CAPSTONE PROJECT REPORT

Smart Parking System

IOP490_G2	
Group Members	Nguyễn Quang Huy – HE151417 Hà Minh Nghĩa – HE160403 Nguyễn Trường Hưng – HE160047
Supervisor	Đặng Văn Hiếu
Ext Supervisor	None
Capstone Project Code	SPS

– Hanoi, November 2023 –

I. PROJECT INTRODUCTION

1. Overview

1.1 Project Information

- Project name: Smart Parking System
- Project code: SPS
- Group name: IOP490_G2
- Project type: Product, Website, Application

1.2 Project Team

Full Name	Role	Email	Mobile
Dang Van Hieu	Instructor	hieudv2@fu.edu.vn	0978826633
Nguyen Quang Huy	Leader	huynqhe151417@fpt.edu.vn	0862424010
Ha Minh Nghia	Member	Nghiahm160403@fpt.edu.vn	0981268853
Nguyen Truong Hung	Member	hungnthe160047@fpt.edu.vn	0386828538

Table I.1.2: Project Team

2. Product Background

Parking is a major challenge in Vietnam's crowded cities like Hanoi and Ho Chi Minh City. Rapid growth in vehicles has far exceeded parking infrastructure capacity. FPT University's campuses, especially Hoa Lac, face huge parking shortages during peak times. Students frequently complain about lateness due to overloaded parking lots, forcing them and teachers to seek spots on distant roads. Inefficient use of parking areas is also a problem. This leads to long search times for spots, illegal parking, traffic jams, and safety issues. Recognizing this reality, our team has proposed a Smart Parking System to optimize supply, guide drivers to open spots, enforce regulations, increase efficiency, and deliver economic benefits.

3. Existing Systems

3.1 My Parking

3.1.1 Description

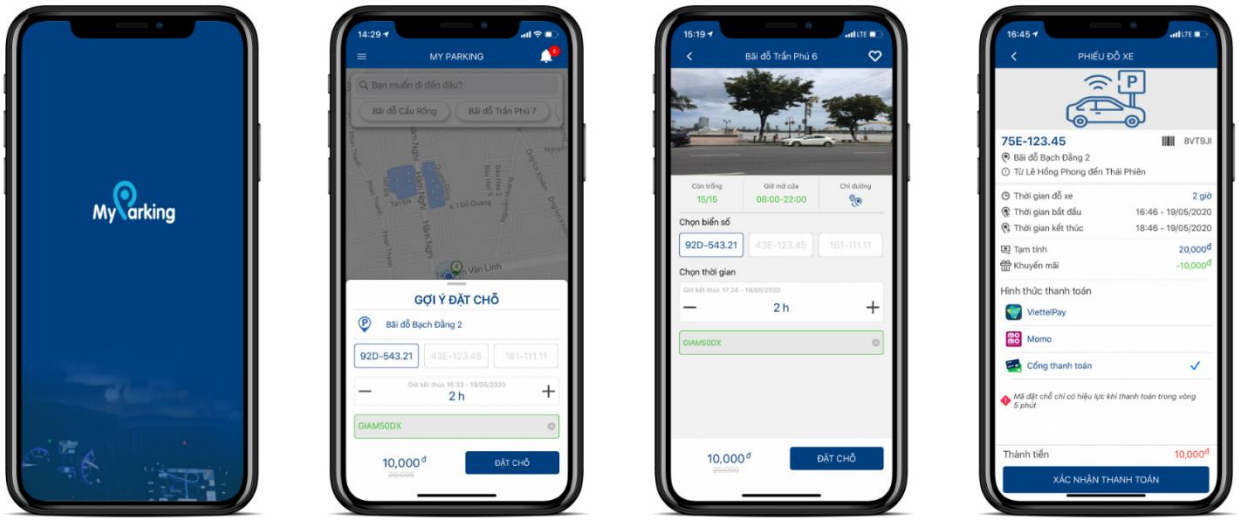


Figure I.3.1: My Parking App Interface

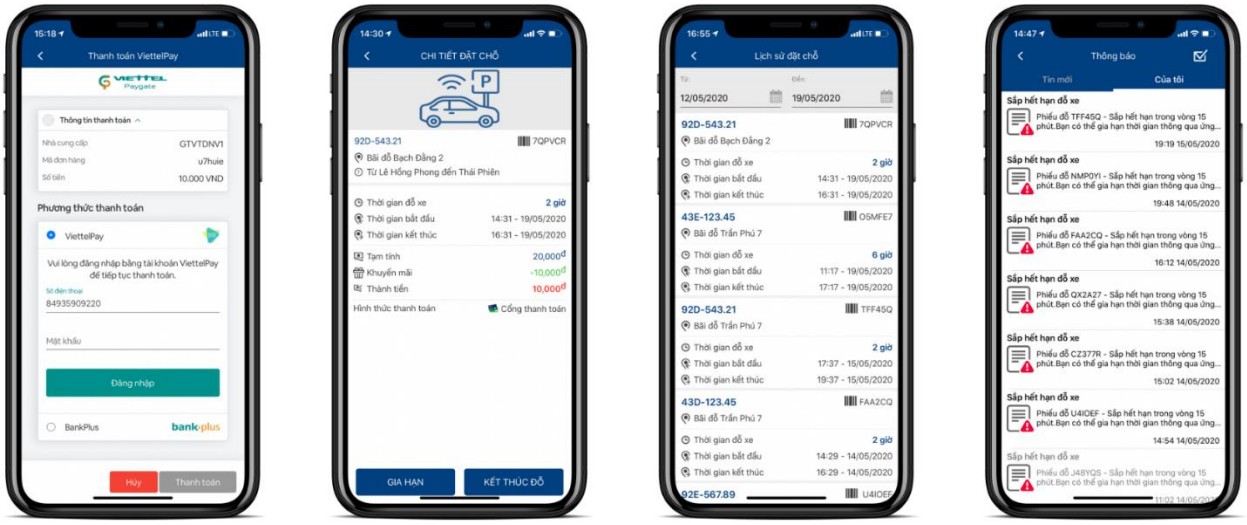


Figure I.3.2: My Parking App Interface

My Parking is a smart parking app developed and launched by Viettel, the largest mobile network operator in Vietnam. The app aims to help drivers easily find and pay for available parking in major cities across Vietnam.

3.1.2 Features

- Real-time parking availability mapping
- Mobile payment for parking
- Parking history and receipts

- Reservation capabilities

3.1.3 Pros & Cons

Pros

+ The app provides an easy way to find, pay for, and manage parking from the phone
+ Allows for contactless digital payments of parking fees through the app
+ Drivers can instantly see parking availability of garages nearby.

Cons

+ Privacy concerns with tracking driver behavior
+ The app is not yet widely used
+ No technological advancements are used

3.2 Smart Parking System

3.2.1 Description

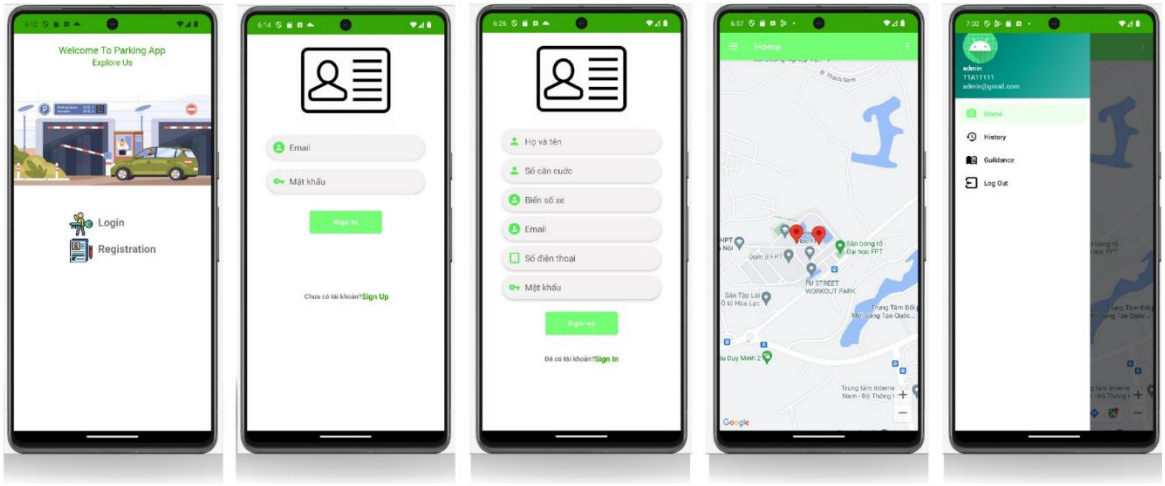


Figure I.3.3: Smart Parking System App Interface

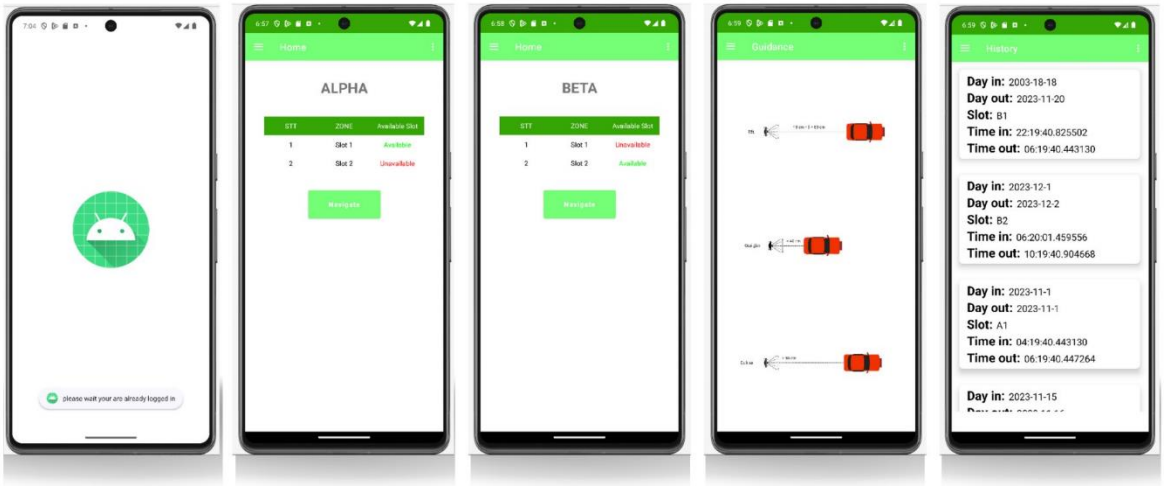


Figure I.3.4: Smart Parking System App Interface

The Smart Parking System is an Internet of Things (IoT) product designed to improve and optimize parking lot management and usage through integrating sensors, microcontrollers, and control software. Sensors and microcontrollers gather data and communicate it via protocols to the server. The server then processes the sensor information, manages parking durations, performs license plate recognition from camera images, and provides real-time parking lot status. Additionally, the mobile app or web interface lets users view availability, reserve spots, and receive parking notifications. By connecting hardware data collection with software information processing and user interfaces, the smart parking system aims to efficiently coordinate parking resource usage.

3.2.2 Features

- Real-time parking occupancy monitoring via ground sensors and cameras
- Parking space availability communicated to drivers via mobile app or signs
- Allow user to see parking history
- Optimization algorithms to direct drivers to open spaces
- Database for managing car park

3.2.3 Pros & Cons

- Pros

+ Reduces search time for parking spot
+ Improve campus parking shortages and congestion
+ Enables data-driven parking management

- Cons

+ Huge costs for hardware and software installation
+ Potential technical issues with sensors or cameras
+ Contactless payment and reservations through app under-develop

4. Business Opportunity

FPT University's rapid growth has created daily parking shortages and congestion issues due to inadequate parking infrastructure and management. This presents an opportunity for an intelligent Smart Parking System to optimize parking utilization. By sensing occupancy and guiding drivers to open spots via mobile apps and websites, the system can significantly improve parking efficiency on campus. Successful pilots at FPT University can demonstrate the benefits and position the made-in-Vietnam solution for broader adoption at other universities, commercial areas, and cities nationwide. The Smart Parking System offers a win-win solution to boost productivity and demonstrate Vietnam's capabilities in smart city technologies.

5. Product Vision

The Smart Parking System will address FPT University's daily parking challenges through real-time occupancy detection and intelligent assignment algorithms to enable efficient campus parking. Our vision is to collaborate with FPT administrators to collect insights and tailor the system to FPT's specific parking needs via mobile apps, signs, and dashboards. We aim to complete development and pilot the system at FPT campuses by March 2024, reducing search times by 50% and improving driver satisfaction. By delivering an

optimized Vietnam-centric parking solution, we envision setting the standard for smart campus parking nationwide. Successful FPT implementation can catalyze adoption at other universities, commercial zones, and cities. The Smart Parking System aligns with FPT's technology leadership and Vietnam's emerging smart city landscape. We foresee a future of convenient, intelligent parking pioneered at FPT University.

6. Project Scope & Limitations

6.1 Hardware Functions

The diagram below illustrates the function of hardware

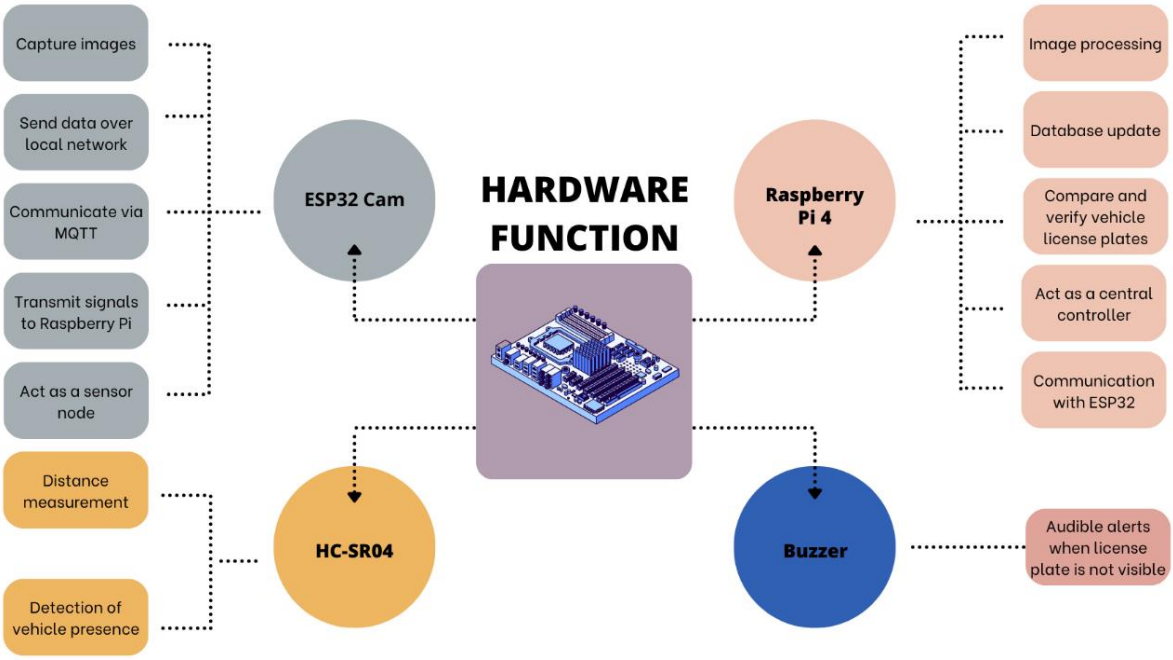


Figure I.6.1 Image of hardware functions

ESP32 Camera
+ Capture Images
+ Send data over local network
+ Communicate via MQTT
+ Transmit signals to Raspberry Pi
+ Act as a sensor node

HC-SR04 Ultrasonic Sensor
+ Distance measurement
+ Detection of vehicle presence

Raspberry Pi 4
+ Image processing
+ Database update
+ Compare and verify vehicle license plates
+ Act as a central controller
+ Communication with ESP32

Buzzer

+ Audible alerts when license plate is not visible

6.2 Software Functions

The diagram below illustrates the function of software

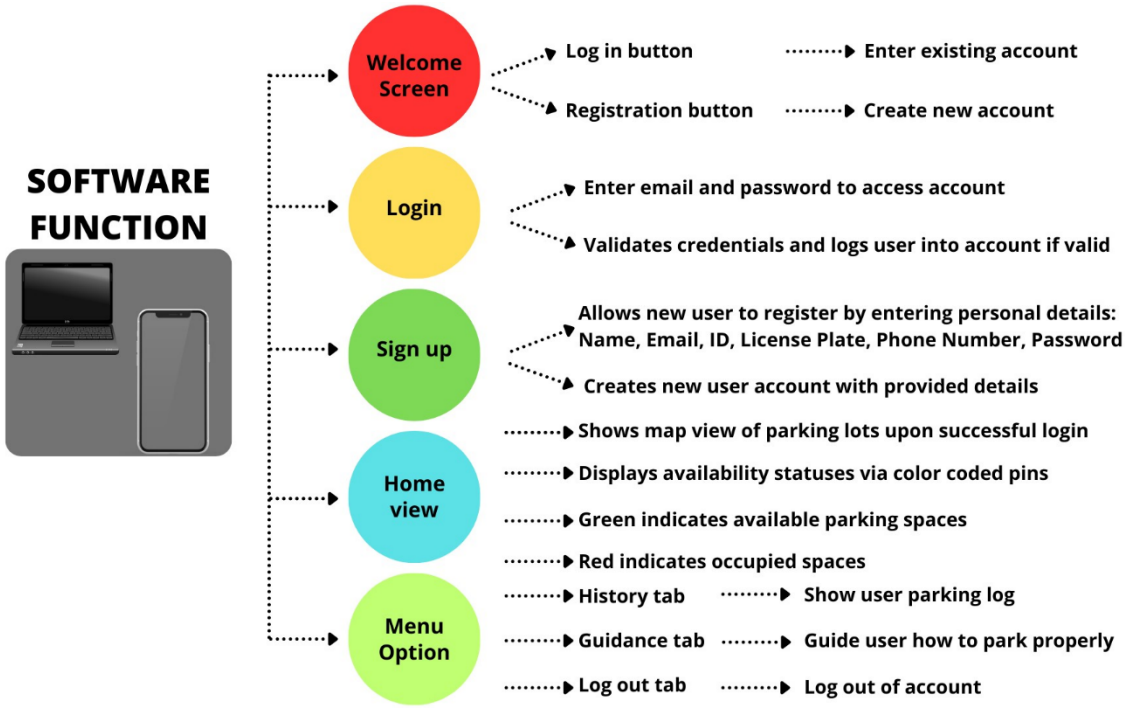


Figure I.6.2 Image of software functions

Home Screen

+ Log In button/option to enter existing account
+ Sign Up button/option to create new account

Login

+ Allows user to enter email and password to access account
+ Validates credentials and logs user into account if valid

Sign Up

+ Allows new user to register by entering personal details: Name, Email, ID, License Plate, Phone Number, Password
+ Creates new user account with provided details

Home View

+ Shows map view of parking lots upon successful login
+ Displays availability statuses via color coded pins
+ Green indicates available parking spaces
+ Red indicates occupied spaces

Menu Option

+ History tab
+ Guidance tab
+ Log out tab

6.3 Limitations & Exclusions

- Condition while taking the picture
- Connection Unstable
- Contactless payment and reservations through app under-develop

II. PROJECT MANAGEMENT PLAN

1. Overview

1.1 Scope & Estimation of Software Function

Simple: 0 - 5 man-days
Medium: 6 - 9 man-days
Complex: more than 10 man-days

#	WBS Item	Complexity	Est. Effort (Man-days)
1	Home Screen	Complex	24
1.1	Login	Medium	10
1.1.1	Enter email and password to access account	Simple	3
1.1.2	Validates credentials and logs user into account if valid	Medium	7
1.2	Sign Up	Complex	14
1.2.1	Allows new user to register by entering personal details	Medium	7
1.2.2	Creates new user account with provided details	Medium	7
2	Home View	Complex	30
2.1	Shows map view of parking lots upon successful login	Medium	7
2.2	Displays availability statuses via color coded pins	Complex	10
2.3	Green indicates available parking spaces	Medium	8
2.4	Red indicates occupied spaces	Simple	5
3	Menu Option	Complex	21

3.1	History tab	Medium	7
3.2	Guidance tab	Medium	7
3.3	Log out tab	Medium	7
	Total Estimated Effort on Software Function (man–days)		***75***

1.2 Scope & Estimation of Hardware Function

#	WBS Item	Complexity	Est. Effort (Man-days)
1	**ESP32 Camera**	Complex	45
1.1	Capture Images	Simple	8
1.2	Send data over local network	Simple	5
1.3	Communicate via MQTT	Complex	16
1.4	Transmit signal to Raspberry	Complex	16
1.5	Act as a sensor node	Simple	1
2	HC-SR04 Ultrasonic Sensor	Simple	2
2.1	Distance measurement	Simple	1
2.2	Detection of vehicle presence	Simple	1
3	Raspberry Pi 4	Complex	42
3.1	Image Processing	Complex	22
3.2	Database Update	Simple	5
3.3	Compare and verify vehicle license plate	Simple	5
3.4	Act as a central controller	Simple	5
3.5	Communication with ESP32	Simple	5

4	Buzzer	Simple	1
4.1	Audible alerts when license plate is not visible	Simple	1
	Total Estimated Effort on Hardware Function (man–days)		***90***

1.3 Project Objectives

- Timeliness (%): 80%
- Allocated Effort (man-days): 165
- Defect Distribution:

#	Testing Stage	Test Coverage	No. of Defects	% of Defect	Notes
1	Reviewing	100%	35	48%	
2	Unit Test	100%	25	20%	
3	Compatibility Test	100%	20	16%	
4	System Test	100%	10	8%	
5	Performance Test	100%	10	8%	

Table 1.3.1: Quality

- Allocated Effort

#	Members	Working hours in a day
1	Nguyen Quang Huy	5h
2	Ha Minh Nghia	5h
3	Nguyen Truong Hung	5h

Table 1.3.2: Allocated Effort

1.4 Project Risks

#	Risk Description	Avoidance plan	Contingency plan	Status
1	Failure to meet deadline	- Plan and develop schedule carefully - Assign tasks carefully	- Find the root cause of the problem - Reassign tasks - Change project scope	Closed
2	Change in requirements	- The supervisor and the entire team must review any new updates to requirements	- All changes in requirements will be announced in the next daily team meeting	Closed

3	Misunderstanding of requirements	- Discuss requirements carefully with the customer - Any ambiguity in understanding requirements of team members will be recorded and handed to supervisor to clarify with customer	- Update code and documentation to adapt with actual requirements	Closed
4	Illness or absence of team members	- Provide meeting schedules in advance - Team member must announce absence in advance	- All meetings with supervisor will be recorded for absent members - Assign the tasks of absent member to other members - Work overtime if necessary	Closed
5	Conflict between team members	- Everything must be documented - Every team member has to express clearly and carefully	- Make sure any miscommunication will be resolved	Closed
6	Server failure	- Use paid and certified servers	- Use a different server	Closed

Table 1.4.1: Risk Management

2. Management Approach

2.1 Project Process

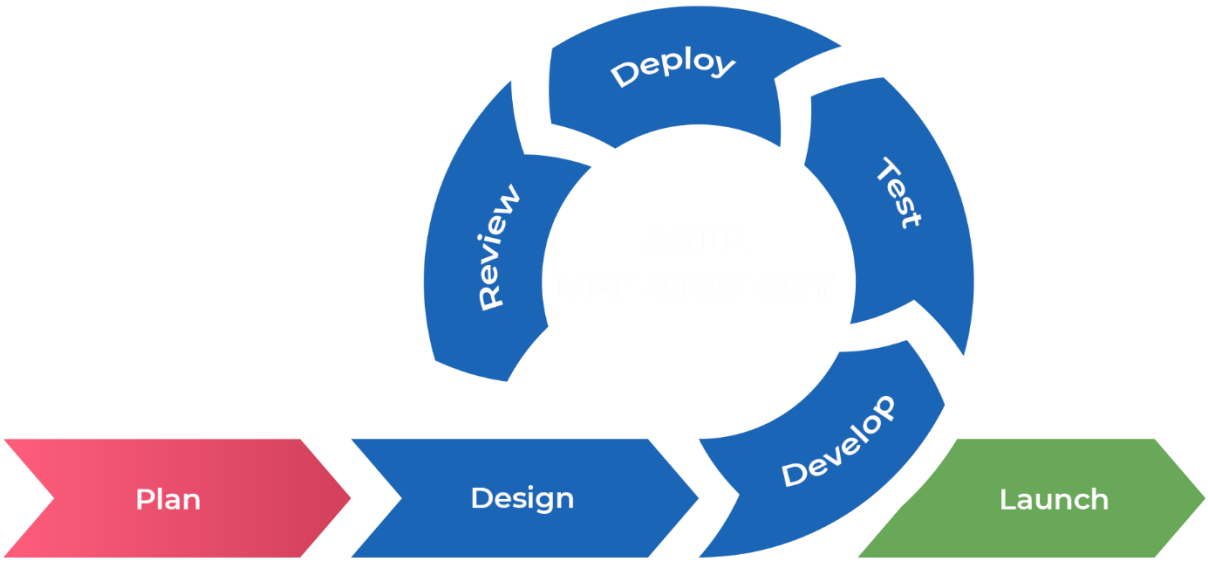


Figure II.2.1: Agile Process

The main reason for using Agile development for the Smart Parking Management system is to enable rapid iterations and continuous integration of user feedback into the parking technology. Agile will allow the team to frequently deliver working software to get stakeholder input, learn, and refine the system.

