    digitalWrite(LED_1, HIGH);
  } else {
    ws1 = false;
    digitalWrite(LED_1, LOW);
  }

  if (scale2.get_units() > CAR_WEIGHT) {
    ws2 = true;
    digitalWrite(LED_2, HIGH);
  } else {
    ws2 = false;
    digitalWrite(LED_2, LOW);
  }

  if (scale3.get_units() > CAR_WEIGHT) {
    ws3 = true;
    digitalWrite(LED_3, HIGH);
  } else {
    ws3 = false;
    digitalWrite(LED_3, LOW);
  }

  if (scale4.get_units() > CAR_WEIGHT) {
    ws4 = true;
    digitalWrite(LED_4, HIGH);
  } else {
    ws4 = false;
    digitalWrite(LED_4, LOW);
  }
}

void rebootScales() {
  scale1.power_down();              // put the ADC in sleep mode
  scale2.power_down();              // put the ADC in sleep mode
  scale3.power_down();              // put the ADC in sleep mode
  scale4.power_down();              // put the ADC in sleep mode
  delay(1000);
  scale1.power_up();
  scale2.power_up();
  scale3.power_up();
  scale4.power_up();
}

void loop() {

  updateAuthState();
  updateWeight();

  if (entranceAuth) { // If auth is true then open gate
    openEntrance();
    delay(1800);
  }
  if (exitAuth) {
    openExit();
    delay(1800);
  }

   if (entranceAuth && sendBool(entrancePath, false)) { // Close gate after
auth
    closeEntrance();
  }
  if (exitAuth && sendBool(exitPath, false)) { // Close gate after auth
    closeExit();
  }

  if (Firebase.isTokenExpired()){
    Firebase.refreshToken(&config);
    Serial.println("Refresh token");
  }

   if (Firebase.ready() && (millis() - sendDataPrevMillis > timerDelay ||
sendDataPrevMillis == 0)){
    sendDataPrevMillis = millis();

    sendBool(spotsPath + "/ws1", ws1);
    sendBool(spotsPath + "/ws2", ws2);
    sendBool(spotsPath + "/ws3", ws3);
    sendBool(spotsPath + "/ws4", ws4);
  }

  rebootScales();
}
```