The main functions of Agile for the Smart parking project will be:

Flexibility: Agile sprints and dynamic backlogs will empower the development team with the adaptive capacity to swiftly reprioritize parking management features in accordance with evolving user needs.

Enhanced Collaboration: Cross-functional Agile rituals including standups, reviews, retrospectives, and planning sessions will promote increased transparency as well as tighter alignment between parking technology developers, parking facility administrators, and campus drivers.

Continuous Improvement: The iterative approach inherent to Agile will drive ongoing enhancement of system functionality and optimization of parking efficiency through successive, rapid build-measure-learn cycles.

Transparency: Agile ceremonies will provide visibility into development status and field challenges. This enables parking managers to gain insights and align policies, drivers and staff to preview new features and provide feedback, and collaborative identification of integration risks.

2.2 Quality Management

2.2.1 Compatibility Testing

- Validate integration with parking gate systems
- Confirm sensor connectivity across various devices
- Assess database

2.2.2 Reliability Testing

- Simulate heavy load for app and API reliability
- Failure mode testing for system redundancy
- Test stability across extended duration

2.2.3 Scalability Testing

- Evaluate ramp up with increased sensors
- Stress test ability to handle usage spikes
- Push database queries and storage volumes

2.2.4 Data Integrity Testing

- Validate accuracy of occupancy status
- Corroborate timestamp consistency
- Audit logs for transaction history

2.2.5 Security Testing

- Penetration testing to identify vulnerabilities
- Re-authentication for sensitive operations
- Encryption of personal identifiers

2.2.6 Performance Testing

- Load testing of real-time occupancy dashboards
- Spike response times for peak campus hours
- Trigger failover checks and recovery

This help focus on quality through IoT testing processes will boost confidence in operational stability, security protections, and system resilience as we scale the solution to meet expanding needs.

2.3 Training Plan

#	Training Area	Participants	Duration	Waiver Criteria
1	Eclipse Mosquitto	HuyNQ	9/9/2023-13/9/2023	Mandatory
2	Firebase	All members	13/9/2023-18/9/2023	Mandatory
3	Git, Github	All members	18/9/2023-20/9/2023	Mandatory
4	Python	HuyNQ	20/9/2023-23/9/2023	Mandatory
5	Bash	HungNT	23/9/2023-26/9/2023	Mandatory

6	Apache	HungNT	9/9/2023-13/9/2023	Mandatory
7	Linux	HuyNQ	20/9/2023-23/9/2023	Mandatory
8	PHP	All members	1/10/2023-5/10/2023	Mandatory
9	Android Studio	NghiaHM	9/9/2023-13/9/2023	Mandatory
10	JavaScript	NghiaHM	26/9/2023-1/10/2023	Mandatory

Table II.2.3: Training Plan

3. Project Deliverables

#	Deliverable	Due Date	Notes
1	Report 1 _ Project introduction	25/09/2023	Product Background, Existing Systems, Business Opportunity, Product Vision, Functions Role, Project Scope & Limitations.
2	Report 2 - Project Management Plan	10/10/2023	WBS, Project Process, Plan and Schedule, Project Organization, Project Communication, Configuration Management.
3	Report 3 – Product System Design	30/11/2023	Product Overview, Business Rule, User Requirement, Functional Requirement, Non-Functional Requirement.
4	Report 4 - Test Document	08/12/2023	Test Model. Test Plan, Test Cases, Test Reports.
5	Report 5 - System Set Up Guide	30/11/2023	Deliverable Package, Installation Guides, User Manual.
6	Report 6 - Final Reports	11/12/2023	The summary of all reports.
7	Final Code & All Report	11/12/2023	All reports, Presentation Slide, Source Code.

Table II.3.1 Project Deliverables

4. Responsibility Assignments

4.1 Team Structure

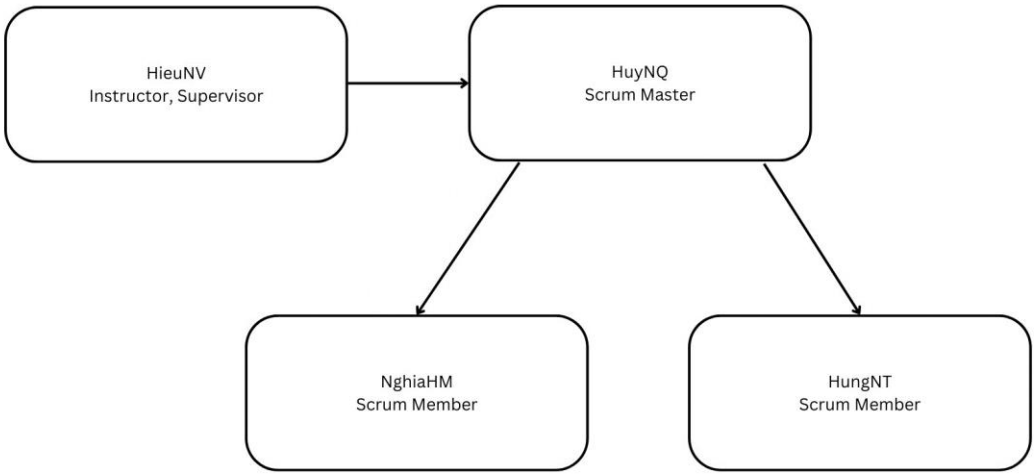


Figure II.4.1: Team Structure

4.2 Responsibility Structures

D~Do; R~Review; S~Support; I~Informed; <blank>- Omitted

Responsibility	HieuNV	HuyNQ	NghiaHM	HungNT
Project Planning	R	D	D	D
Project Tracking	I	D	R	R
Prepare Project Introduction Document	R	S	S	D
Prepare Document	R	R	R	D
Code Back-End	R	I	D	I
Code Front-End	R	I	D	I
Prepare Test Document and Report	R	S	S	D

Prepare Final Report (all document and presentation slide)	R	D, S	D, S	D, R

Table II.4.2: Responsibility Assignments

5. Project Communication

Communication Item	Participants	Purpose	When	Type, Tool, Method(s)
Daily team meeting	All team members	Review and create a work plan for the next day	9pm everyday	Online – Google Meet
Weekly meeting with Supervisor	All team members Supervisor	Review report and document	9am every Monday	Offline - At school

Table II.5.1: Project Communication

6. Configuration Management

6.1 Document Management

- We use MS Office as our primary tool for sharing, editing and version control of our documents, along with Weekly Reports documents.
- It allows us to observe what has changed in the documents and who is accountable for that modification, as well as to compare versions of the document at the same time.

6.2 Source Code Management

For version control of our source code, we use Arduino, GitLab manipulates every time that need to store, upload, and download code easier. Control changes in code quickly and accurately. It easily manages, distributes work, and completes better quality programming projects

6.3 Tools & Infrastructures

Category	Tools / Infrastructure
Technology	Eclipse Mosquitto (MQTT),
Database	Firebase (Firestore), phpMyAdmin, MariaDB (MySQL)
IDEs/Editors	Visual Studio Code, Arduino IDE, Android Studio, Thonny IDE

Diagramming	Draw IO, Canva
Documentation	MS Office, Google Docs
Version Control	GitHub (Source Codes), Google Drive/Google Doc (Document)
Deployment server	Apache2 (Linux Server)
Communication tools	Gmail, Messenger, Zalo

Table II.6.1 Tools and Infrastructures

III. Product Requirement Specification

1. Product overview

The Smart Parking System is an intelligent real-time solution designed to optimize parking through automated guidance. The diagram below illustrates the system process. Version 1.0 focuses on core capabilities of occupancy monitoring via sensors, digital mapping of availability, mobile app and website for navigation. Future releases will build on this foundation to add reservations, payments, charging coordination for convenience of the users, and data analytics for parking administrators.

2. Hardware Design

2.1 Operation Parameters

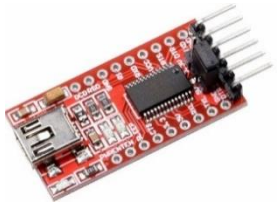
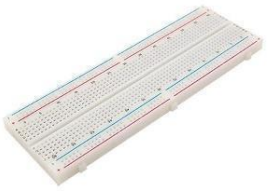
Parameter	Property	Unit
Operating voltage	5	V
Network	Wi-fi	
Connect Ability	Wi-fi	
Server	LAMP (Linux, Apache, MySQL, PHP)	

Processor	ESP32 CAM AI thinker	1
Processor	Raspberry Pi 4	1
Sensor	HC-SR04	1
Alert	Buzzer	1
Module	FT232RL FTDI	1
MQTT	Mosquitto	

Table III.2.1: Table of materials

2.2 Devices

Raspberry Pi 4		1
ESP32 Cam AI thinker		1

HC-SR04 Ultrasonic Sensor Module		1
Camera OV2640		1
Buzzer		1
FT232RL FTDI		1
XIAOMI 10000MAH 18w backup battery		1
Breadboard		1

10 Jumper Wire male – female 10cm		1

Table III.2.2: Table of devices

In our ongoing project, we have successfully integrated all four key components - sensors, applications, networks and backend infrastructure.

The hardware encompasses ultrasonic HC-SR04 sensors, 5V buzzers, AI Thinker ESP32 Cam devices, a Raspberry Pi 4, circuit boards, wiring and a power bank supplying system electricity. Wi-Fi connectivity links the ESP32 Cam and Raspberry Pi to enable efficient, continuous MQTT-based data exchange.

The backend typically comprises databases for information storage, servers for request processing, cloud platforms for large-scale data and tools to analyze and handle sensor-derived data.

2.3 Research Approach & Method

By applying fundamental knowledge of IoT technology, sensor devices, Artificial Intelligence, network, and drawing insights from research outcomes of existing products in daily life, it becomes feasible to develop smart parking system. This system monitor, record, and save the entry-exit timings information and the license plate numbers of vehicles within a large parking lot. Capitalizing on IoT technology, the system will be continuously monitored and evaluated in real-time, functioning in the following diagram:

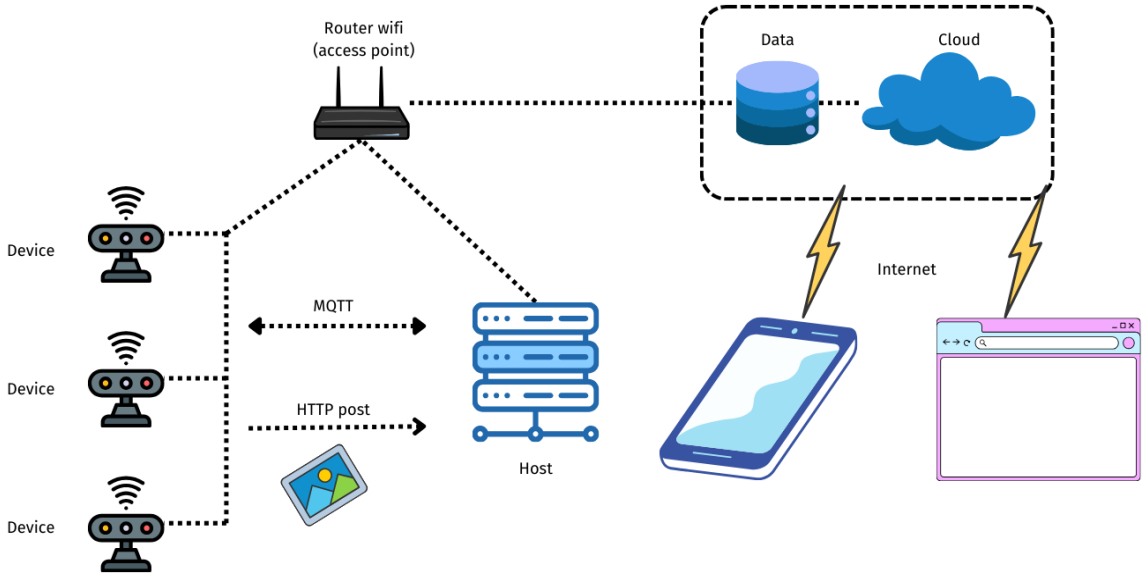


Figure III.2.3: Structure Diagram

The system relies on an innovative combination of sensors and artificial intelligence to enable automated and efficient parking management. As vehicles enter parking spots, the sensors automatically detect occupancy, while dedicated cameras capture license plate images for precise documentation. The license plate data is aggregated on a central server, then made accessible via cloud hosting for flexible remote access.

By compiling real-time occupancy data and vehicle records, the system allows applications and interfaces to provide smart services to users. Drivers can conveniently check parking availability, review parking history, and utilize other assistive features for an optimized parking experience with minimal human intervention required in the management process.

Main functions of the project:

1) ***Ultrasonic Sensor Detects Objects***: The ultrasonic sensor is used to detect the presence of cars in the parking area.
2) ***ESP32-CAM Captures License Plate Images***: The ESP32-CAM is integrated to capture images of vehicle license plates when the ultrasonic sensor detects a vehicle.
3) ***Sends Images and Notifications to Raspberry Pi via LAMP Server***: The system sends captured images and notifications of vehicle detection to the Raspberry Pi through a LAMP (Linux, Apache, MySQL, PHP) server.
4) ***Updates Entry/Exit Time and Parking Status to Firebase***: Data on entry/exit times and parking status is stored and updated on Firebase (Google cloud database).
5) ***Uses AI to Analyze Images and Extract License Plate Information***: The system uses artificial intelligence to analyze captured images and extract vehicle license plate information from the images.
6) ***Checks Image Data against Registration Data on Firebase***: License plate data extracted from images is checked for accuracy. The system then compares it to license plate registration data stored on Firebase.
7) ***Notifications to Users If License Plate Cannot Be Captured or to Admin If Vehicle Is Not Registered***: If the vehicle license plate cannot be captured automatically by the system, it will notify the user with the sound from buzzer to remind them to park properly. If the license plate does not match or is not registered in the database, the information will still be saved in the data for the admin to easily manage.

The diagram below illustrates the process and important steps in operation between the ESP32 CAM and the Raspberry Pi. The boxes in the diagram represent the functions, settings, and interactions between the different components of the system.

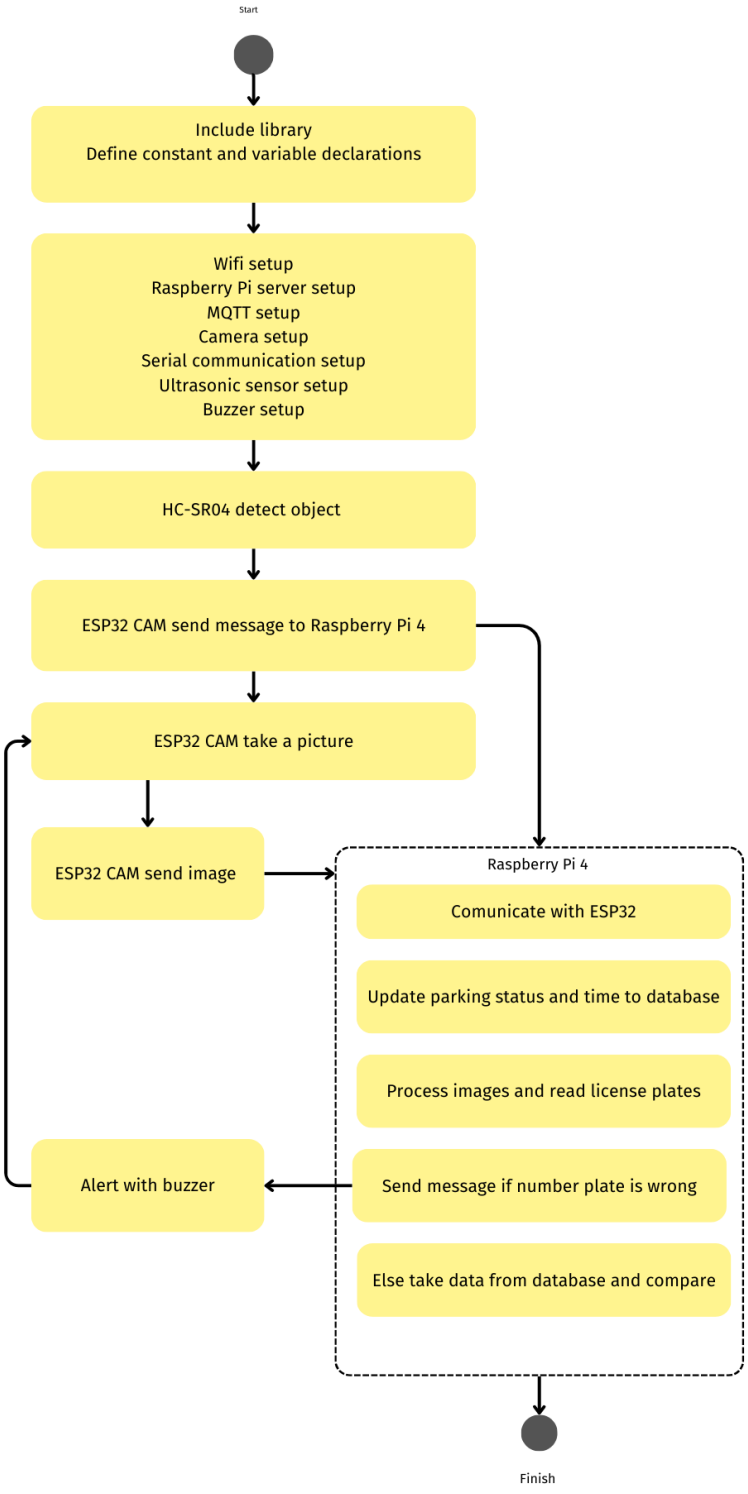


Figure III.2.1: Main Operation and Interaction Algorithm Diagram

2.3.1 ESP32_CAM

2.3.1.1 Car detection by Ultrasonic Sensor

The following diagram illustrates the sequential and logical operating process of the HC-SR04 ultrasonic sensor. The program utilizes ultrasonic sensors to measure distance and subsequently determine whether to transmit vehicle detection notifications to a server via MQTT protocol.

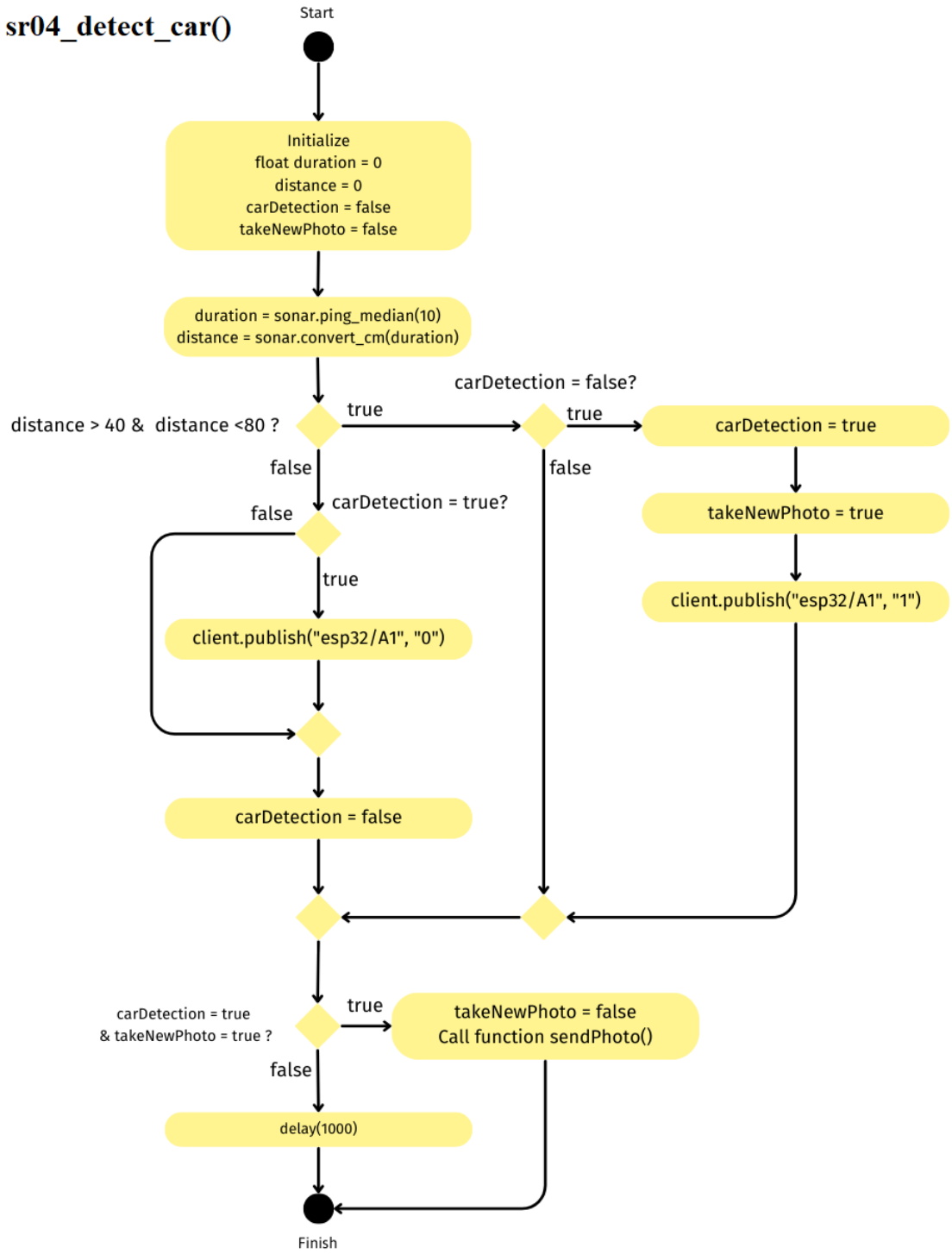


Figure III.2.2: Ultrasonic Sensor Algorithm Diagram

The program uses an ultrasonic sensor to measure distance and then decides to send information about car detection to the server via MQTT protocol. The process starts by measuring the time-of-flight value of the reflected ultrasonic wave from the object and converting it into distance units. This distance value can be displayed on the Serial Monitor for monitoring.

If the measured distance is within the range of 40cm to 80cm, the program will check the state of the carDetection variable. If no car has been detected previously (*carDetection == false*), the program will send the message "1" to the topic "esp32/A1" via MQTT and set the takeNewPhoto variable to true. In case a car has been detected previously (*carDetection == true*), the program sends the message "0" to the corresponding topic.

If the distance does not lie within the range of 40cm to 80cm, the program will proceed to check the state of the carDetection variable to determine sending the "0" message to the server via MQTT and sets the carDetection variable to false.

After the checking and sending information process, if a car has been detected and the takeNewPhoto variable is set to true, the program will capture and send the photo via the *sendPhoto ()* function. The purpose of the two variables carDetection and takeNewPhoto is for the ESP32 CAM to only capture the photo once without repetition in *void loop ()*.

Finally, before re-measuring the distance, the program will delay 1 second to prepare for the next measurement.

2.3.1.2 Communicate with Raspberry via MQTT Protocol

In order to utilize and communicate through the MQTT protocol, two essential functions are required: the callback function and the *receive_MQTT_message()* function. The callback function is declared within void *setup()* and serves the purpose of handling incoming MQTT messages from the broker.

The callback function is defined to handle incoming messages from the MQTT Broker – a messaging middleware system. When a new message arrives at the device, the callback function is triggered, receiving parameters like topic, message, and length. The process starts by printing out the notification "Message arrived on topic: " along with the MQTT message's topic. This helps identify the message's destination and provides a monitoring step for the subsequent process.

Next, the message content is printed out character-by-character sequentially. At the same time, each character is integrated into the String type variable messageTemp. This allows storing and processing data received from the MQTT message.

After that, through a loop, the MQTT message's data bytes are iterated through and integrated into messageTemp. This step provides an organized, accurate mechanism to collect data from the incoming message.

An important next part is converting messageTemp's content – collected from the received message – into an integer (MQTTmessage) via the *toInt ()* function. This helps convert data from character format to numeric for convenient usage in subsequent processing operations.

All activities and processing in the callback function are illustrated via printouts to the Serial Monitor, providing a way to track and verify each step in handling the MQTT message.

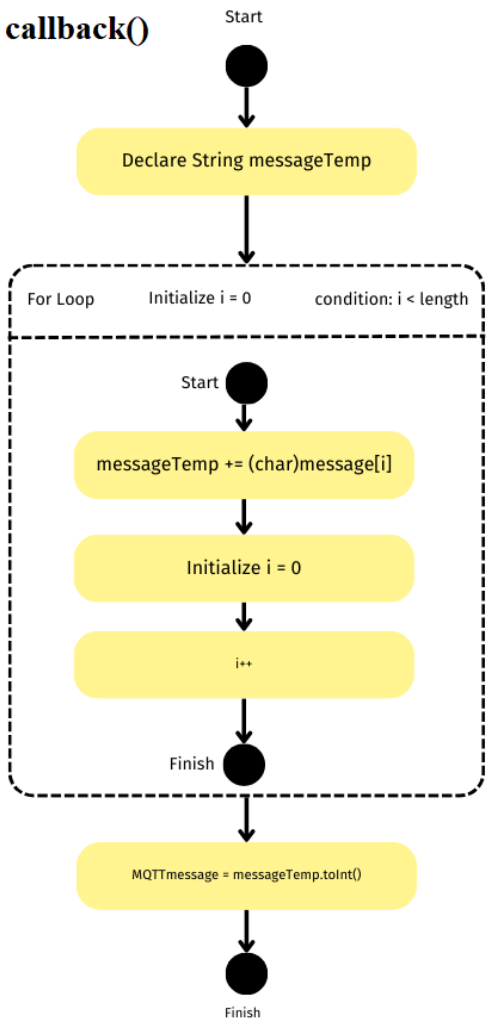


Figure III.2.3: *Callback ()* algorithm diagram

On the other hand, the *receive_MQTT_message()* function is responsible for establishing the MQTT connection and subscribing to specific topics ("rpi/broadcast" and "example/topic") to receive messages sent by the broker.

The receive_MQTT_message function is written to manage the connection and communication between the ESP32 CAM and an MQTT Broker – a common system for transmitting information in IoT networks. This code plays an important role in maintaining a stable and reliable connection with the MQTT Broker, allowing the ESP32 CAM to receive and send messages to/from specific topics on the broker.

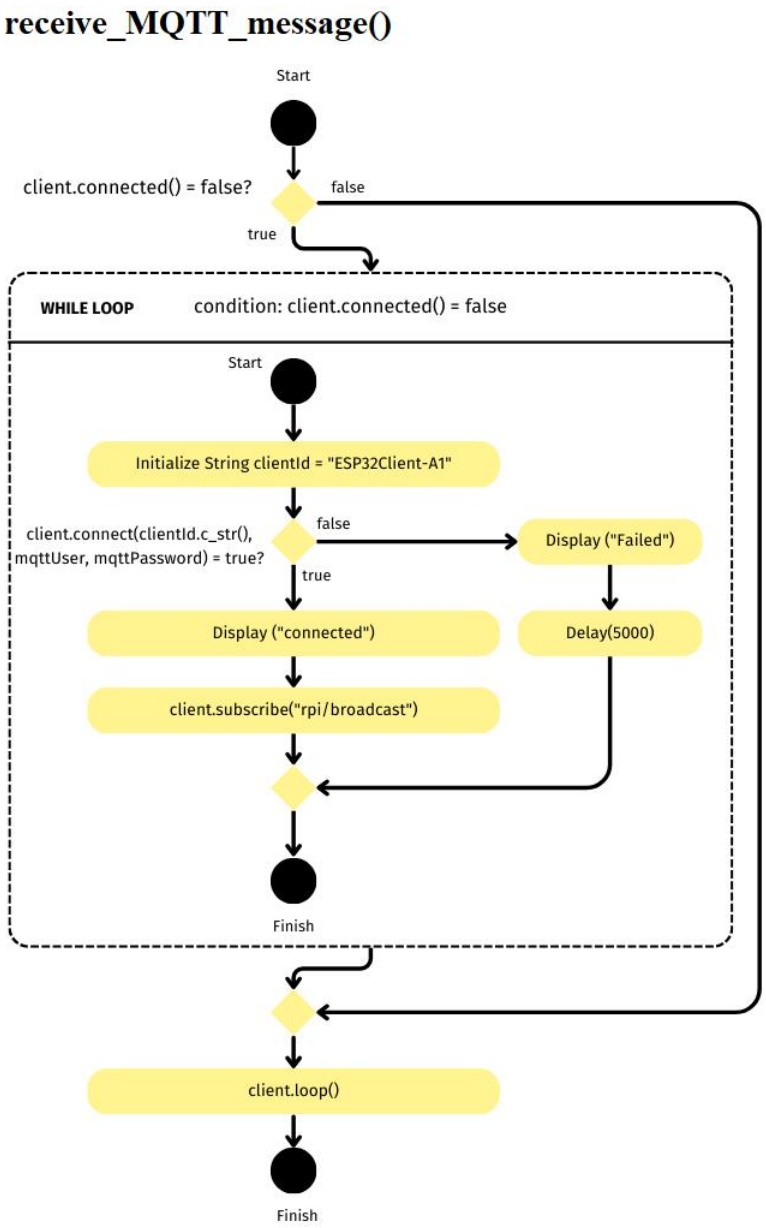


Figure III.2.4: *receive_MQTT_message ()* algorithm diagram

The function starts by checking the connection state of the MQTT client. If the client is not connected or loses connection with the MQTT Broker, the function will attempt reconnecting. In this process, the client ID is first created with the ID "ESP32Client-A1", and the ESP32 CAM tries connecting to the MQTT Broker by using the provided login credentials of client ID, MQTT user and MQTT password.

If the connection succeeds, the ESP32 will subscribe to desired topics on the MQTT Broker. This allows it to receive messages from those topics when new information is posted.

In case of unsuccessful connection, the ESP32 CAM prints out the specific error message and waits for a period (5 seconds) before trying to reconnect. This process continues repeating until successful connection, ensuring the ESP32 CAM maintains a stable link with the MQTT Broker to continuously and reliably exchange information. This is crucial for IoT system operation, where continuous messaging and connectivity are indispensable for remote device management and control.

2.3.1.3 Capturing and Sending images to Raspberry Pi

The diagram below will describe the function of taking a photo using a camera connected to the ESP32 and processing the photo data and sending the captured photo to an HTTP server via the POST method

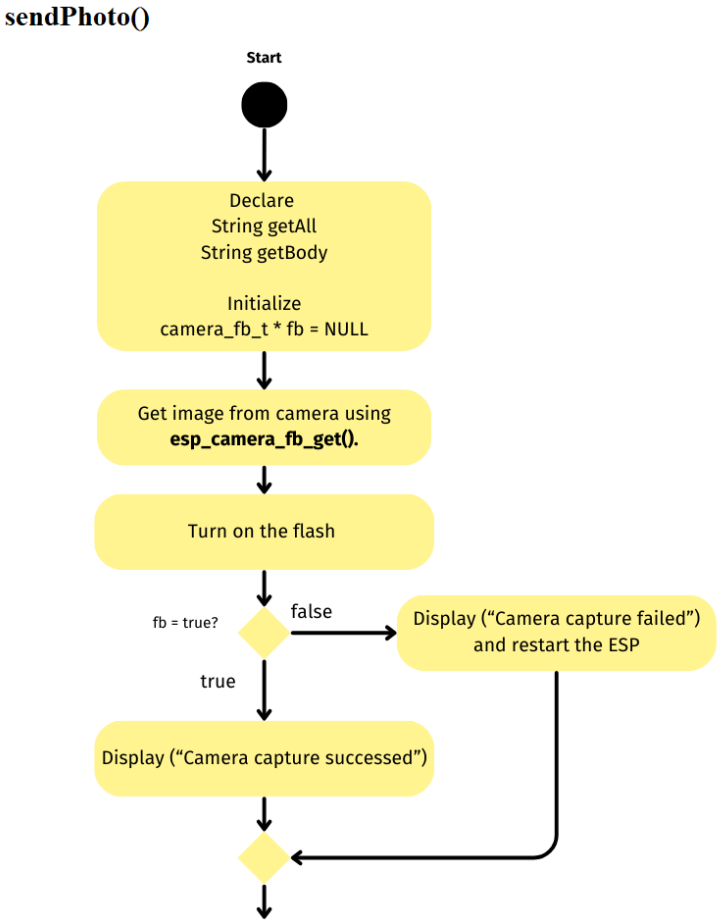


Figure III.2.5: *send_photo ()* algorithm diagram (1)

The diagram shows a process of capturing an image from the ESP32 CAM camera and then connecting to a server via network. It starts by initializing two string variables 'getAll' and 'getBody' to store the destination path and time information. The code continues by requesting image information from the camera via the '*esp_camera_fb_get ()*' function. At the same time, it calls a function to turn on the flash LED.

After getting the image information, the code checks if the image capture was successful by checking the fb variable. If there is no image data (***fb* = *NULL***), the ESP32 CAM will print "**Camera capture failed**" on the Serial port and then restart to resolve the issue. In case of successful image capture, the ESP32 CAM will print "**Camera capture succussed**" on the Serial port.

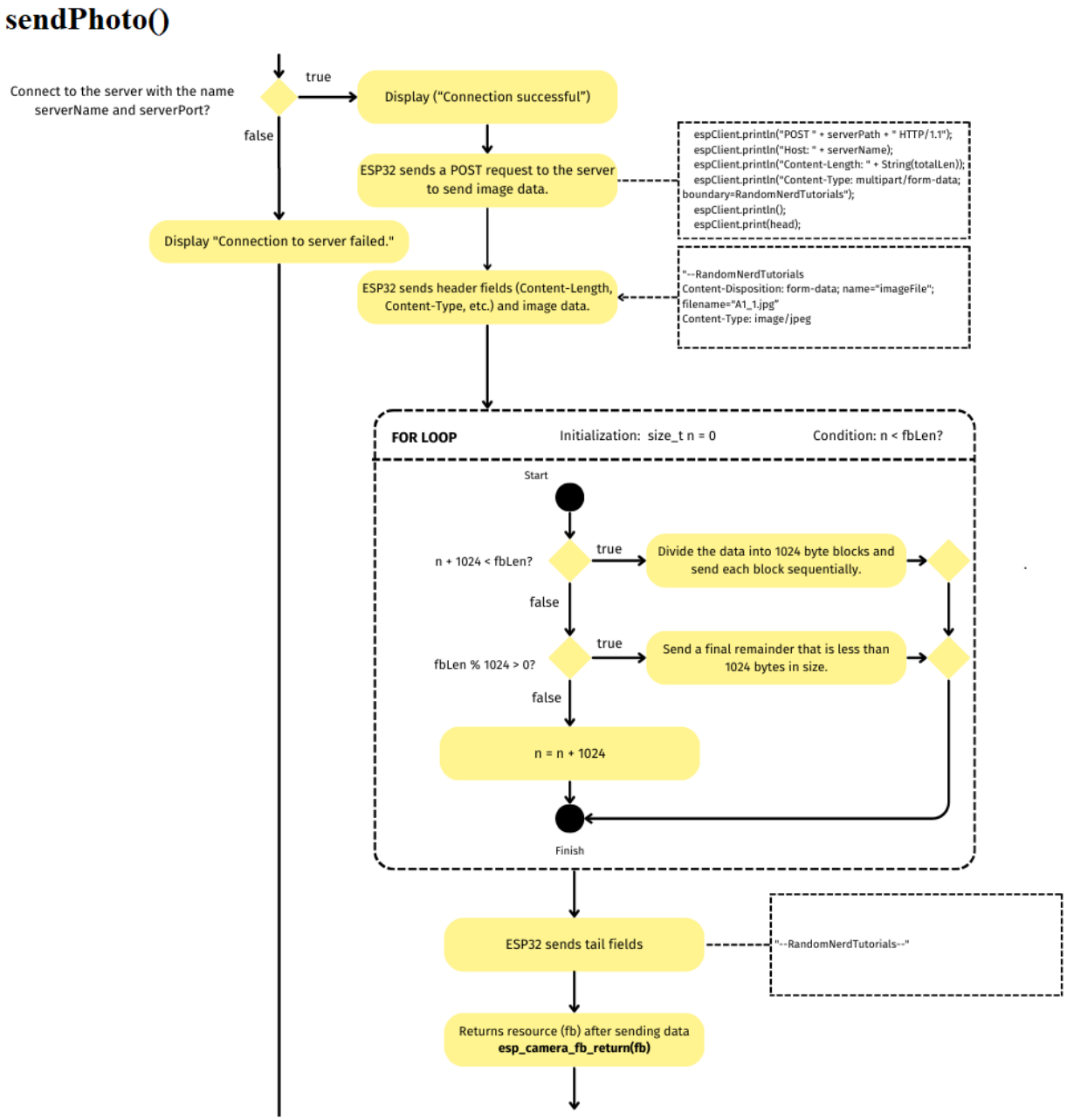


Figure III.2.6: *send_photo ()* algorithm diagram (2)

The ESP32 CAM performs the process of sending image data to a server via HTTP protocol. First, the ESP32 CAM device connects to the server through a specific port. If the connection succeeds,

it prepares the image data and necessary header fields to send. Otherwise, it displays "Connection to " + serverName + " failed."

This process involves creating information fields about the image data such as file format, data length, and other headers to identify the data.

Next, the ESP32 CAM performs data transmission from a buffer to the server in 1024 byte or smaller data chunks. In this process, a loop is used to divide the data in the buffer into small data blocks and send them sequentially to the server via the transport protocol.

First, the loop checks and sends data chunks of exactly 1024 bytes if the buffer size allows. If the buffer has less than 1024 bytes left, this remainder will also be sent to ensure complete and accurate data transmission.

This process helps split up the data for sending in individual blocks, ensuring efficiency and reliability during data transmission. The server receives and processes these data blocks sequentially, providing flexibility in handling data received from the client (ESP32 CAM).

When data transmission is complete, the ESP32 CAM finalizes the sending and returns camera resources after finishing sending data. This involves freeing memory or camera resources after using them to capture and transmit images.

The program starts by initializing variables and setting initial values. It sets a timeout period and starts calculating the start time using the *millis ()* function.

During operation, the program executes a loop to check the timeout period and read data from espClient. In each loop, the ESP32 CAM prints a "." and waits 100ms. The code checks for available data from the server via *espClient.available ()*. If there is data, the ESP32 CAM reads each character from espClient and processes it.

If the program receives a newline character '\n', it starts processing the data. If in waiting state (*state == true*), the program saves data into the getBody variable. At the same time, it updates the start time (*startTimer = millis ()*) to reset the timeout period.

If the getBody variable has data (*getBody.length ()* > 0), the program exits the loop. After that, the program stops the espClient connection with the server (*espClient.stop ()*).

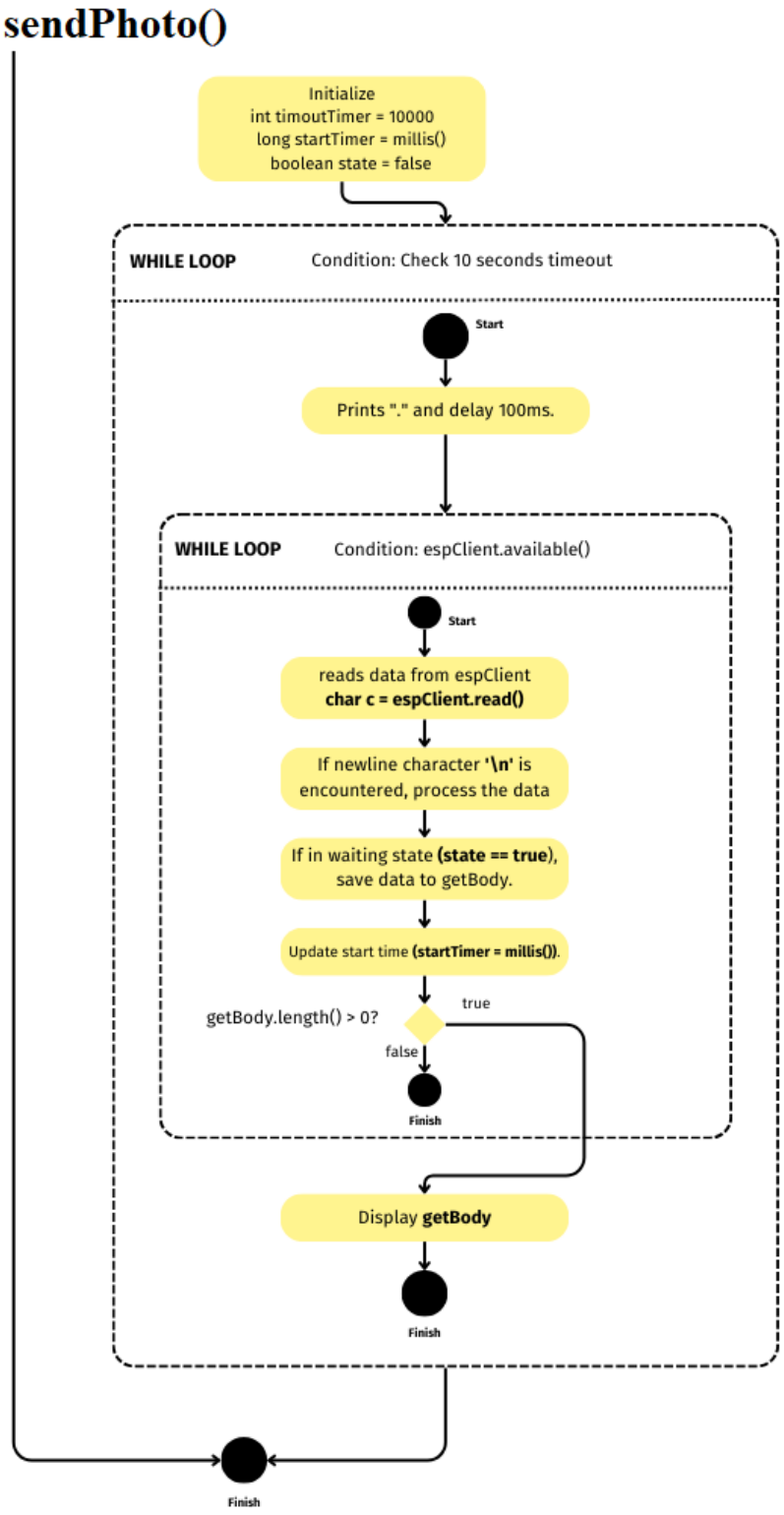


Figure III.2.7: send_photo () algorithm diagram (3)

2.3.1.4 Alert Users if Vehicle License Plate Image Not Captured

The *alert_to_user()* function is utilized to notify either the user or the system when there's a failure in reading the license plate. The system will signal the user by sounding a buzzer, enabling them to park the car correctly for the system to capture the license plate.

At this step, the program checks two conditions. First, it checks if the MQTTmessage variable equals 1 and checks if the carDetection variable is currently true. This is equivalent to checking if there is a notification signal from the Raspberry Pi by reading the wrong license plate and the variable checking if the car is in the parking lot is currently true or not. If both of these conditions are true, the program proceeds to send notifications and images via the *sendPhoto()* function.

Next, the alarm will be activated and the image recapture counter will be increased. If 4 images have been taken but the license plate still cannot be read, the program will send a notification to the Raspberry Pi to update the information on Firebase. This information will be used by the admin to monitor and handle.

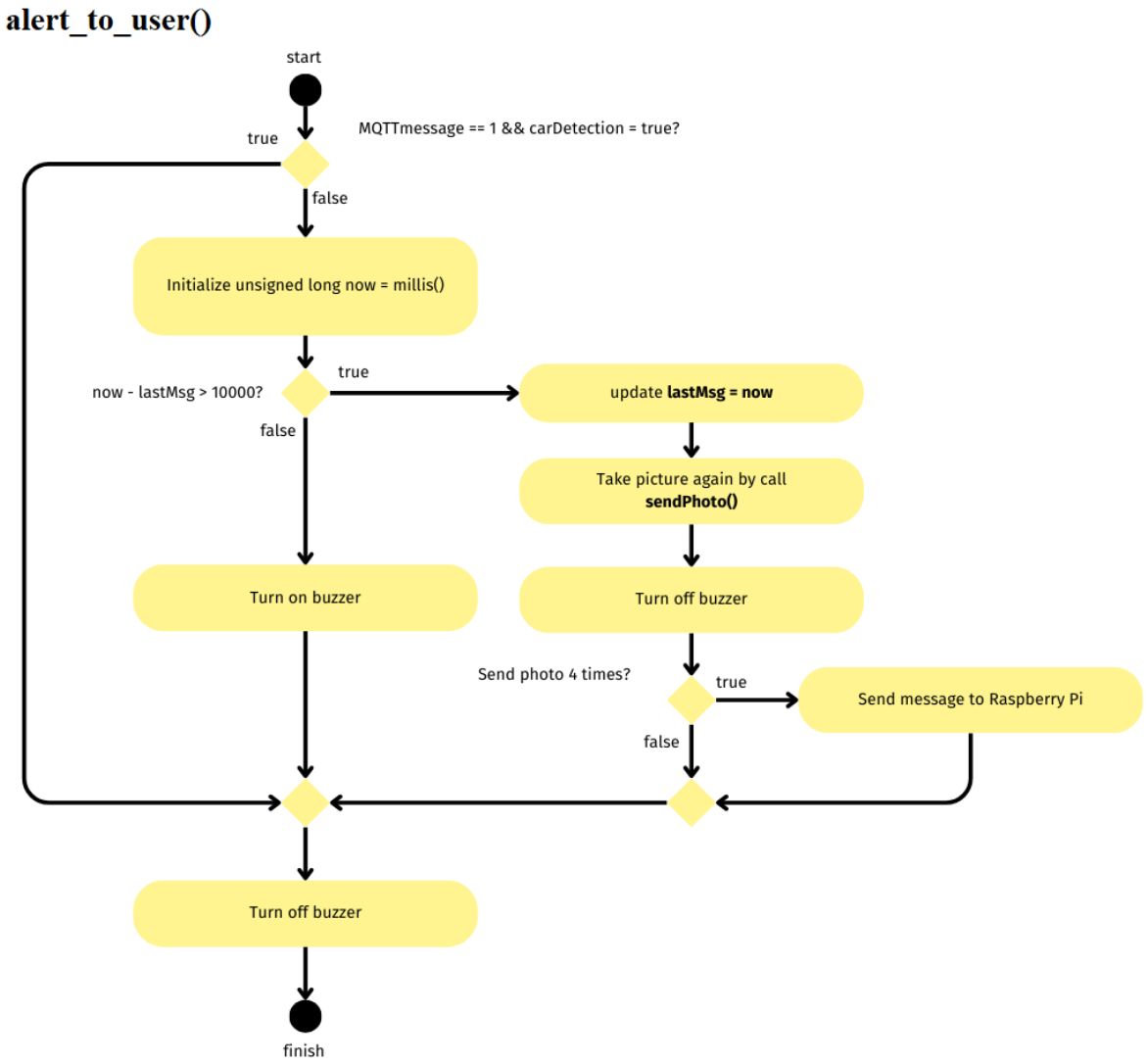
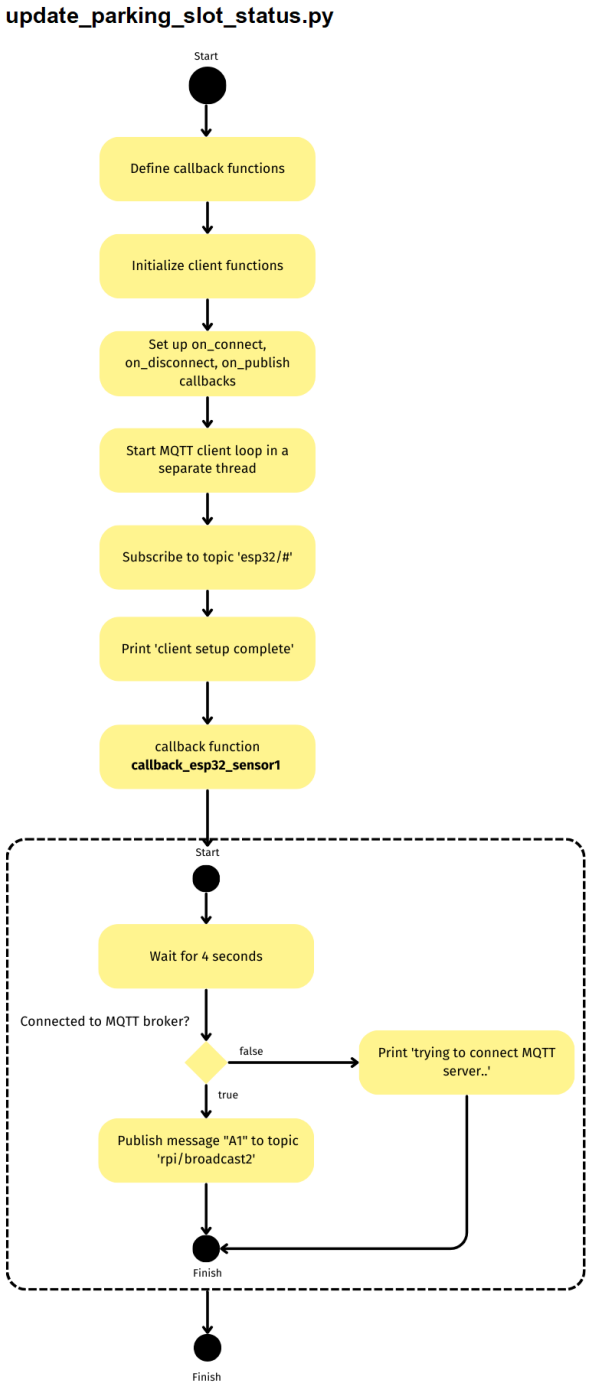


Fig III.2.8: *alert_to_user ()* algorithm diagram

2.3.2 Raspberry Pi 4

2.3.2.1 Communicating with ESP32 CAM via MQTT

This diagram illustrates the interaction between a client and MQTT broker involving connecting, sending receiving messages, and data processing via callback functions.



The interaction process between client and MQTT broker begins when client initializes callback functions and establishes connection to the MQTT broker. Client sets up callbacks like on_connect, on_disconnect, on_publish, as well as callback_esp32_sensor1 for topic 'esp32/A1'. Then, client attempts connecting to MQTT broker at address '127.0.0.1' and port 1883.

The client starts a continuous loop, waiting 4 seconds before checking connection status to MQTT broker. If not successfully connected, client prints 'trying to connect MQTT server…'.

In the program, there is an important function which is callback_esp32_sensor1. When the client receives a message from the MQTT broker, the callback callback_esp32_sensor1 is triggered. In this callback, the client decodes the message from the payload of the received message and process the data to prepare for updating the Firebase at the path 'parking/Alpha/A1'.

Fig III.2.9: The Flowchart of update_parrking_slot_status ()

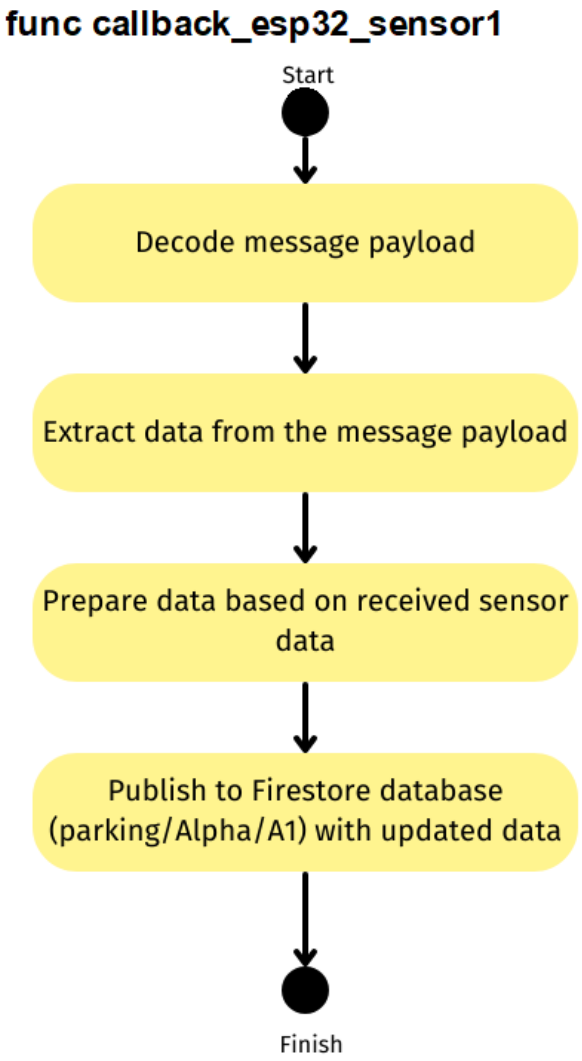


Fig III.2.10: The flowchart of callback_esp32_sensor1

If the Raspberry Pi receives a notification that a vehicle has entered the parking space, it will update the parking lot status to occupied and record vehicle entry time in Firebase corresponding to the parking position. When the vehicle leaves, Raspberry Pi will update exit time information and log it to Firebase.

Then, it publishes information with 'topic' "rpi/broadcast2" sent to update_user_parking_ history.py to update parking history into users' data

2.3.2.2 Updating Parking Slot Data to User Parking History

The diagram describes the interaction between a client and an MQTT broker, showcasing the process of connection, message transmission, reception, and data processing via callback functions. The primary objective of the mentioned program is to update users' parking histories on Firebase by recording entry and exit times along with parking locations

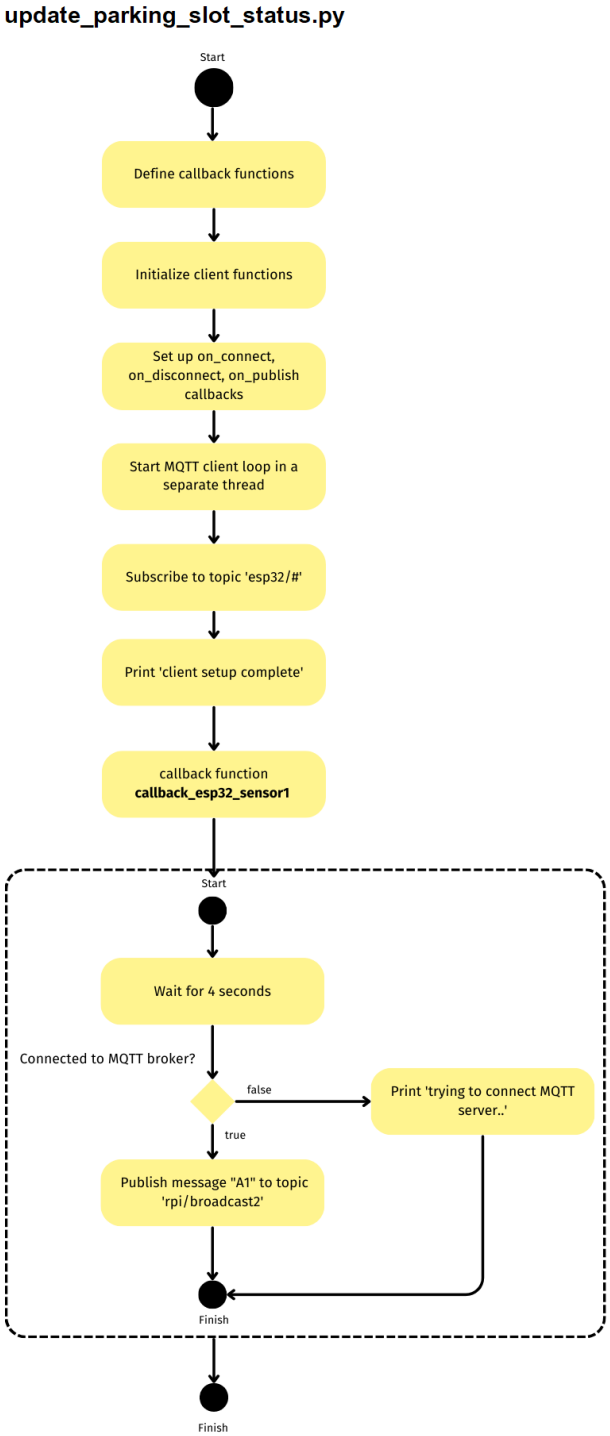


Fig III.2.11: Flowchart of function *update_user_parking_history.py*

MQTT client is initialized with name "rpi_client3" and *flag_connected* initially set to 0. *Client.on_connect* is assigned to connect event (*on_connect*), when connected successfully, *flag_connected* is set to 1 and client subscribes to monitor topic "*rpi/broadcast2*".

When MQTT connection succeeds, flag_connected updated to 1 and client subscribes to monitor topic "*rpi/broadcast2*". A "Connected to MQTT server" message is printed.

In the loop, the program waits 4 seconds and checks if MQTT connection is disconnected. If disconnected, "trying to connect MQTT server…" message is printed.

When receiving a message from topic "*rpi/broadcast2*", client handles message by checking and printing message content. If the message starts with "A", client queries Firebase to get parking data from a location A1. Parking data is retrieved and sent to *update_parking_time_to_user_history* function to update to Firebase.

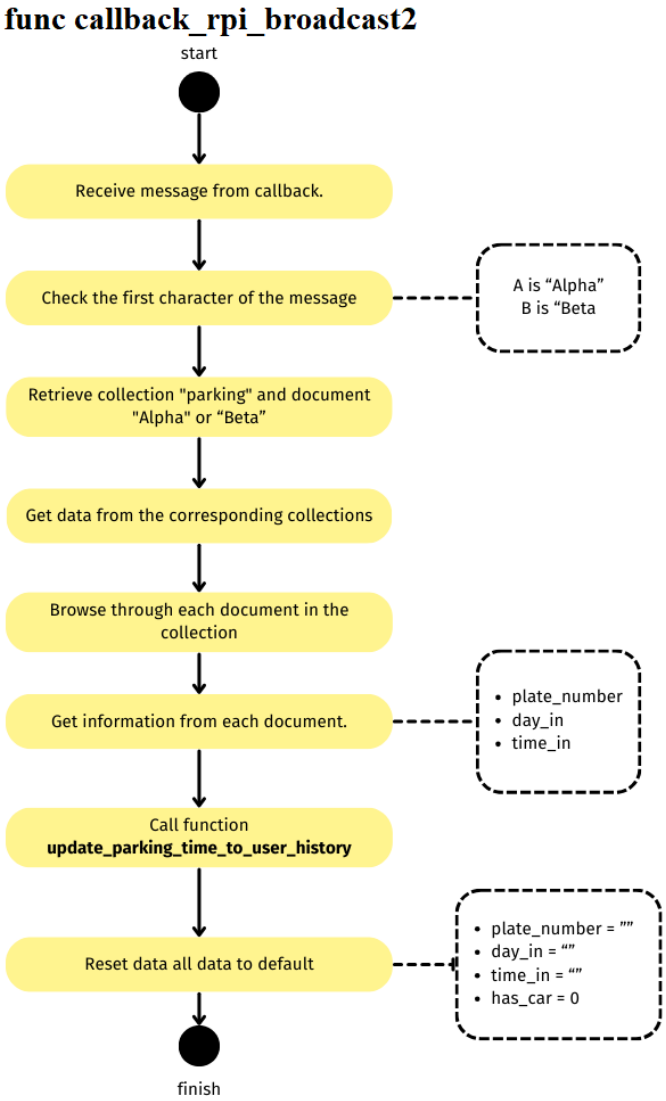


Fig III.2.12: *func_ callback_rpi_broadcast ()* algorithm diagram

The *callback_rpi_broadcast* function is a callback called when receiving a broadcast event from Raspberry Pi. First, this function converts the payload content of the message from bytes to UTF-8 string and displays the message content on the console.

Next, it checks if the first character of the message is "A" or “B”. If "A" it will correspond to “Alpha” document on Firebase, the function will query collection "parking" from Firebase, using document "Alpha" and collection with name corresponding to the message.

Then, for each document in that collection, the function gets information like license plate, day in, day out, time in, time out and updates this information to the user's parking history via the *update_parking_time_to_firestore_with_plate_number* function.

After retrieving data and updating user history information to Firebase. The data at the parking location is reset to default.

The update_parking_time_to_user_history function is called to update parking data into user database.

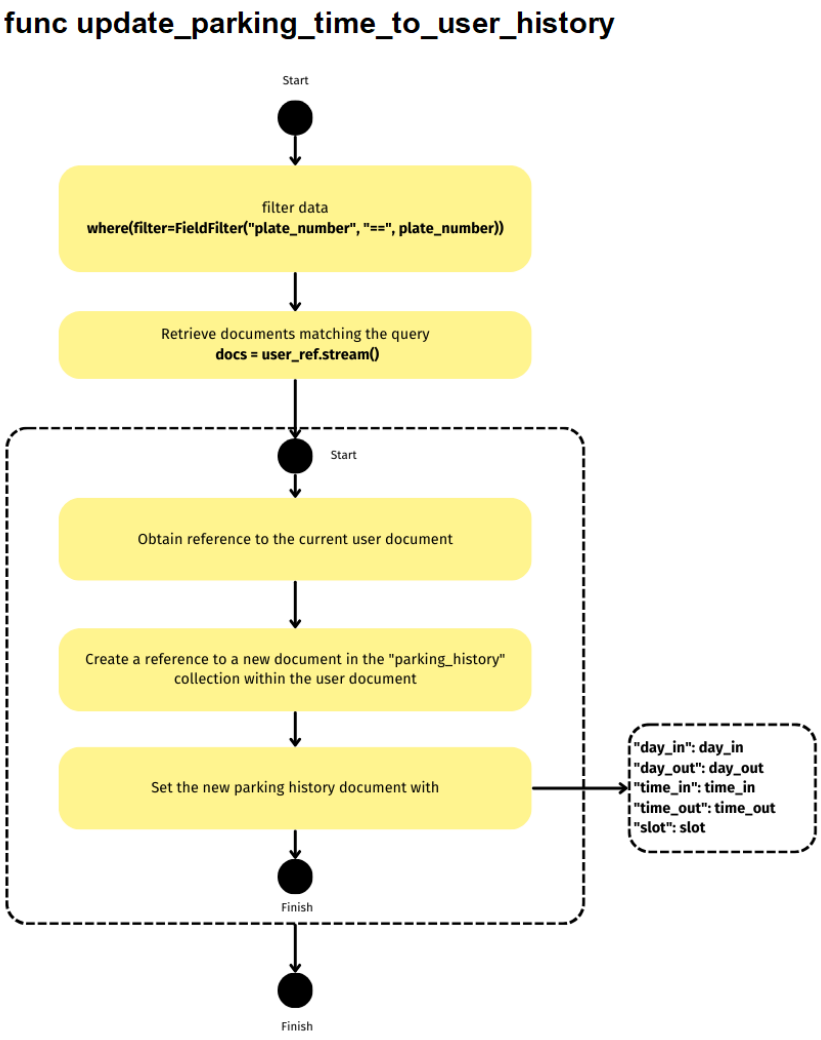


Figure III.2.13: Flowchart of function update_parking_time_to_user_history

2.3.2.3 Process images to identify license plates and update corresponding information into the Firebase

This program process images to identify license plates and update corresponding information into the Firebase.

main.py

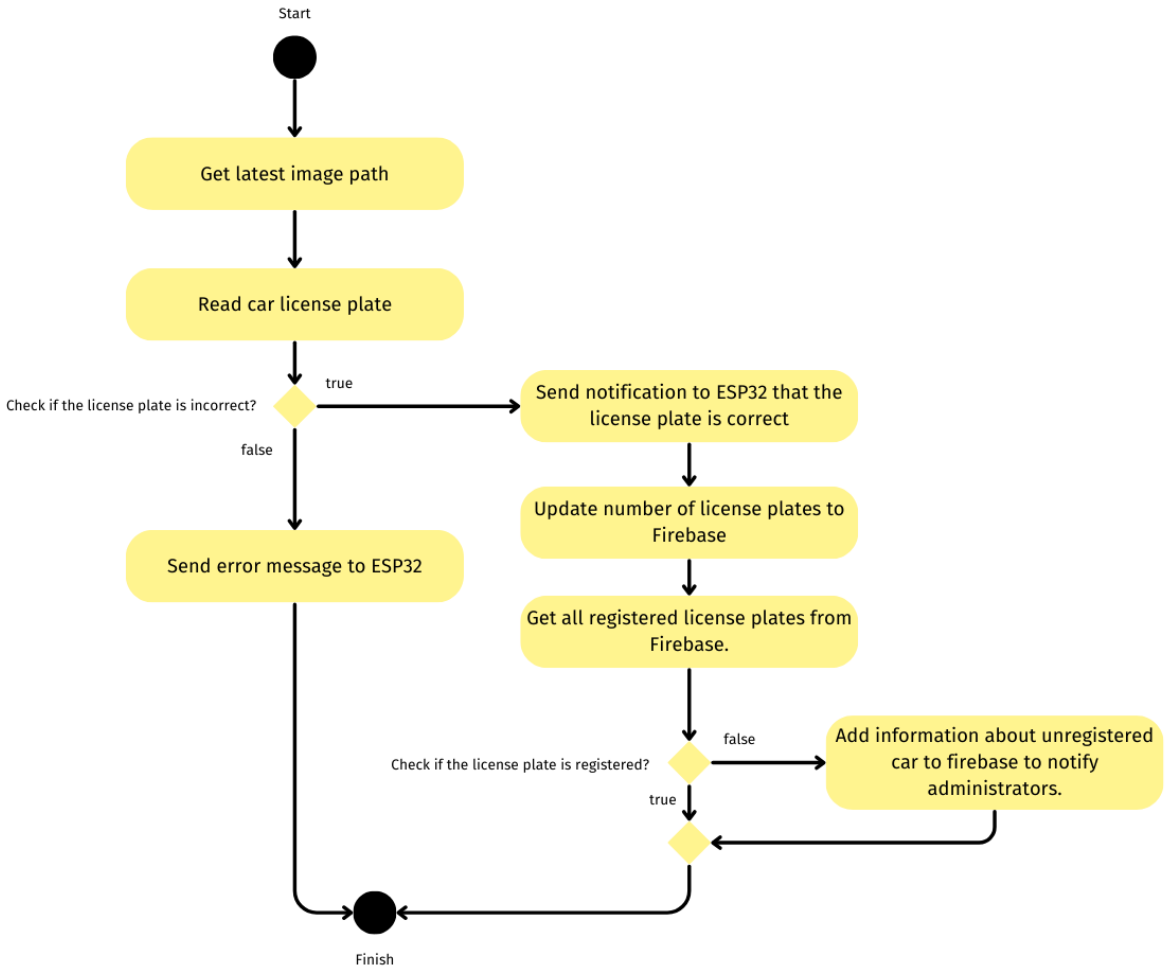


Figure III.2.14 Flowchart of function *main.py*

This code section performs image processing to recognize license plates, updates information to Firebase, and notifies processing states to the ESP32 device and admin when there is new data or errors. First, the program starts by getting the file path of the latest image via the *get_lastest_img _path ()* function for further processing with that image. Then, through the *read _and _extract _ plate.read_plate_number (image_path*) function, the program extracts license plate information from that image and checks the accuracy of the license plate (*is_wrong_plate*).

If the license plate recognition process is inaccurate, a notification about the recognition error is sent back to the ESP32 device via the *send_data_to_esp (is_wrong_plate)* function and the “wrong plate” message is printed out.

In case of successful license plate recognition, the correct recognition notification is also sent to the ESP32 device and the “right plate” message prints out. Additionally, vehicle information is updated to Firebase via the *update_car_data_to_firestore (license_plates)* function.

Next, the program checks if the vehicle’s license plate is already saved in the Firebase by comparing with the list of all existing license plates (*db_plate_number = get_all_plate_from_firestore ()*). If the license plate does not exist in database, the unregistered vehicle information is added into Firebase data via *update_car_not_register (license_plates)* function to notify the admin.

2.3.2.4 Raspberry receiving incoming images and store it

This PHP program is responsible for receiving incoming images from the ESP32-CAM, rename the images with a timestamp and store them in the uploads folder.

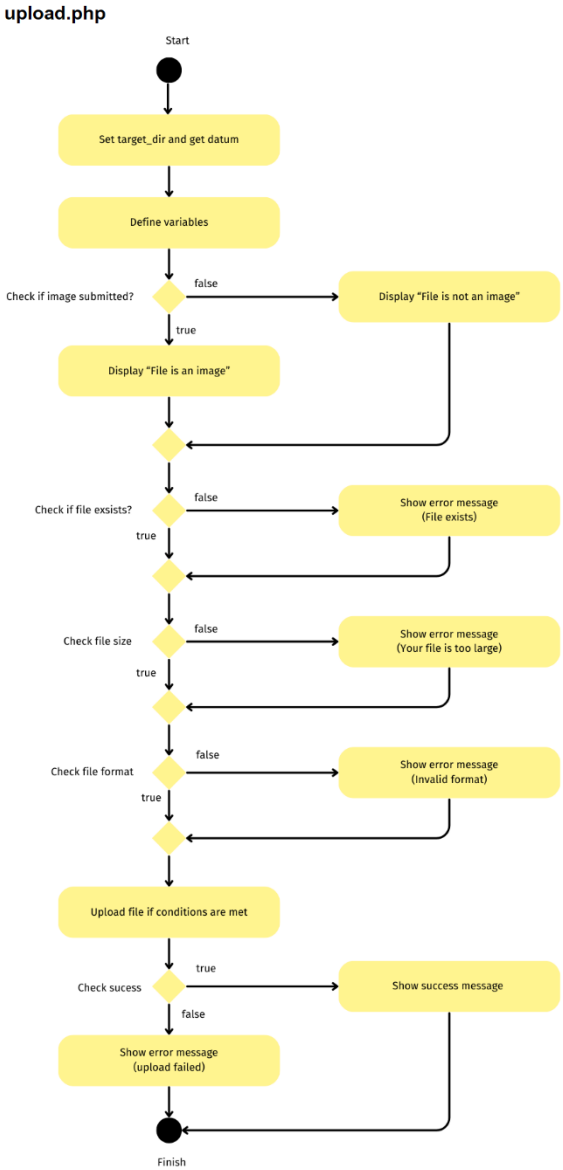


Figure III.2.15.: workflow of *upload.php* diagram

The *upload.php* program is responsible for handling image file uploads in the PHP code segment.

It first initializes the "target_dir" and "datum" variables to store destination path and timestamp information. The next step defines other variables related to file handling.

Next, the program checks if an image file has been uploaded. If the image file was uploaded, the program proceeds to check if that file is actually an image. If it is an image, the program will print “File is an image” and continue to confirm this is a valid image file.

After that, the program checks if the image file already exists in the uploads folder. If the file exists, the program will print “Sorry, file already exists”.

Following that, the program checks image file size. If the file is too large (exceeds 500,000 bytes), the program will print “Sorry, your file is too large”.

It then checks the image file format to determine if it is a valid image format (JPG, JPEG, PNG, or GIF). If not, the program prints “Sorry, only JPG, JPEG, PNG & GIF files are allowed”.

Finally, if all above conditions are met (valid image file, not duplicate, appropriate size and format), the program proceeds to upload the image file into the uploads folder. If upload succeeds, the program displays the message “The file [filename] has been uploaded.” If not, the program prints “Sorry, there was an error uploading your file.”

2.3.2.5 Displaying Images on the gallery

The function starts by setting header and title for the webpage. Then, the program continues setting styles and layouts with CSS. Next, the web browser displays title and introductory information about the gallery page.

After showing the information, the program continues handling the HTTP GET request to check for any image deletion requests. If there is a request, the program checks file format and existence in the storage folder. If the image exists and has correct format, it will be deleted from the folder and display “File found and deleted” message.

Next, the program checks existence of image storage folder. If the folder exists, it displays images in flexible (flex) layout by looping through each file in the folder. Each file is displayed with a “Delete file” link to delete the file and show filename.

If there is no image in the folder, the program displays “No images found” message.

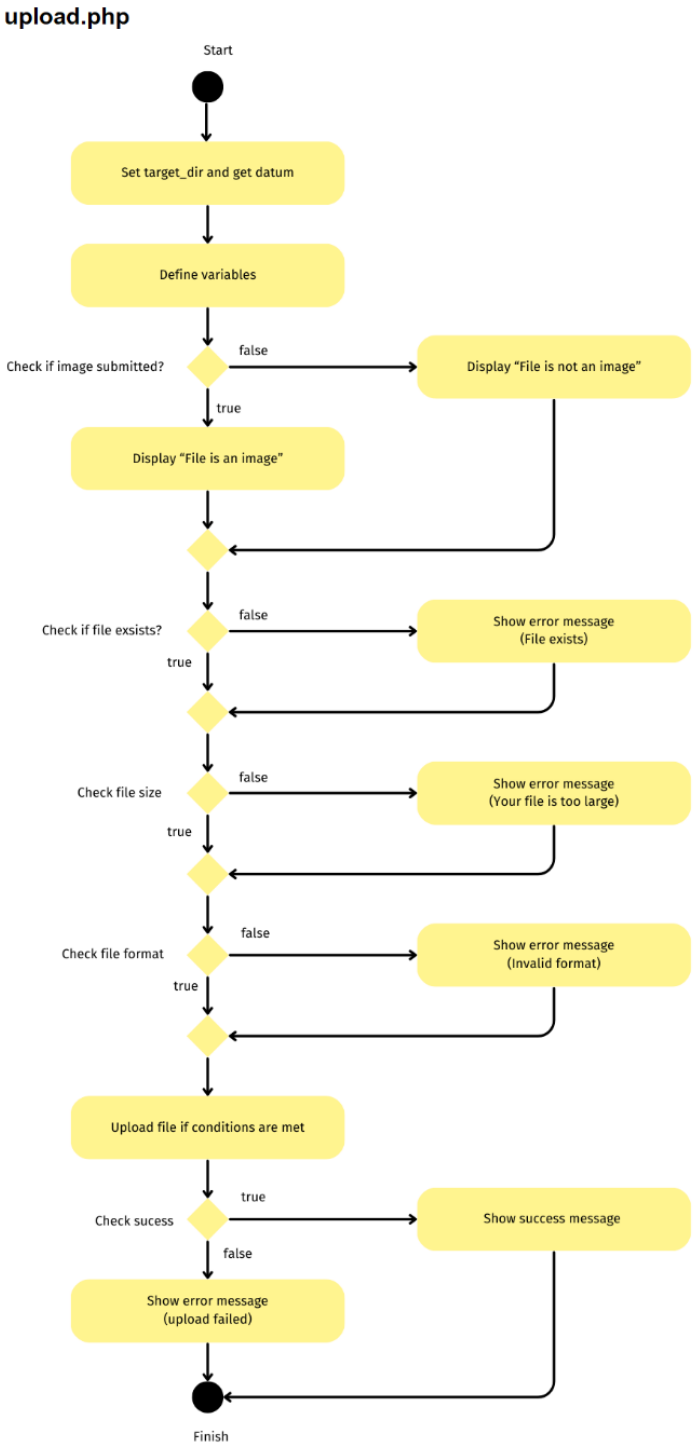


Figure III.2.16: Workflow of *upload.php* diagram

2.3.2.6 Track changes in directories

This PHP program is responsible for tracking changes to the "uploads" folder. When a new photo is added to this folder, the associated Python file will execute to extract and analyze the license plate.

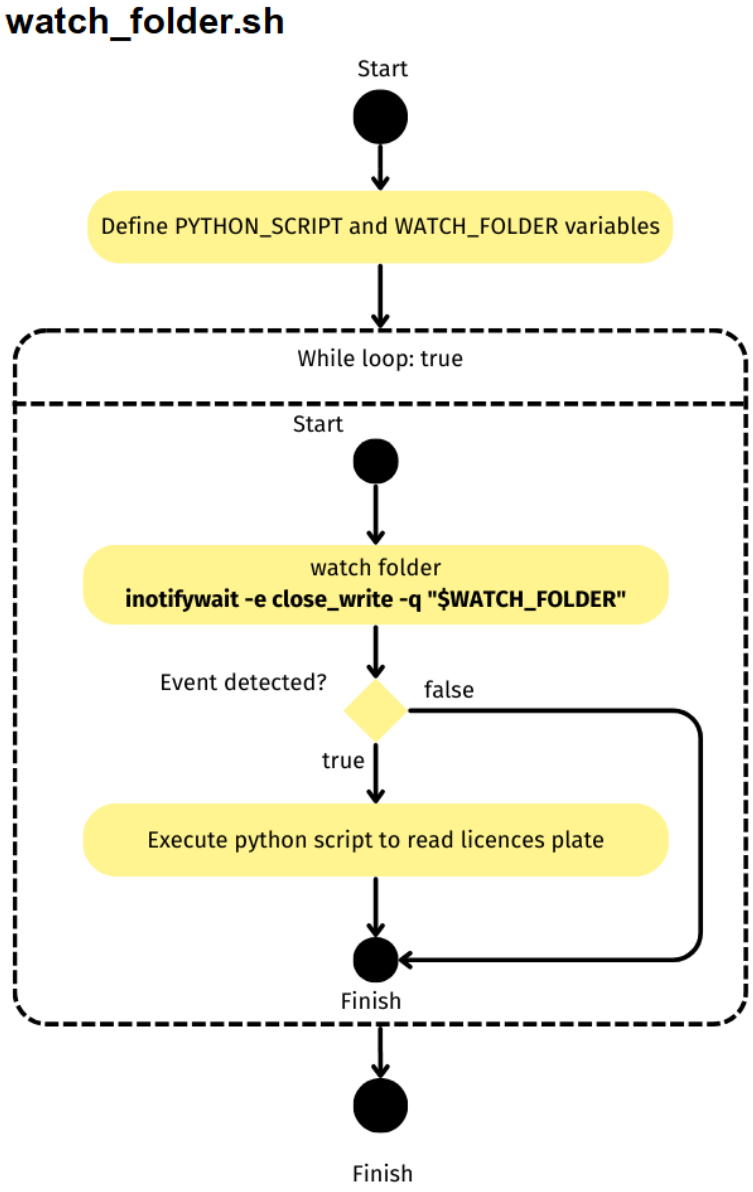


Figure III.2.17: workflow of *watch_folder.sh* program

It first starts and initializes two variables: PYTHON_SCRIPT containing path to the Python script, and WATCH_FOLDER containing path to the folder to watch.

After that, the program starts an infinite loop. Inside this loop, the program uses the *inotifywait* command to wait and catch “close_write” event inside the folder specified by WATCH_FOLDER variable.

When a change event is caught, the program continues by triggering the Python script specified by PYTHON_SCRIPT variable, using the Python 3 interpreter After running the Python script, the program loops back and continues watching for changes in the folder, repeating the process.

This creates a continuous cycle: the program watches for changes in the folder and when a change event occurs, runs the specified Python script, looping this process indefinitely.

2.4 Design and deploy system

2.4.1 Block Diagram

The diagram below illustrates the comprehensive framework of a smart parking system. This block diagram comprises two main sections: the left portion depicting the client device, and the right section representing the host device.

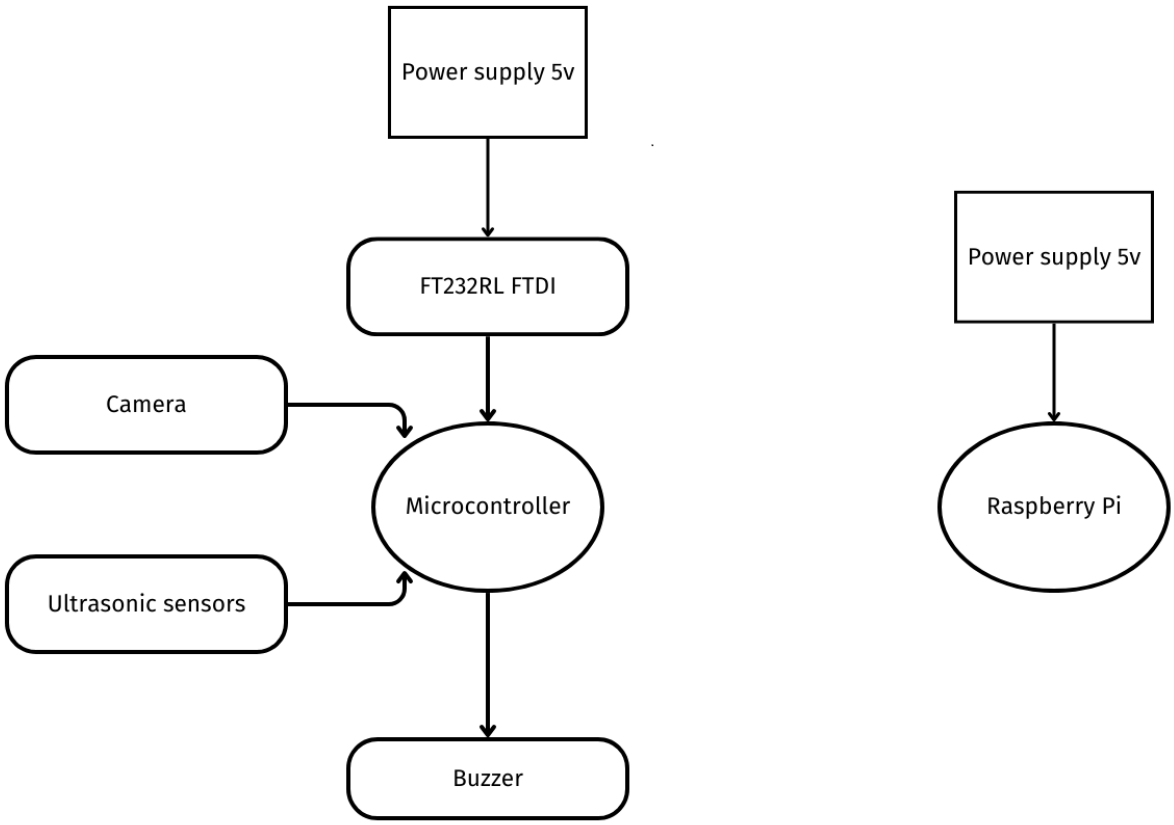


Figure III.2.18 System Block Diagram

The device is powered by a 5V power source and utilizes the FT232Rl FTDI IC for establishing a serial communication port (UART). This facilitates the transmission of data, commands, or interaction with embedded microcontrollers. Apart from its communication functionality, the FT232RL FTDI also manages power and data transmission between the computer and the microcontroller. Ultrasonic sensors are employed to gauge the distance between the sensor and an object, thereby detecting the presence of a car in a parking spot. Additionally, a camera is deployed to capture photos whenever the ultrasonic sensors detect a car. Subsequently, the microcontroller

will activate a buzzer upon receiving a signal from the host. The host part will be processed by a Raspberry Pi and powered by a 5V power source. Its function involves collecting signals and images from the client side for processing and delivering necessary directives back to the client.

2.4.2 Connection Diagram

The following connection diagram will assist in the design process by outlining the necessary connections and providing an overall architecture of the system.

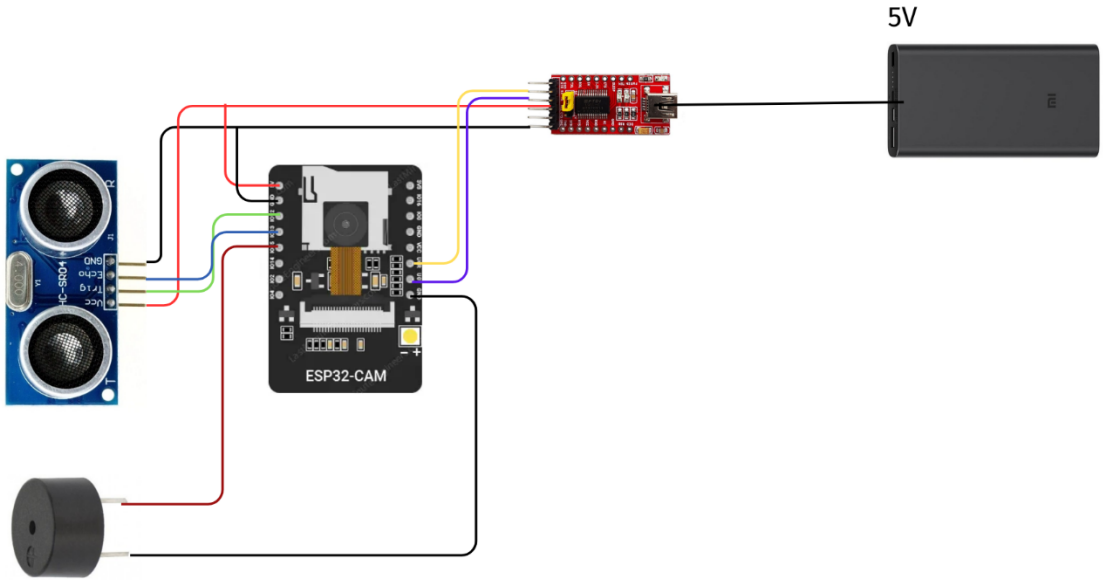


Figure III.2.19: ESP32 CAM connection diagram

The image demonstrates the wiring connections for the ESP32 CAM, HCSR04, Buzzer, and IC FT232Rl FTDI devices. The ESP32 CAM typically has limited available port pins due to most being allocated for the camera. Usually, only pins IO2, IO4, IO14, IO15, IO12, IO13, and IO16 are left unused. Among these, IO12 and IO13 will be utilized to link to the HC-SR04 ultrasonic sensor's Trigger and Echo pins respectively, while IO15 will be allocated for the buzzer to provide notifications. Although not depicted in the diagram, the IO4 pin is employed to activate the built-in flash whenever a photo is captured. For the IC FT232Rl FTDI, connections are established via the TX and RX pins.

The entire system will be powered by a 5V power source supplemented by a backup battery. Additionally, in figure 2.4.3, the Raspberry Pi also operates on a 5V power supply facilitated through an adapter converting the main 220V power source to 5V.

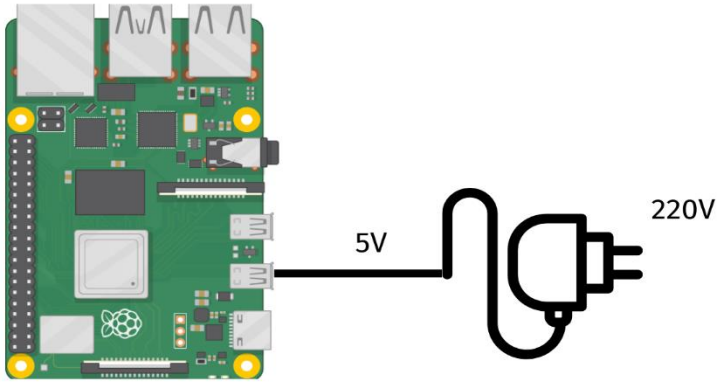


Figure III.2.20: Raspberry Pi Connection Diagram

2.4.3 Communicate with Raspberry Pi and ESP32 Cam via MQTT Protocol

It can be seen the Raspberry Pi running an MQTT broker and containing three python client scripts (update_user_parking_history.py, update_parking_slot_status.py and main.py). The ESP32 device utilizes its own C code to function as a client. Information exchange between clients occurs through 'topics', denoted in blue. Inward arrows into a 'topic' denote a client publishing data to that topic. Outward arrows represent a client having subscribed to the topic, consuming published information.

Within the Raspberry Pi program, update_parking_slot_status.py and update _user _parking _ history.py continuously execute to subscribe for updates from other clients.

Whenever a vehicle parks in a slot, the ESP32 CAM publishes data to the respective esp32/A1 topic (e.g., spot A1). The update_parking_slot_status.py script subscribes to receive this information. Upon intake, it updates parking status and logs the vehicle's arrival time to Firebase.

For main.py function, after reading and extracting a license plate, it publishes to the “rpi/broadcast” topic to report successful recognition. The ESP32 Client subscribes to get this data and triggers a buzzer if the read fails.

Similarly, when the vehicle departs, the ESP32-CAM publishes another update which the Raspberry Pi receives. The update_parking_slot_status.py script updates the slot status and records the exit time, also publishing a packet to the “rpi/broadcast2” topic.

The update_user_parking_history.py function further subscribes to this data to retrieve usage details, updating per-user parking histories in Firebase based on license plates.

The following diagram illustrates the workflow between a Raspberry Pi and an ESP32 CAM using the MQTT protocol. This communication occurs through subscribing to and publishing information across different topics.

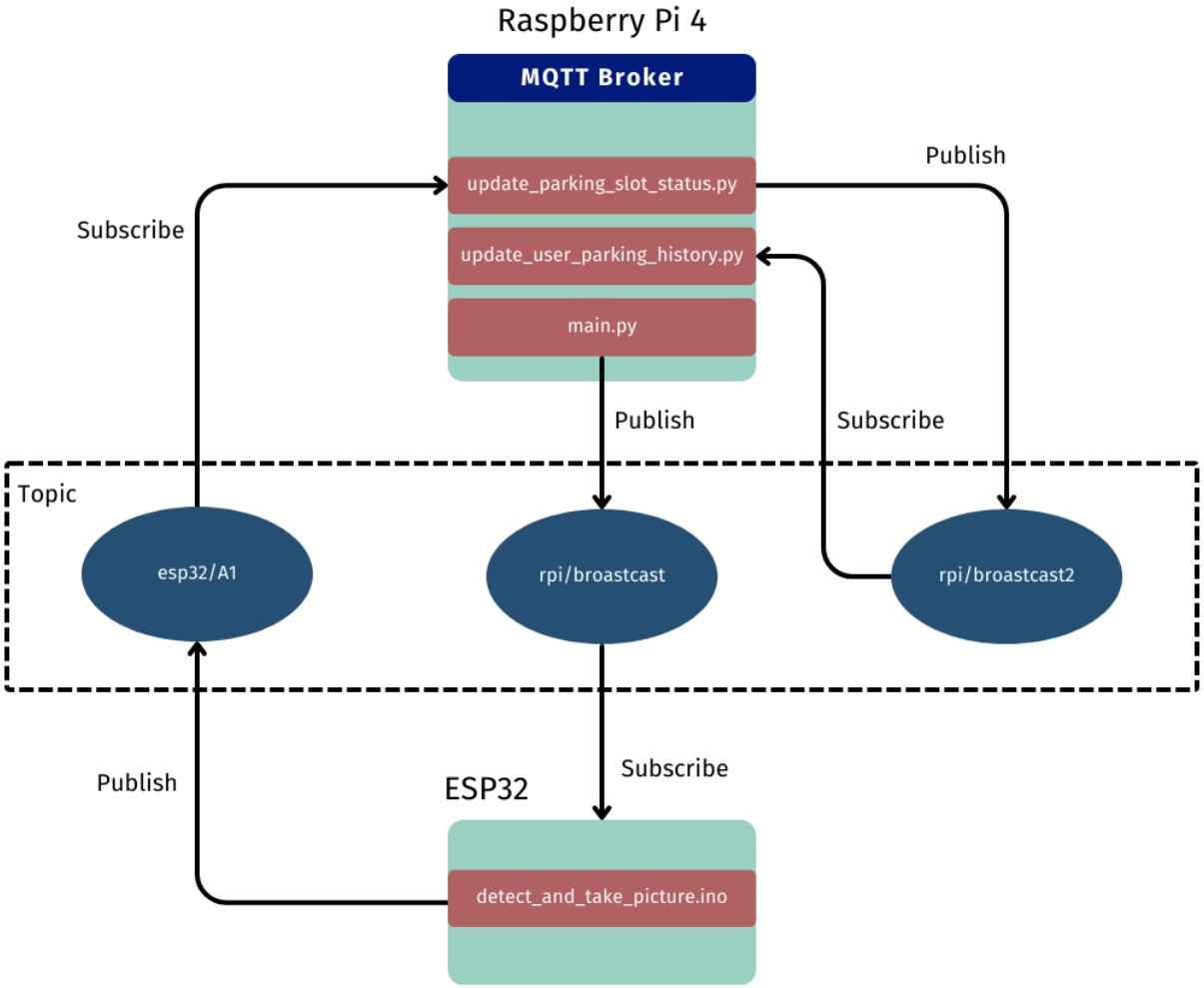


Figure III.2.21: The Information Flow Diagram

2.4.4 Raspberry Pi Local Server

The ESP32 Cam and Raspberry Pi 4 are connected to each other via a LAMP server configuration to perform the process of managing and displaying images from the ultrasonic sensor and ESP32-CAM.

The operation process begins when the ultrasonic sensor on the ESP32-CAM detects the presence of a vehicle. When this signal occurs, the camera on the ESP32-CAM will take a picture and send it to the Raspberry Pi 4 via the HTTP POST method.

On the Raspberry Pi 4, a website is set up with a LAMP server to receive images sent from the ESP32-CAM via the /upload.php file. Here, the received images are processed by renaming them based on their timestamps and then stored in the "uploads" folder.

In parallel with the image storage process, the /gallery.php page on the Raspberry Pi 4 is responsible for displaying these images in a gallery interface. Users can access this page to view the images captured from the ESP32-CAM, creating a user-friendly interface for managing and viewing captured images.

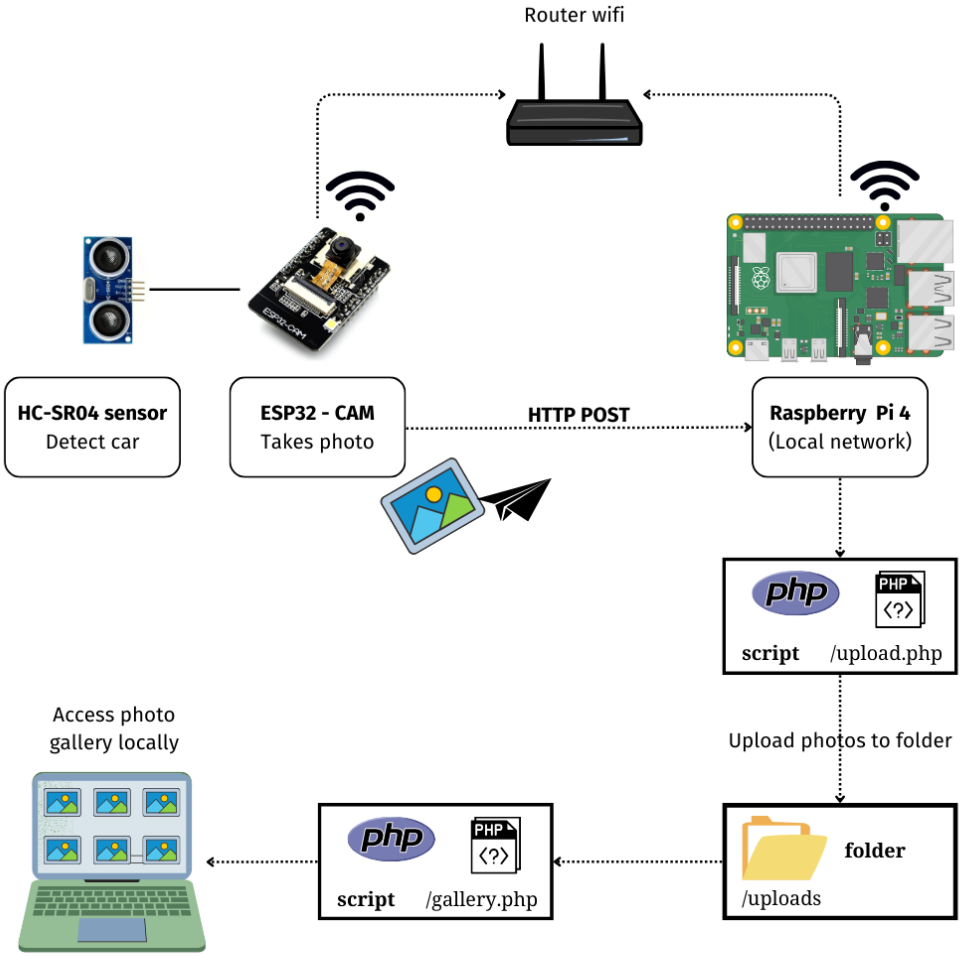


Figure III.2.22: Raspberry Pi LAMP Server

2.5 Artificial Intelligence Design for License Plate

2.5.1 Vehicle License Plate Concepts

Vehicle license plates, also called vehicle registration plates, are metal plates attached to every motor vehicle. They get officially issued per vehicle by local police authorities when purchased or transferred. Typically made of aluminum alloy, plates come in a rectangular or near-square shape, stamped with numbers and letters displaying owner information. The region, locality, specific digits and lookups on computers can even derive identity of registrant, purchase dates, and facilitate security processes.

Standard Dimensions:

Current auto license plates in Vietnam have standardized dimensions with two primary types:

Short plates: Height 165mm, Width 330mm

Long plates: Height 110mm, Width 520mm

Plates contain around 7-9 characters, with individual character height of 80mm and width of 40mm.

Given these traits, relevant parameters can define filters to selectively detect plates and characters of interest for targeted applications.

2.5.2 Overview of Vehicle License Plate Detection

License plate recognition involves automated or semi-automated computer technology to identify and extract information from vehicle license plates. The process typically comprises:

1) Image Capture
2) Image Processing
3) Localizing and Segmenting the License Plate
4) Character Segmentation on License Plate
5) Character Recognition

Additional capabilities beyond plate recognition may include:

Information Extraction: After successful recognition, relevant license details like the license number, vehicle type, location and timestamp may extract.

Data Cross-Referencing: The extracted information often cross-checks against databases to identify vehicles linked to traffic violations or special scenarios.

Reporting and Data Storage: The processing outputs may log and archive in databases for downstream usages like accident causation analysis or traffic pattern monitoring.

2.5.3 Vehicle License Plate Detection

2.5.3.1 Image Capture

To capture license plate images from fixed roadside CCTVs, we focus on selecting frames with plates per the following characteristics:

Parking Alignment and Orientation: Parked vehicles typically align near lane dividing markers in parallel with road paint lines. Cameras mount at perpendicular angles to license plate surfaces. Such positioning enables optimal quality image capture.

Shooting Angle and Camera Adjustments: Tune the camera optical axis to an approximate perpendicular angle against license plate planes. This optimization maximizes recognition and readability within the captured footage.

Image Controls and Filtering: Given position and angle constraints, necessary parameters can define to isolate license plate regions. Possible measures include brightness thresholds, color balancing, or image processing techniques to extract plate-containing subsets.

Figure III.2.5.1: Cars in Car Park

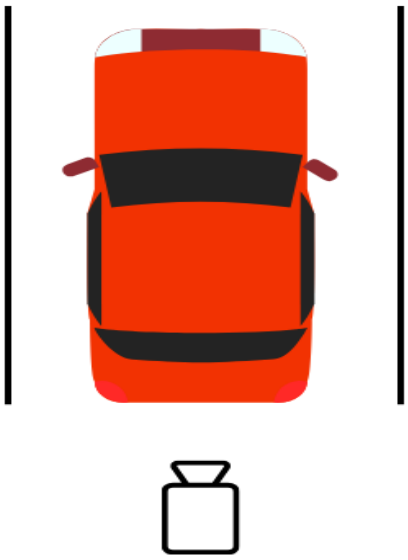


Figure III.2.5.2: Camera Set up

Together, considerations around camera alignment, scene perspective, and programmatic filtering allow reliably extracting vehicle license plate shots for further analysis.

2.5.3.2 Image Processing

To enable effective license plate recognition, key image quality factors include:

- Resolution: Higher resolutions provide more image detail but require greater computational resources. Selecting an optimal balance between quality and processing needs is crucial.

Figure III.2.5.3: Low Resolution Picure of a vehicle license plate

Figure III.2.5.4: High resolution picture of a vehicle license plate

- Lighting and Contrast: Light and contrast levels impact pre-processing requirements. Images lacking adequate brightness or contrast have less clarity and definition.

Figure III.2.5.5: Bright License Plate

Figure III.2.5.6: Dark License Plate

- Noise: Noise in image can reduce processing efficacy. This cause by environmental interference or extraneous scene components.

Figure III.2.5.7: Noise in Image

- Shooting Angle: Perspective impacts object rendering and visibility. Non-perpendicular angles may distort shapes and omit critical portions like license plates.

a b c

Figure III.2.5.8 a, b, c: Different Shooting Angle Vehicle License Plate

Thus, near ninety-degree capture directly facing the plate isolates it with optimal positioning for recognition.

Vehicle body height and plate mounting varies. Determining ideal camera distance and elevation for precise number plate shots remains vital.

By averaging measured ground-to-plate heights across various car makes, a mean plate location height calculates to approximate camera positioning. This standardized view focusing specifically on the license plate allows optimized imaging.

Figure III.2.5.9: Car

For each type of car, they have distinct sizes and shapes. Therefore, the license plate height and positioning will vary between models. It is critical to identify the optimal distance and angle between the camera and license plate to accurately capture the plate number.

The methodology is to measure the distance from the ground to the location of the license plate, then take the average of these values. This provides an estimate of the average distance from the ground to the license plate position across various vehicle types.

Below are some measured ground-to-plate distances for example vehicles. However, it should be noted car park differently, introducing slight measurement error to license plate.

Car	Front Plate		Rear Plate	
Sedan (Kia)	42	53	32	48.5
SUV (Toyota cross)	40	56.5	100	111

Sedan (Camry)	40	51	76	92.5
Sedan (Honda)	38	49	78	94.5
SUV (Kia Sorento)	46	57	83	96.5
Sedan (Kia morning)	43	54	33	49.5
Sedan (Hyundai)	42	53	76	92.5
Sedan (Camry)	40	51	68	84.5
SUV (Honda)	45	61.5	83	96.5
SUV (Peugeot)	44	60.5	34	50.5
Sedan (Audi)	36	52.5	72	88.5
SUV (Mitsubishi)	43	59.5	68	84.5
SUV (Hyundai)	40	56.5	78	94.5
SUV(Hyundai)	46	57	86	102.5
SUV (Mitsu)	49	60	87	103.6
Sedan (Volvo)	36	47	75	91.5
Sedan (Suzuki)	36	52.5	39	55.5
Sedan (Kia)	36	52.5	72	88.5
Sedan (Kia)	36	52.5	72	72-88
Sedan (Vinfast)	28	44.5	49	65.5
Sedan (Honda)	34	45	47	53.5
SUV (Vinfast)	46	57	47	64.5
Sedan (Merc)	33	44	72	89.5
Sedan (Mazda)	43	54	33	49.5
SUV (Hyundai)	46	57	86	102.5
Sedan (Hyundai)	41	52	33	49.5
SUV (Mitsubishi)	59	75.5	80	96.5
Sedan (Toyota)	42	53	69	85.5
Sedan (Mazda)	40	51	31	47.5
Sedan (Honda)	33	44	75	91.5
Sedan (Kia)	41	52	76	92.5
Sedan (Suzuki)	47	58	83	99.5
Sedan (Merces)	25	41.5	74	90.5
Sedan (Peugeot)	46	62.5	80	96.5
SUV (Hyundai)	48	59	87	103.5
SUV (Toyota)	64	75	94	100.5
SUV (Mazda)	39	50	87	98
SUV (Volvo)	46	57	57	73.5

Table III.2.5.1: Table of ground-to-plate distances for each type, brand of the vehicles

From the measurement data, we observe that front license plates demonstrate relatively consistent vertical positioning across models. However, there is far wider variance in height for rear plates between vehicles.

The calculated averages of the license plate heights relative to the ground are:

- Front Plates: 48.11842105 cm
- Rear Plates: 75.32758621 cm

Now with the average plate heights calculated, and given camera mounting height parameters. The next step is calculating the ideal camera distance ranges for capturing front and rear plates based on the variance.

2.5.3.2 Image Processing

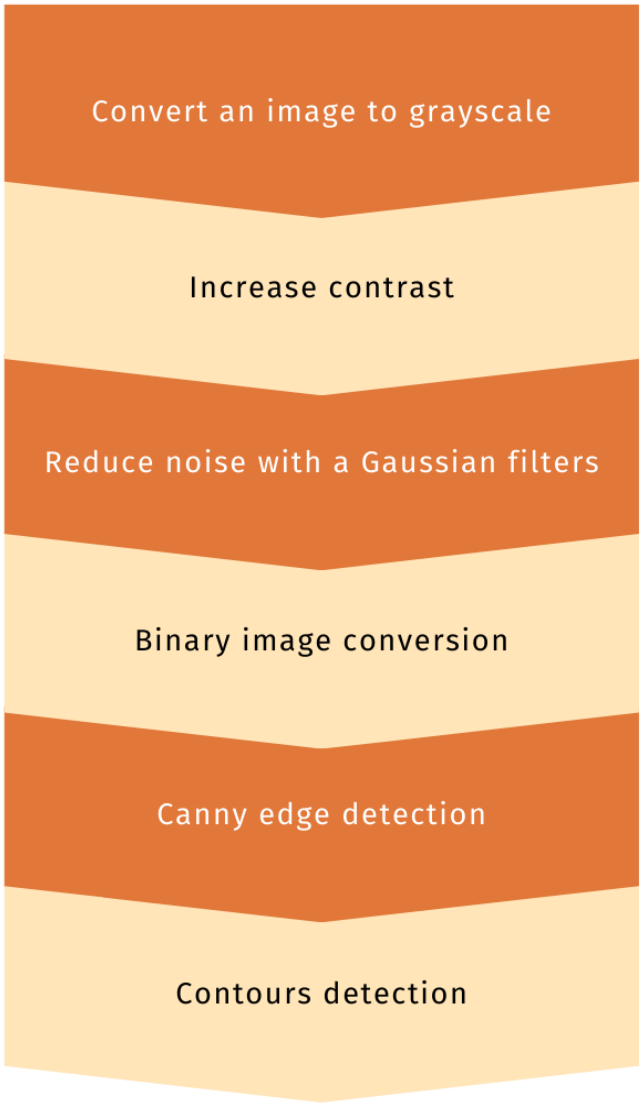


Figure III.2.5.10: Process

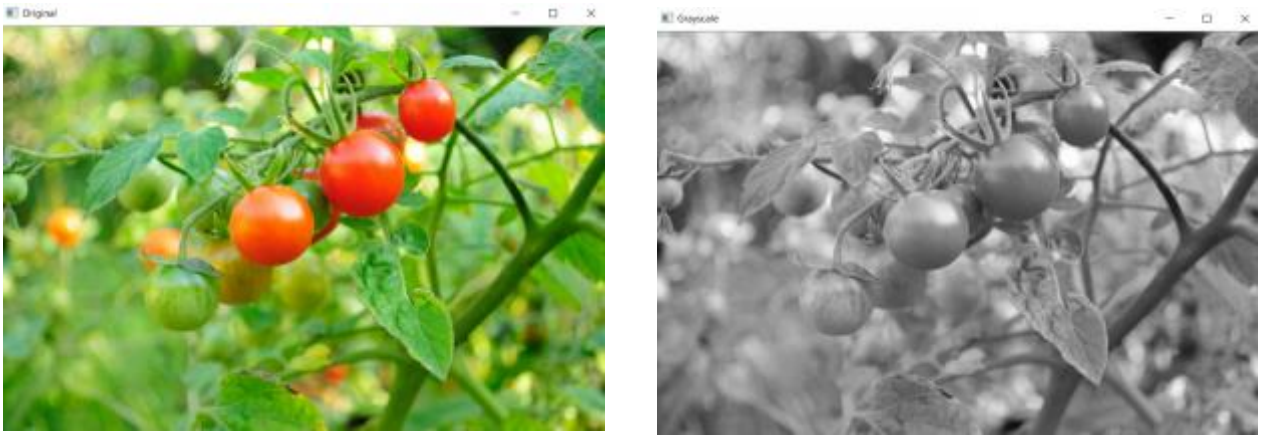
a) Grayscaling

Grayscaling is the process of converting images from color spaces like RGB, CMYK, HSV, and other color spaces into grayscale images. This process converts from a complex color representation to a simple representation that only includes shades of gray from black to white. Differentiating grayscale and other color representations is necessary because:

Reduces size: In RGB images, there are three color channels and dimensions, while grayscale images only have one dimension. This helps reduce the size of image data.

Reduces model complexity: A 10x10x3 pixel RGB image will have 300 inputs to the input layer. In contrast, a grayscale image of the same size only needs 100 inputs. This helps reduce model complexity and increase processing speed.

Compatibility with other algorithms: Many algorithms are tuned to only work on grayscale images. For example, the Canny edge detection function in the OpenCV library typically only works on grayscale images. Converting images to this format also helps make these algorithms work more effectively.



a b

Figure III.2.5.11 a, b: Color and Black & White Image of tomatoes [1]

b) Increase contrast

Morphological image processing is a set of nonlinear operations that affect the shape or morphology of binary points in an image. These operations are based on AND, OR, XOR and NOT to transform the binary points.

Morphological operators are applied to both binary and grayscale images. They are useful for changing the size of objects in images, from enlarging to shrinking. Additionally, they can also be used to reduce the distance between objects or to expand them.

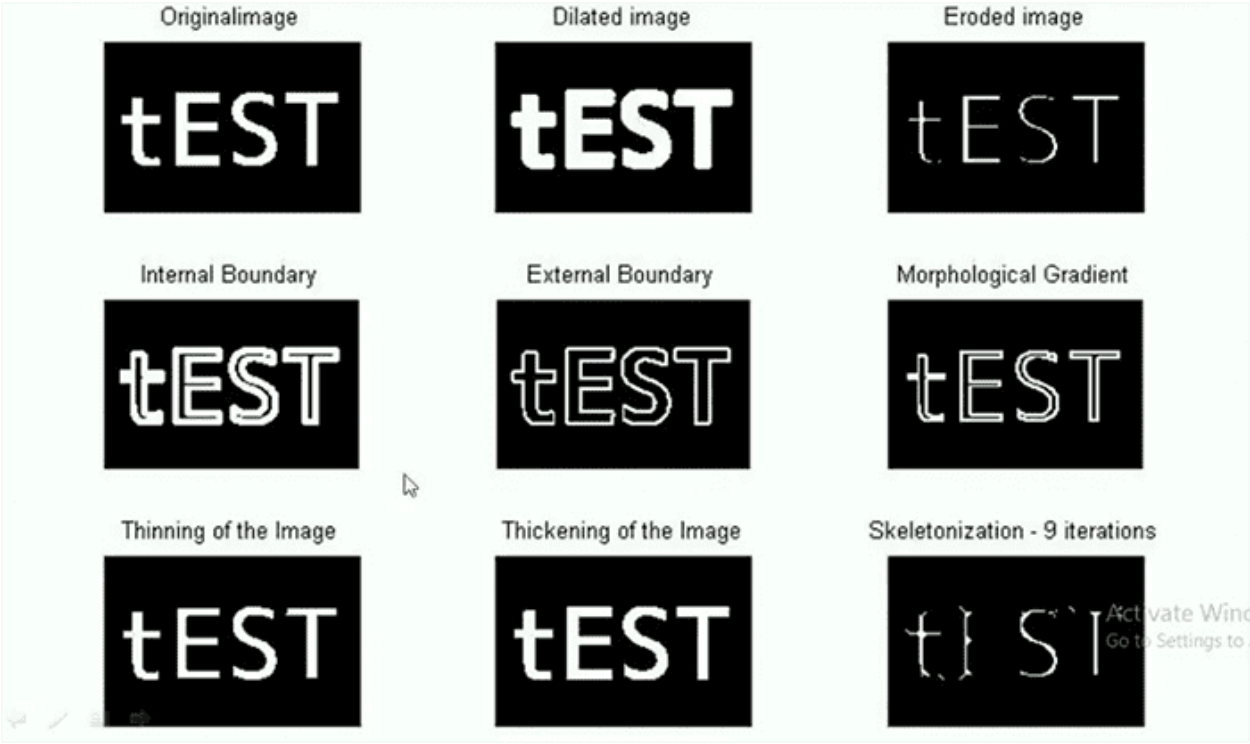


Figure III.5.12: Increase Contrast Example Image [2]

- Erosion

Erosion is the operation of shrinking morphological structures in the image. This is done by moving a morphological structure (kernel) over the image, and only retaining points in the image that lie within the kernel. This work reduces the size of objects, separates nearby objects, thins and finds the skeleton of objects.

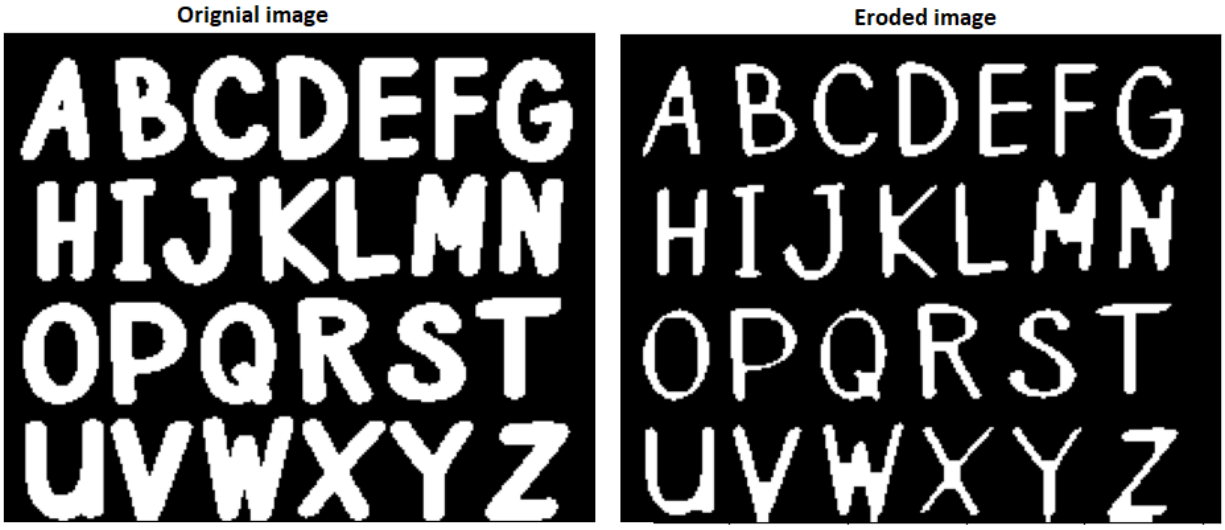


Figure III.2.5.13: Erosion Example Image [3]

- Dilation

Dilation is a morphological transform applied to structures in the image. This operation is performed by moving a morphological structure (also called a kernel) over the entire image and marking all points where the kernel passes through. The main purpose of dilation is to increase the size of objects in the image, fill small holes and connect discrete edge segments in the image to help create more continuous edges.

Figure III.2.5.14: Dilation Example Image [3]

- Closing

Closing is a combination of erosion and dilation operations. This process starts with performing erosion first, followed by dilation. The purpose of closing is to connect discrete morphological structures back together, smoothing object edges, filling gaps, and removing small unwanted holes.

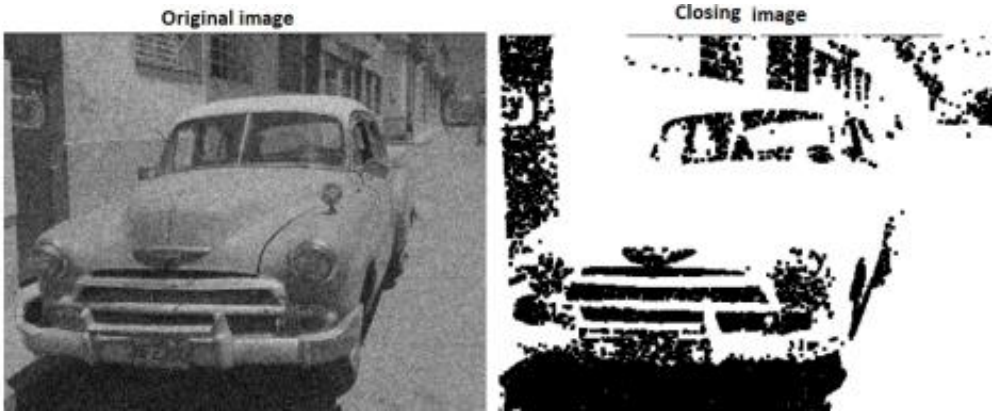


Figure III.2.5.15: Closing Example Image [3]

- Opening

Opening is an image transform combining dilation and erosion operations. This process involves first performing dilation, then erosion. This method is used to remove small noise from images, while also improving edge processing of objects to make them smoother and more continuous.

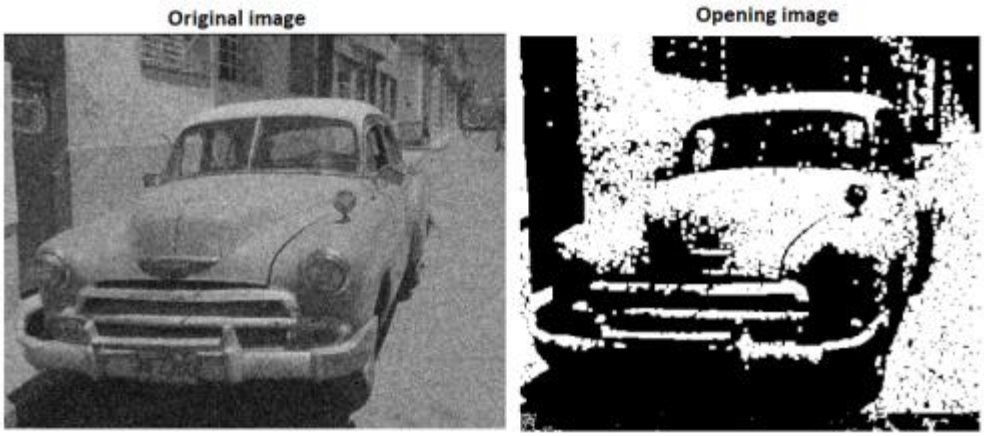


Figure III.2.5.16: Opening Example Image [3]

- TopHat

The TopHat operation is the result of subtracting the original image from the image obtained after applying erosion. The result of this operation is generated by subtracting the original image with the image obtained after opening. The main function of TopHat is to highlight white details in the dark background of the image. This operation is often used to search and clarify morphological structures smaller than a specific morphological structure.

Figure III.2.5.17: TopHat Example Image [2]

- BlackHat

The BlackHat operation is the process of comparing differences between the original image and the image after dilation. It is done by taking the subtraction result of the dilated image from the original initial image. This operation emphasizes dark details against a white background. The main function of BlackHat is to detect and highlight morphological structures larger than a specific morphological structure.

Figure III.2.5.18: BlackHat Example Image [2]

To increase license plate contrast, the main method is to use the Top Hat and Black Hat operators. The key idea is that the processed image is generated by combining the original image with the results from the Top Hat operation and then subtracting the results from the Black Hat operation. Bright details become brighter while dark details become darker, creating higher contrast for the license plate.

c) Reduce noise with Gaussian filter

Image noise refers to random variations in image brightness or color information that is the primary cause of degraded image quality during digital image transmission. The basic characteristics of noise often correspond to high frequencies, and filter theory focuses on only allowing signals within specific frequencies to pass. Therefore, using low-pass or median filters is commonly preferred for noise processing.

The Gaussian filter is very useful for processing noise and is a weighted average filter used in image processing to smooth images. The general idea when using a Gaussian filter is to use a Gaussian function to calculate weights for pixels in the image. The Gaussian function is a bell-shaped probability distribution function. The weights of the pixels are calculated based on the distance of that pixel from the center of the Gaussian function.

Pixels that have a shorter distance from the Gaussian function's center will have higher weights than pixels further away from the center. This means pixels with values closer to the average value of the image will have a greater influence on the pixel value after filtering.

The Gaussian filter can be used for various different purposes in image processing, including:

- Image smoothing: The Gaussian filter can be used to remove noise and blur images.
- Noise removal: The Gaussian filter can be used to remove grain noise, salt and pepper noise, and Gaussian noise.
- Edge detection: The Gaussian filter can be used to highlight edges in images.
- Background blurring: The Gaussian filter can be used to blur backgrounds in images.

Figure III.2.5.19: Gauss Filter Example Image

d) Binary image conversion [4]

A binary image contains only two colors, black (with value 0) and white (with value 255). The goal is to convert a grayscale image (or a 2D matrix with the value of each cell between 0-255) into a new 2D matrix, where the value of each cell is 0 or 255.

To solve this problem, we choose a threshold to distinguish between black and white pixels. If the pixel value on the grayscale image is greater than the threshold, we assign the corresponding pixel on the black and white image a value of 255 (white); otherwise, if the value is less than or equal to the threshold, we assign the pixel a value of 0 (black).

When calling the grayscale image Gray and the black and white image to create BW, with the coordinates of each pixel on the image as (x, y), we apply:

BW (x, y) = 255 if Gray (x, y) > threshold
BW (x, y) = 0 if Gray (x, y) <= threshold

Depending on the threshold used, we will have different resulting images.

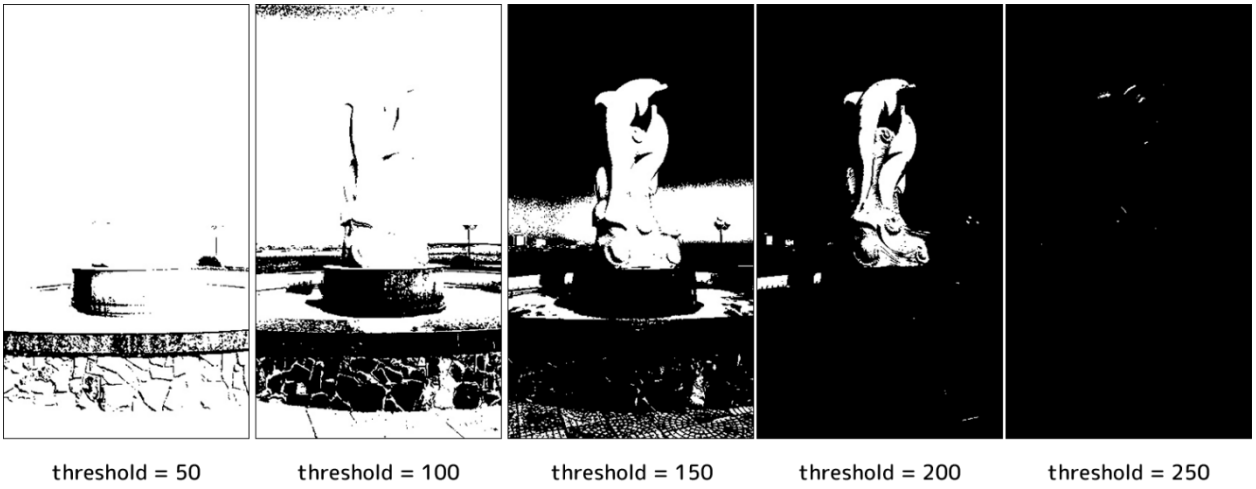


Figure III.2.5.20: Image shows the result of using different thresholds

And we have some types of binary image transformations as follows:

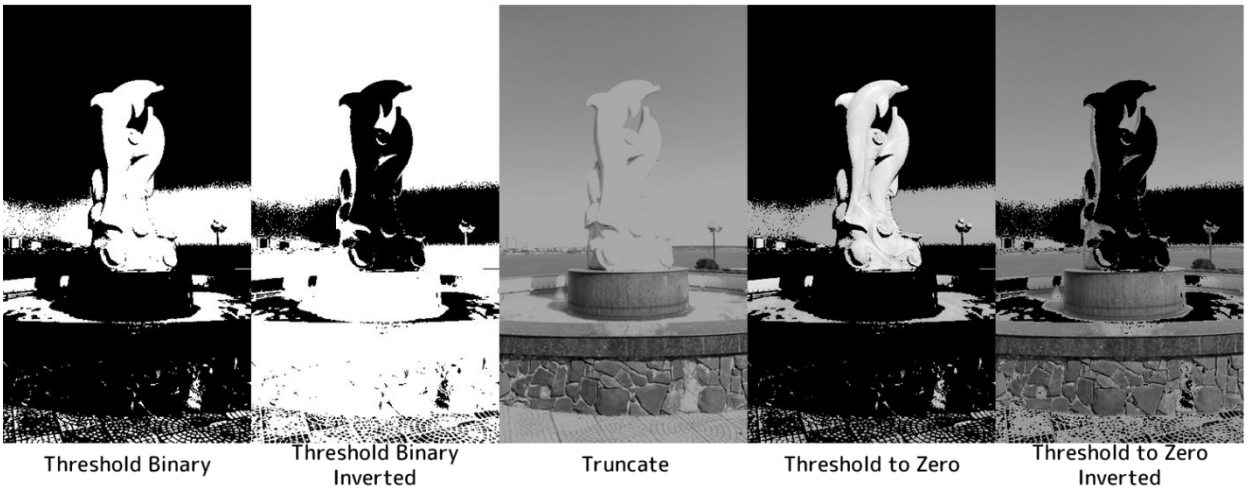


Figure III.2.5.21: Image shows some modes of binary image transformation

Threshold Binary: Converts the image to black and white based on a predefined threshold.

Threshold Binary Inverted: Creates a black and white image inverted to the Threshold Binary technique, meaning the colors will be reversed.

Truncate: If the pixel value exceeds the threshold, the value will be kept at the set threshold level (limits the highest brightness).

Threshold to Zero: Pixels with values lower than the threshold will be turned black.

Threshold to Zero Inverted: Pixels with values greater than the threshold will be turned white.

However, the results depend heavily on the threshold used. In practice, for real-time image processing applications, applying a fixed threshold is not feasible due to continuous changes in light intensity. To solve this problem, we can use the adaptive thresholding method.

Basically, this method divides the image into small regions and then calculates the corresponding threshold for each region based on the intensity of the pixels in that region. The specific approach to calculating thresholds may vary depending on the algorithm. A common method is to use the average or median intensity value of the area. Another method is to use a weighted average of the intensity values, where pixels near the edge of the region are considered to have higher weights.

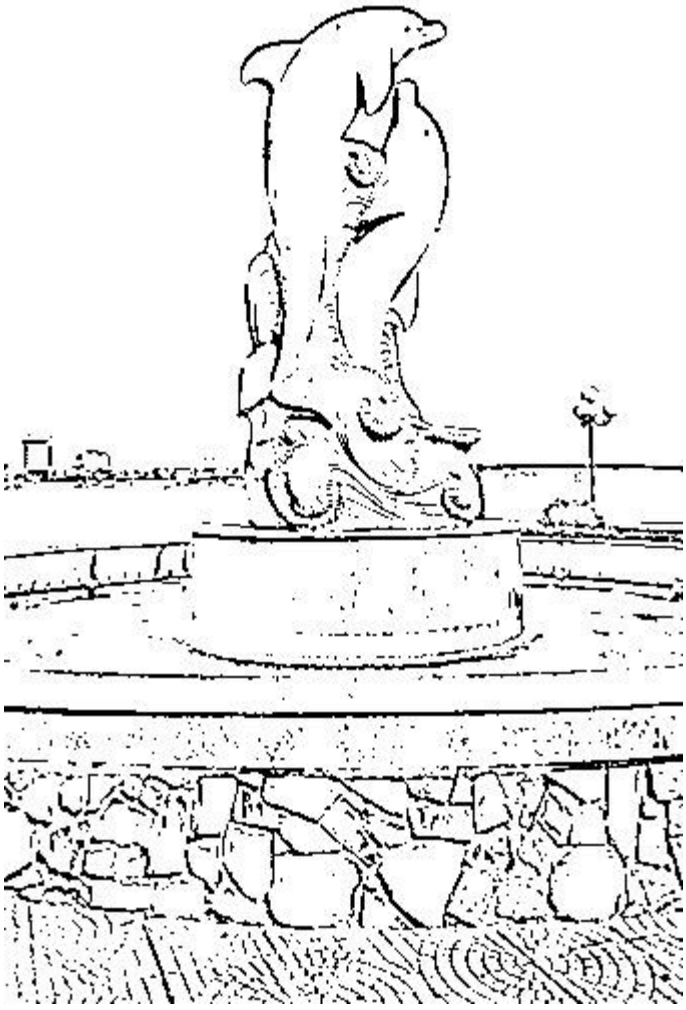


Figure III.2.5.22: Image shows dynamic image thresholding

e) Edge detection [5]

Detecting edges in an image identifies discontinuities in pixel brightness or simply abrupt changes and shifts in pixel values. Edge features often reflect the rapid variation of objects in the image.

The Canny edge detection algorithm, developed by John F. Canny in 1986, is an effective and robust edge detection method that can be widely applied in image processing.

Canny edge detection works in four main steps:

Gaussian smoothing: The first step is to smooth the image to reduce noise and blur edges. This is often done by applying a Gaussian filter, such as a 5x5 Gauss filter, to reduce noise.

Gradient magnitude and orientation: The next step is to calculate the image gradient, describing the intensity and direction of rapid pixel change in the defined area.

*Non-maxima suppressio*n:

Non-maxima suppression may seem like a confusing process, but in reality, it's not that complicated - it simply is a step to thin edges. To do this, we use a 3x3 filter to process each pixel in the gradient image. In this process, we check if the magnitude of the gradient at the center pixel is the largest compared to the surrounding pixel gradients. If it is a maximum point, we keep that pixel. If not, meaning the pixel is not a maximum point, we set the magnitude of the gradient to zero. We only compare the center pixel to the two adjacent pixels along the gradient direction to determine if it is a maximum point.

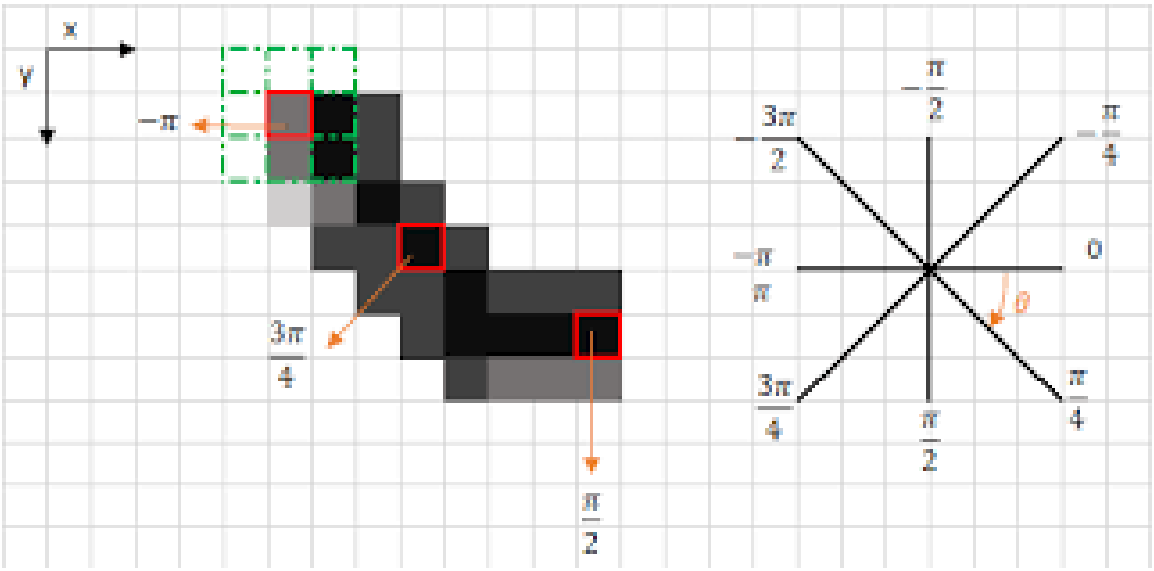


Figure III.2.5.23: Image shows Non-maxima suppression process [6]

Hysteresis thresholding: The final step is to track and identify edges in the image through connectivity algorithms to help determine contour lines.

Even after applying Non-maxima suppression, we may still need to remove areas of the image that are not technically edges, but still respond as edges after calculating gradient magnitude and Non-maxima suppression.

To ignore these areas of the image, we need to determine two thresholds: T_{upper} and T_{lower}

Any gradient value $G > T_{upper}$ that is definitely an edge.

Any gradient value $G < T_{lower}$ that definitely is not an edge, so immediately remove those regions.

And any gradient values that fall within the range of $T_{lower} < G < T_{upper}$ need to undergo further testing:

- If the specific gradient value is connected to a strong edge (e.g., $G > T_{upper}$), mark the pixel as an edge.
- If the gradient pixel is not connected to a strong edge, remove it.

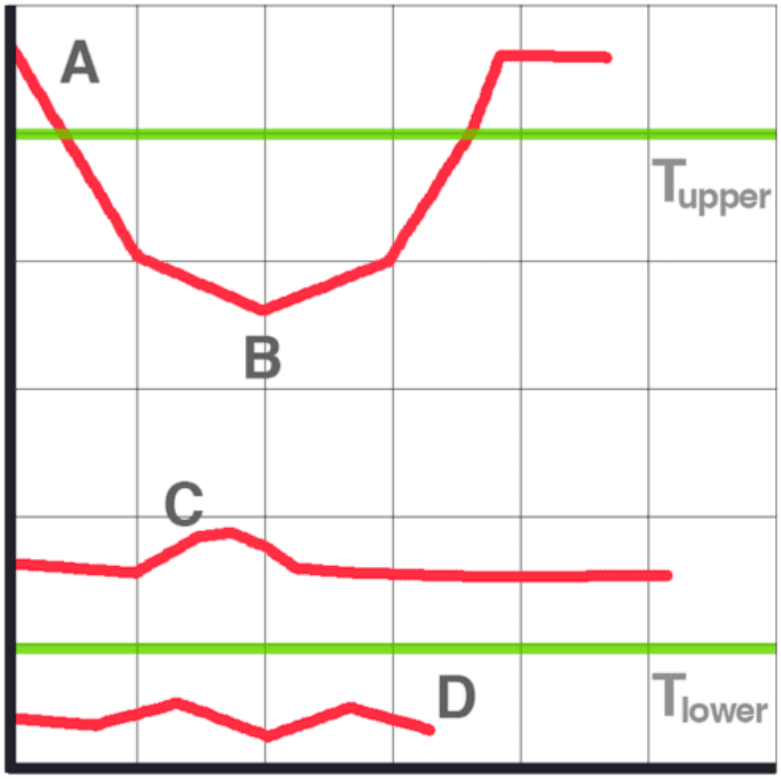


Figure III.2.5.24: Image show threshold filtering

With different T_{lower} and T_{upper} values, we also get corresponding results:

```
wide = cv2.Canny(blurred, 10, 200)
mid = cv2.Canny(blurred, 30, 150)
tight = cv2.Canny(blurred, 240, 250)

# show the output Canny edge maps

cv2.imshow("Wide Edge Map", wide)
cv2.imshow("Mid Edge Map", mid)
cv2.imshow("Tight Edge Map", tight)
cv2.waitKey(0)
```

Fig 2.5.3.15 Image shows the Results with different threshold filters

f) Contour detection [7]

Contours can simply be explained as a curve joining all the continuous points (along the boundary) that have the same color or intensity. Contours are a useful tool for analyzing shape and detecting and identifying objects.

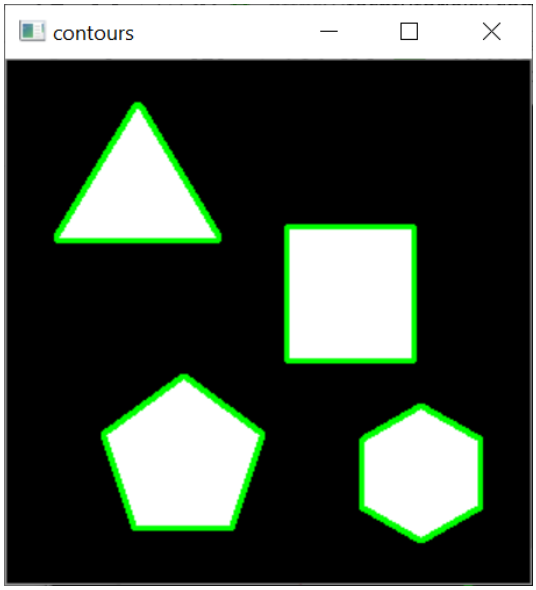


Figure III.2.5.26 Image show Examples of contours [8]

- For better accuracy, use a binary image. So before finding contours, apply thresholding or Canny edge detection.
- In OpenCV, finding contours is like finding white objects from black backgrounds. So, remember that the object to be found must be white and the background must be black.

2.5.3.3 Localizing and Segmenting the License Plate

The findContours() function in OpenCV is used to find contours in an image. Contours are curves that represent the outline of an object in an image. They are useful for object detection, shape analysis, and image segmentation.

findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())

The findContours() function takes the following parameters:

image: The input image. It must be a grayscale or binary image.
contours: The output vector of contours. Each contour is represented as a vector of points.
hierarchy: The output vector of hierarchical relationships between contours.
mode: The contour retrieval mode. It can be RETR _EXTERNAL, RETR _LIST, RETR_ CCOMP, or RETR_TREE.

- RETR_EXTERNAL: This mode retrieves only the external contours. External contours are the top-level contours that are not inside any other contours.
- RETR_LIST: This mode retrieves all contours, including both external and child contours. Child contours are contours that are inside other contours.
- RETR_CCOMP: This mode retrieves all contours and reconstructs a two-level hierarchy of contours. In this hierarchy, parent contours are contours that contain other contours, and child contours are the contours that are contained within parent contours.
- RETR_TREE: This mode retrieves all contours and reconstructs a full hierarchy of contours. In this hierarchy, there can be more than two levels of contours.

method: The contour approximation method. It can be CHAIN_APPROX_SIMPLE, CV_CHAIN_APPROX_NON or CHAIN_APPROX_TC89.

- CHAIN_APPROX_SIMPLE: This method approximates the contour using a polygonal curve with the minimum number of vertices. This is the most commonly used method, as it is the simplest and fastest.
- CV_CHAIN_APPROX_NON: This method does not approximate the contour. It returns the original contours as they were extracted from the image.
- CHAIN_APPROX_TC89: Apply approximate algorithm Tech-Chin.

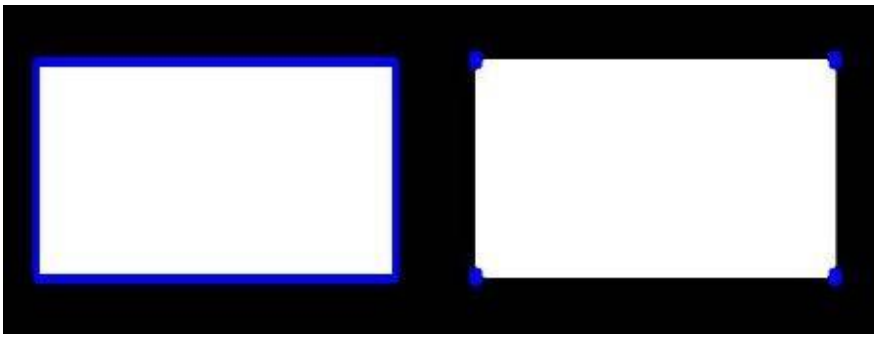


Figure III.2.5.27: Image show using method CV_CHAIN_APPROX_NON and method CHAIN_APPROX_SIMPLE [7]

offset: The optional offset for all the points in the contours.

License Plate Filtering

Figure III.2.5.28: Image show drawing contour with OpenCV

After applying the above steps, we get an image with blue contour lines surrounding the objects. The next step is to approximate those contours into polygons. In other words, when approximating the contours, the computer only remembers the vertex points of that polygon in an array. The number of sides of the polygon will be equivalent to the number of vertex points and equal to the length of this array.

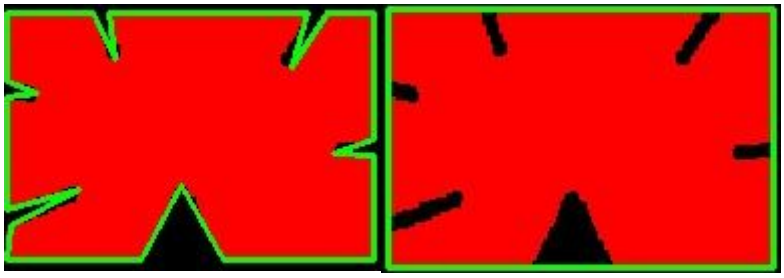


Figure III.2.5.29: Image show Unapproximated contour and Approximated polygon contour [9]

After polygon approximation, the result is:

Figure III.2.5.30: Image show after drawing the approximated polygons

We see that in addition to the rectangle surrounding the license plate, there are some other rectangles. So here we will calculate the aspect ratio and area of the rectangle that fits the criteria, then save the license plate in the image as vertex coordinates. From here we can cut out the license plate image from the known location coordinates to serve the next purpose of "Separating characters on license plates".

2.5.3.4 Character Segmentation on License Plate

Figure III.2.5.31: Image of obtained license plate

After cutting out the image containing the license plate, we will invert the image to black and white to change the black text to white. Next, we will find contours for the white characters and draw rectangles around them. However, at the same time, we can also notice the appearance of noise that is not characters. To eliminate this noise, we will use criteria regarding character height, width and area compared to the license plate area to identify and eliminate them.

Figure III.2.5.32: Image of license plate contours

The result obtained:

Figure III.2.5.33: Image of Polygon approximation of characters

The next step is to cut each character from the binary image for character recognition

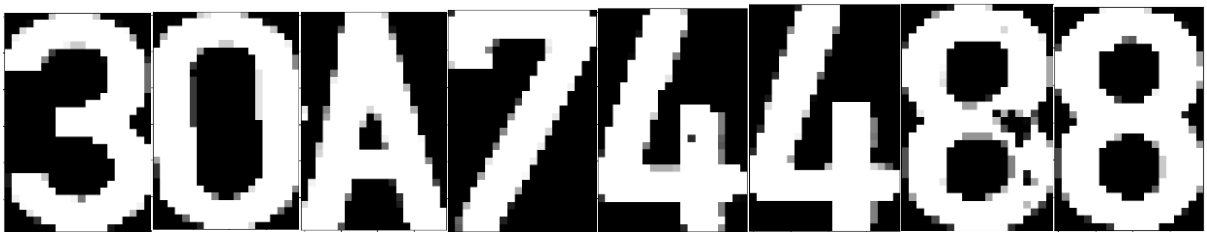


Figure III.2.5.34: Image show result when cutting out the characters

2.5.3.5 Character Recognition

Character recognition (OCR) is a computer technology that allows computers to recognize characters in handwritten or printed text. OCR can be used for various applications such as document digitization, workflow automation, and information access for the disabled.

K-nearest neighbors is a simple, easy to implement supervised machine learning algorithm. It is often used in classification and regression problems. KNN works by finding the nearest data points to the new data point and uses the data from those data points to predict the value of the new data point. For example, you want to predict a student's grade type in an exam, but you don't have an accurate way to classify the grade. You want to use other students' grade data to estimate that student's grade type.

Given a student's grade is 6.8. You have collected data from the 5 students with grades closest to 7 and their grade types as follows:

Student A: Grade 7.1 => Type: Good
Student B: Grade 7.2 => Type: Good
Student C: Grade 6.7 => Type: Good
Student D: Grade 6.6 => Type: Good
Student E: Grade 6.4 => Type: Average

Based on this information, you want to estimate the grade type of a student with grade 6.8. And you guess that the grade type may be "Good" based on the results collected from the sample group. And you may notice that the more and wider data you investigate, the more accurate the prediction (Assuming no one else in your class has a "Good" grade except you, no matter how many people closest to your grade you take, the result will still be wrong) [10].

The KNN algorithm workflow can be divided into the following steps:

1. Training: In this step, the KNN algorithm is trained on a labeled dataset. This dataset includes data points that have been labeled with corresponding classes.

2. Find nearest data points: In this step, the KNN algorithm finds the nearest data points to the new data point. The distance between two data points is usually calculated using a distance function, such as the Euclidean distance function.

3. Prediction: In this step, the KNN algorithm uses data from the nearest data points to predict the value of the new data point. The predicted value is usually the label of the class that the new data point is most likely to belong to.

How to determine K value

The K value is an important parameter of the KNN algorithm. The K value determines the number of nearest data points that will be used to predict the value of the new data point. The larger the K value, the smaller the influence of the nearest data points.

There are two common ways to determine the K value:
- Manually: This method requires the user to experiment with different K values and choose the value for the best result.
- Automatically: This method uses an algorithm to determine the optimal K value.

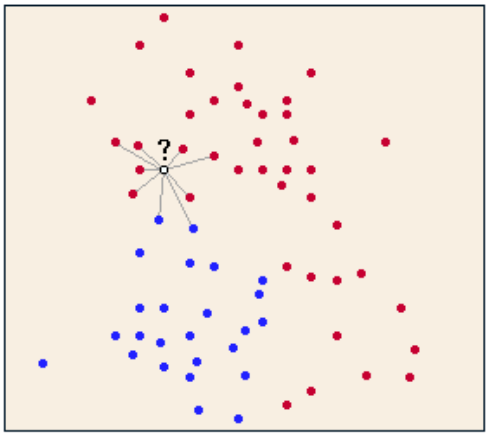


Figure III.2.5.35: Example of KNN algorithm [11]

Advantages
- Simple, easy to implement algorithm.
- Low computational complexity.
- Handles noisy data well

Disadvantages
- With small K values, noise can lead to inaccurate results
- Requires a lot of time to execute since distances to all objects in the dataset need to be computed.
- Data types need to be converted into categorical features. [12]

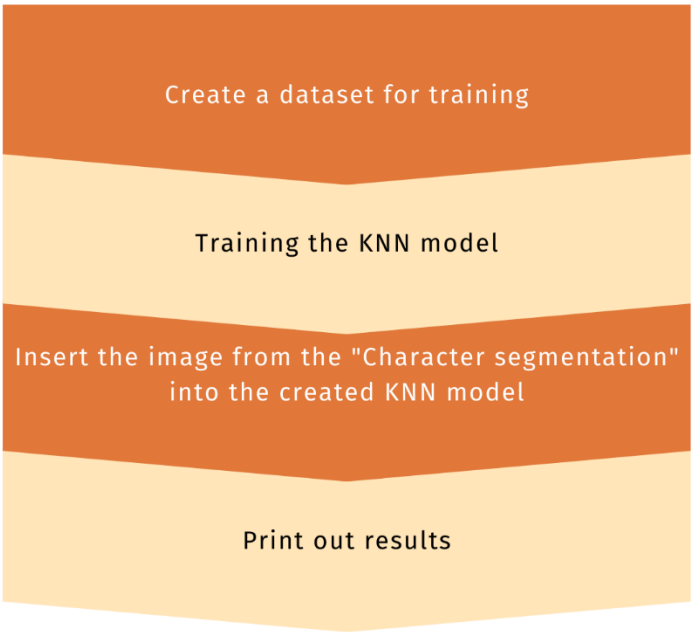


Figure III.2.5.36: Order of solution steps

Step 1: We will create a diverse dataset including numbers and common characters appearing on Vietnamese license plates, using a variety of fonts. At the same time, we will rotate the characters at different angles such as -15, -10, -5, 5, 10, and 15 degrees. This rotation is aimed at increasing data diversity, thereby improving prediction accuracy when building models and performing classifications.

Figure III.2.5.37: Image of Training dataset

Step 2:

Next, we will apply thresholding, draw contours and cut out each character from the image. Since each character has a different size and requires complex processing, image normalization becomes necessary to ensure a standard aspect ratio of 30:20 pixels for each character. These characters will be labeled through keyboard keys.

Once all characters have been labeled, we will save two .txt files called classifications.txt and flattened_images.txt. The classifications.txt file is used to store ASCII codes corresponding to each character and the flattened_images.txt file will contain pixel values of the character image. This 20x30 pixel image contains a total of 600 pixels with values of either 0 or 255.

Step 3:

We will use the segmented character images and put them into the pre-trained KNN model. Next, we will calculate distances from the samples to this new character image. The result will be converted to the ASCII code of that image.

Figure III.2.5.38: Vehicle license plate after character recognition

With two license plate types: single line and double line license plates. To determine the correct character position in the license plate, we will consider the position of the character relative to the

height of the license plate. If the character position is within 1/3 of the license plate height, we will classify it into the first line. Conversely, if the character position does not lie within this range, we will put that character into the second line.

Figure III.2.5.39 Vietnam's 2 types of car license plate

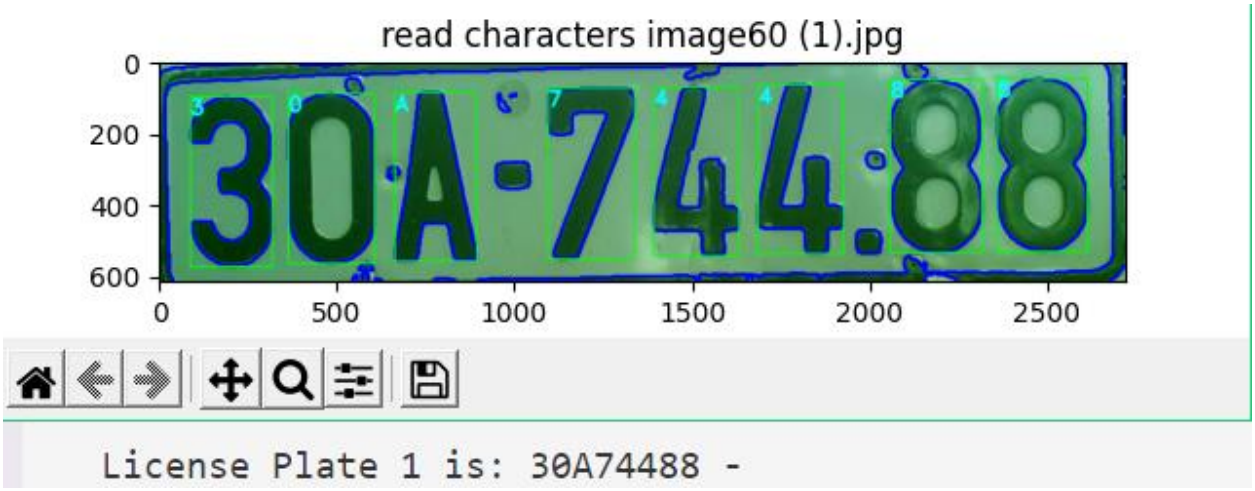


Figure III.2.5.40: Image of printed license plate

3. Software design

3.1 Welcome Screen Interface

The below image is the welcome interface of the SPS application

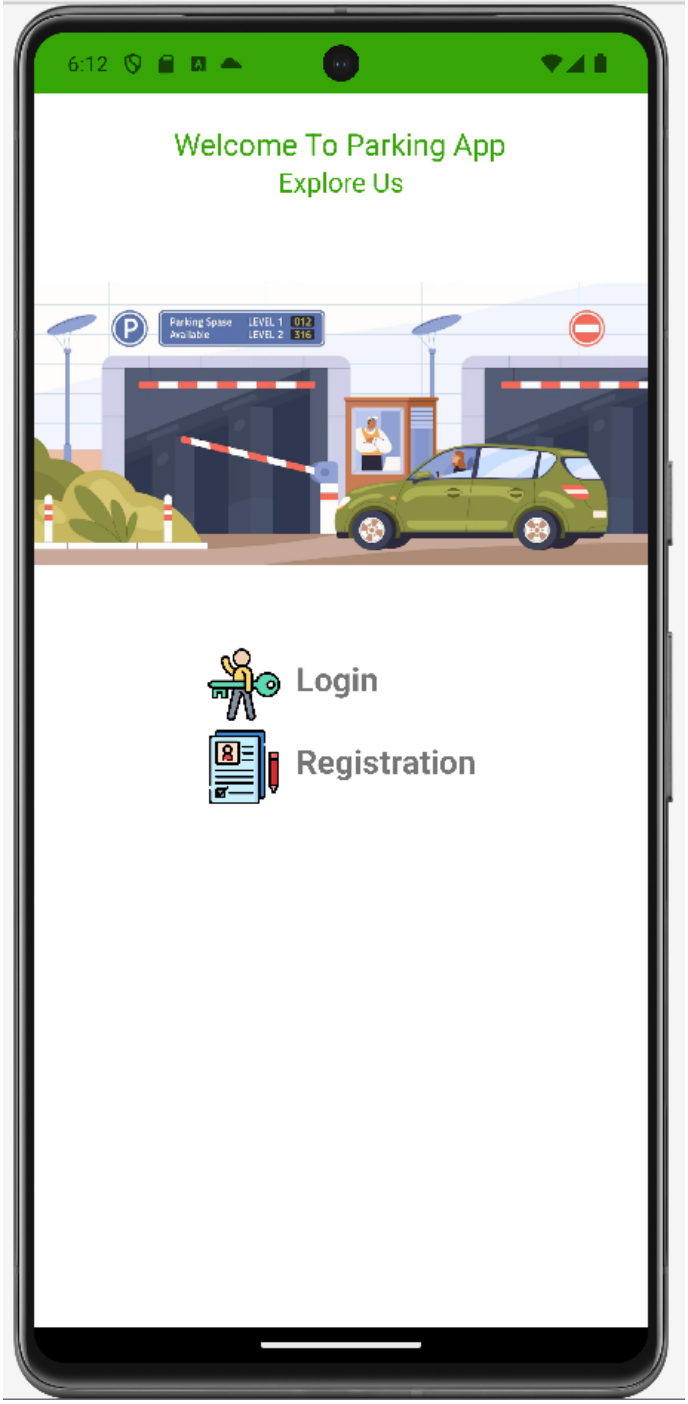


Figure III.3.1: Application Interface

The welcome screen displays "Welcome to Parking App" text with two buttons presented in a vertical linear layout which "Login", lead users to enter existing account and "Registration", for new users to create accounts.

3.2 Login Interface

The image shows the login interface, after the user tap on the “Login”

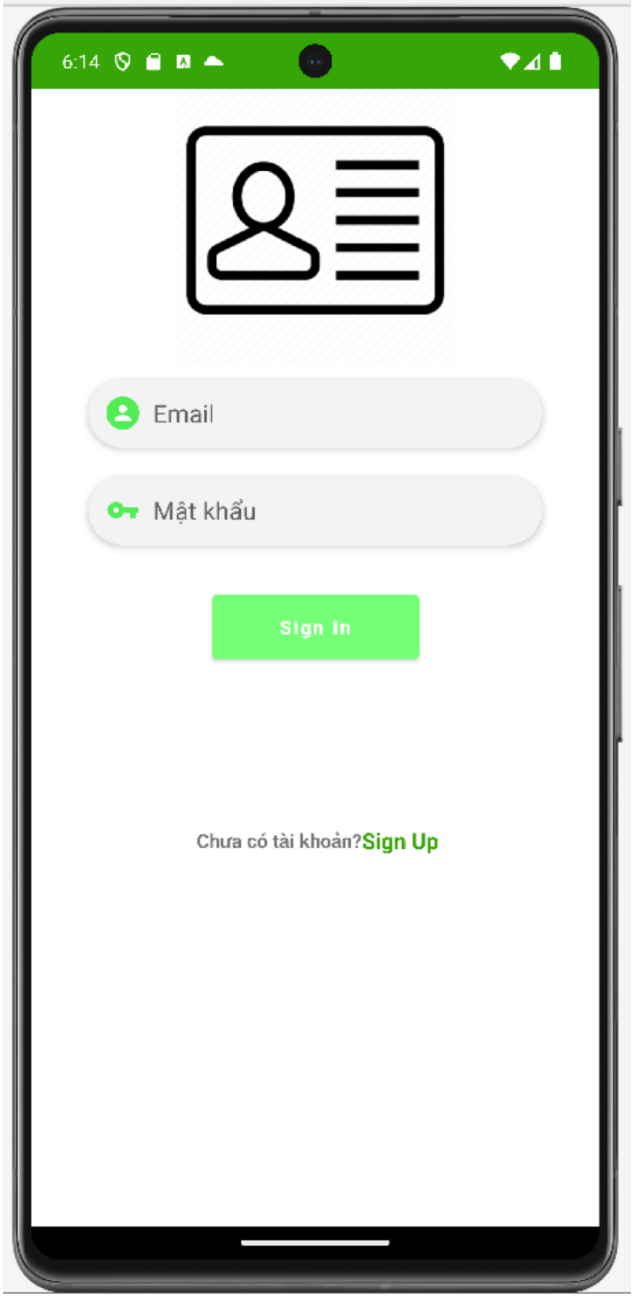


Figure III.3.2: Application Login Interface

After press Login, the screen will change to Login Interface, which allow users to enter their Email and Password to sign in to their existed account.

3.3 Sign up Interface

The image shows the sign-up interface

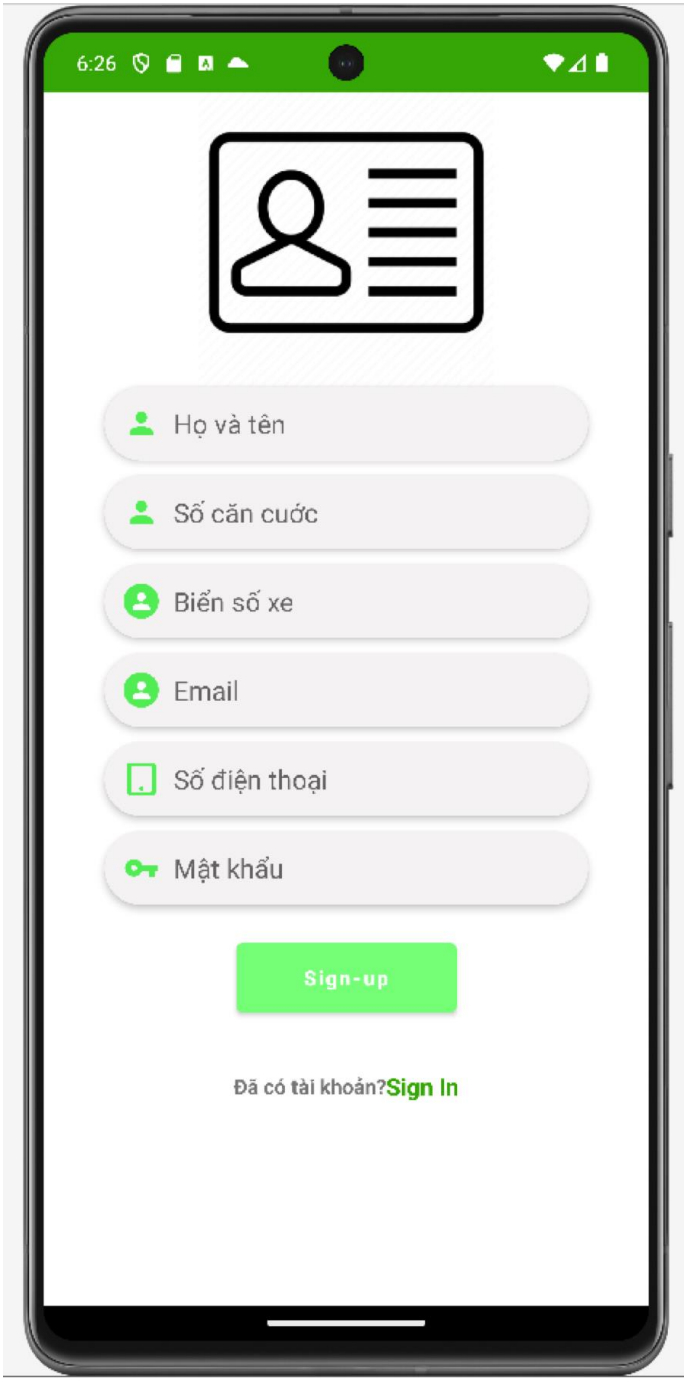


Figure III.3.3: Application Sign Up Interface

After the users choose Registration, the screen will change to sign up interface, which allow users enter their personal information including: Name, ID number, License Plate, Email, Phone Number and desired Password to creating a new account.

3.4 Home View Interface

This image shows the home screen of the application

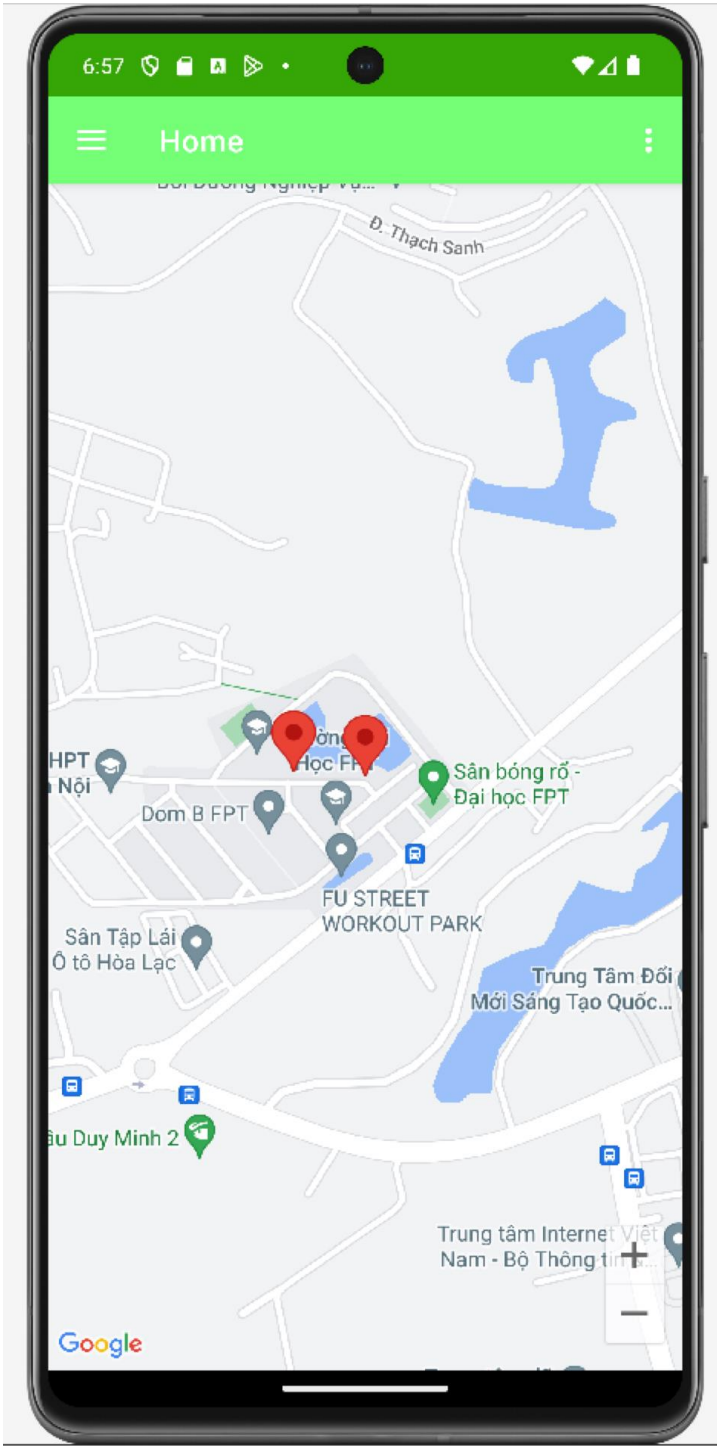


Figure III.3.4: Application Home View Interface

After the user's login successfully, the screen will change to home view, which showing the map view of parking zones. On this map view, there are two markers, indicate two zones.

The first marker indicates Alpha parking zone

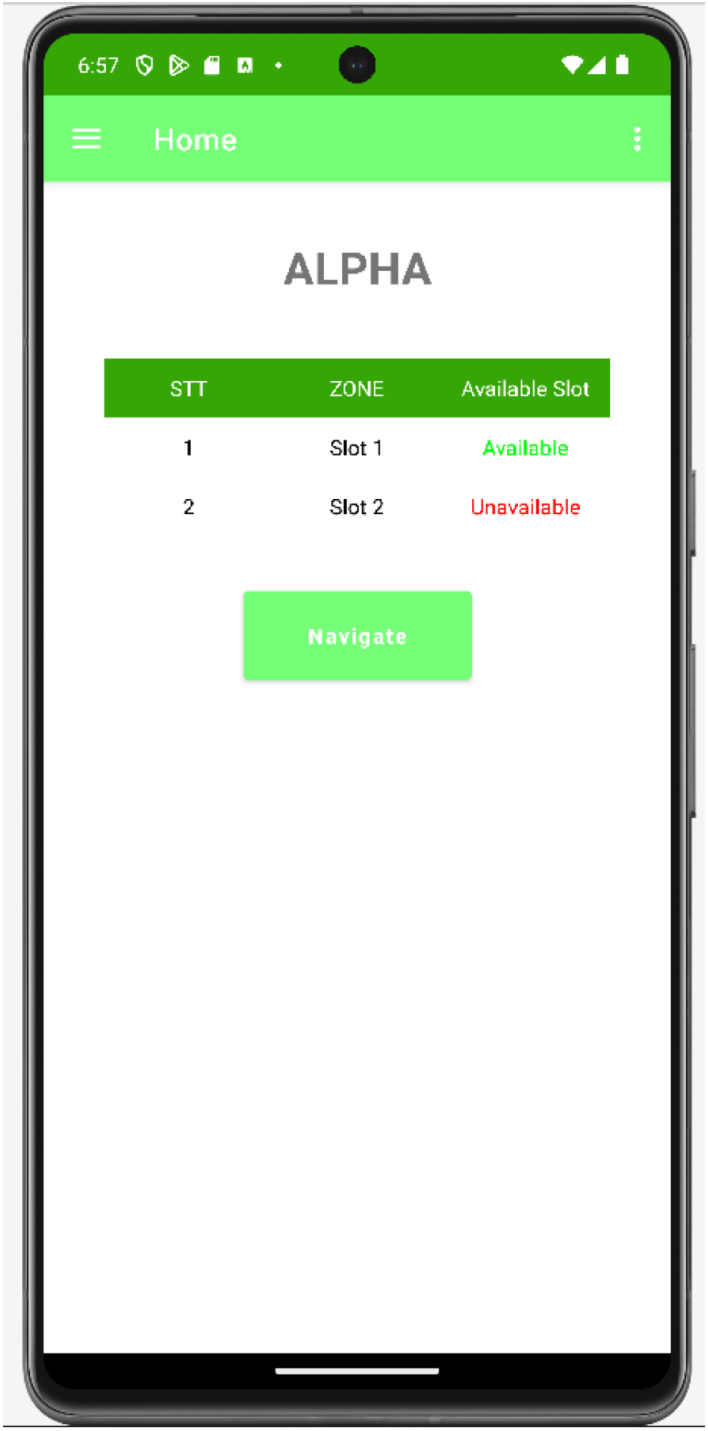


Figure III.3.5: Alpha Parking Zone Status Interface

On this status interface, it shows numbers and availability of Alpha parking zone. “Available” means a lot is vacant, “Unavailable” means a slot is occupied.

The second marker indicates Beta parking zone

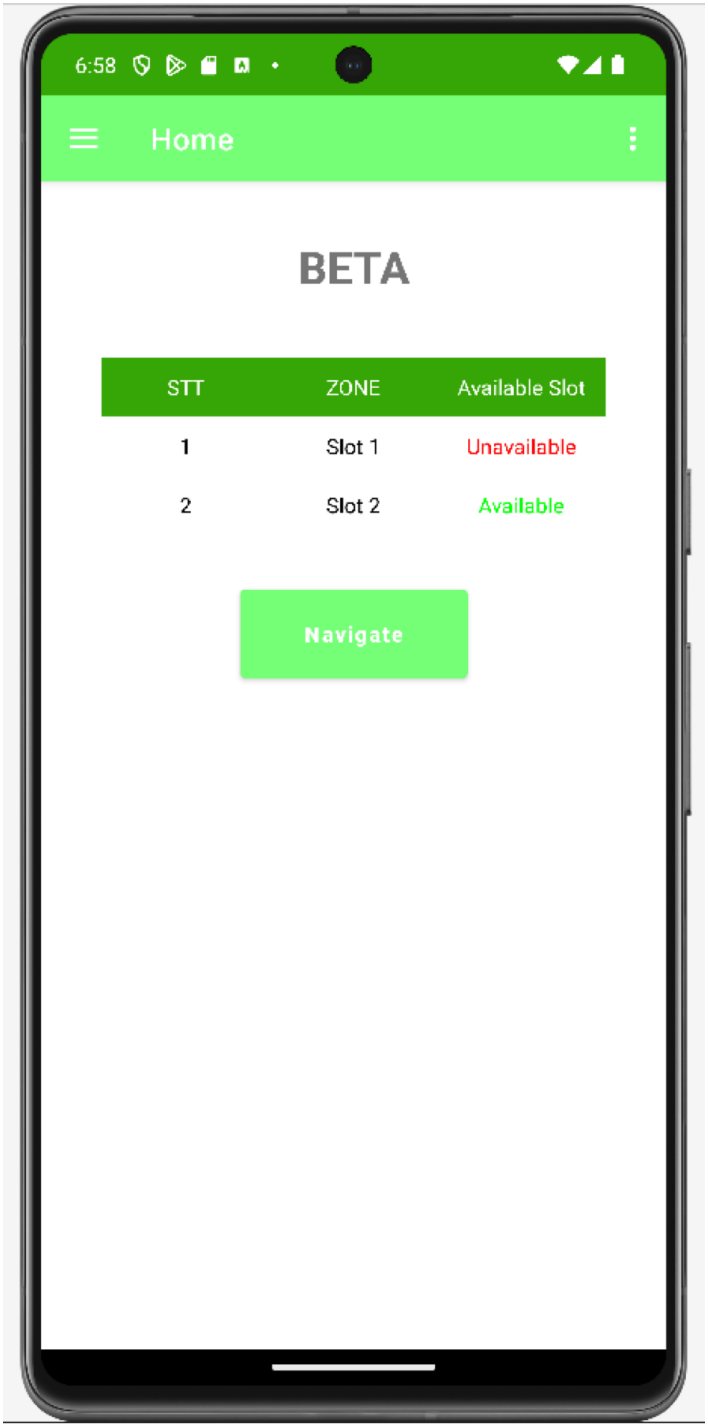


Figure III.3.6: Beta Parking Zone Status Interface

It has the same functions as the Alpha status interface.

After pressing navigate, the app will change to Google Map app, which shows the map with the direction to the address (coordinate) of Alpha or Beta

Figure III.3.7: Home View with Direction

The app uses Google Maps for powering directions and parking lot marking interfaces. If it is not installed, the screen redirects to the Google Play Store prompting the user to download the required app. After installation, user returned to the app to leverage the newly available Google Maps features. If an outdated Maps version exists instead, the flow similarly goes to the Google Play Store suggesting an app upgrade then relaunching with updated dependencies.

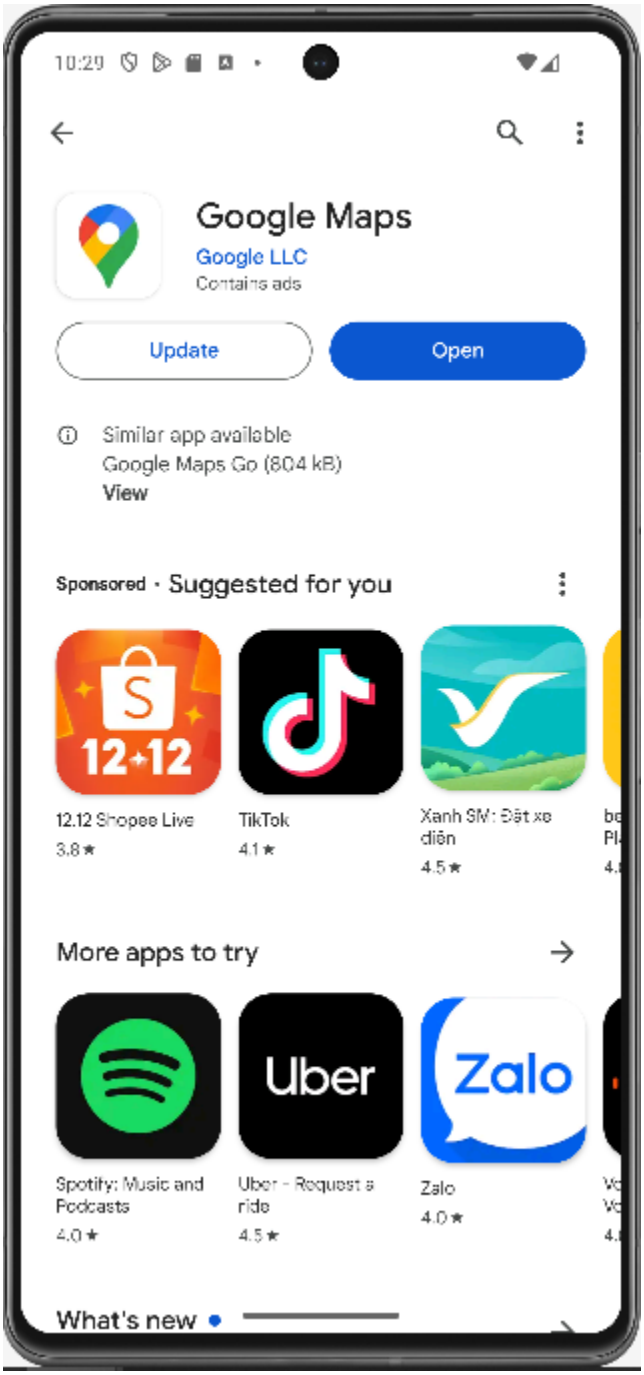


Figure III.3.8: Google Play Interface

On the home view interface, on the top of the screen, there is a menu icon, after pressing the icon, a navigation bar appears from the left.

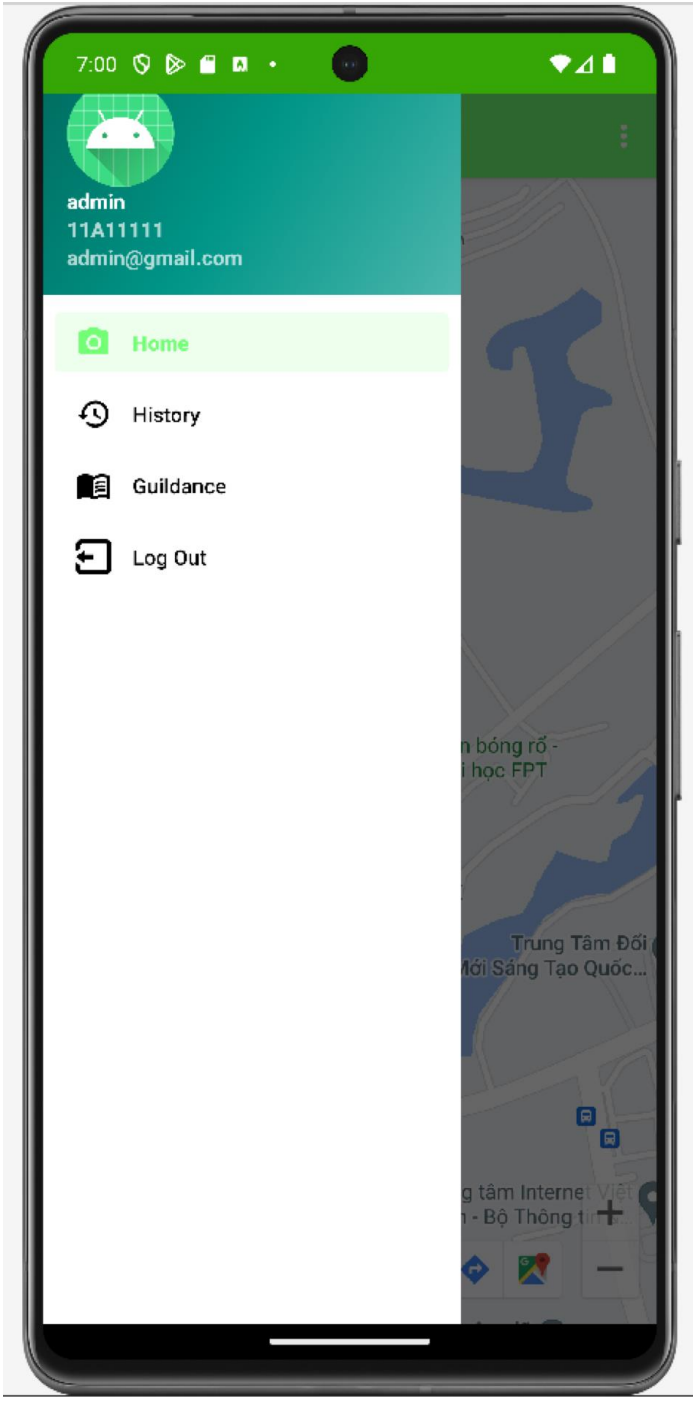


Figure III.3.9: Application Navigation Bar

The bar contains user's information and 4 tabs. Home tab indicate the home view, which is the map.

History tab indicate user's parking history which allow user to access their parking log data

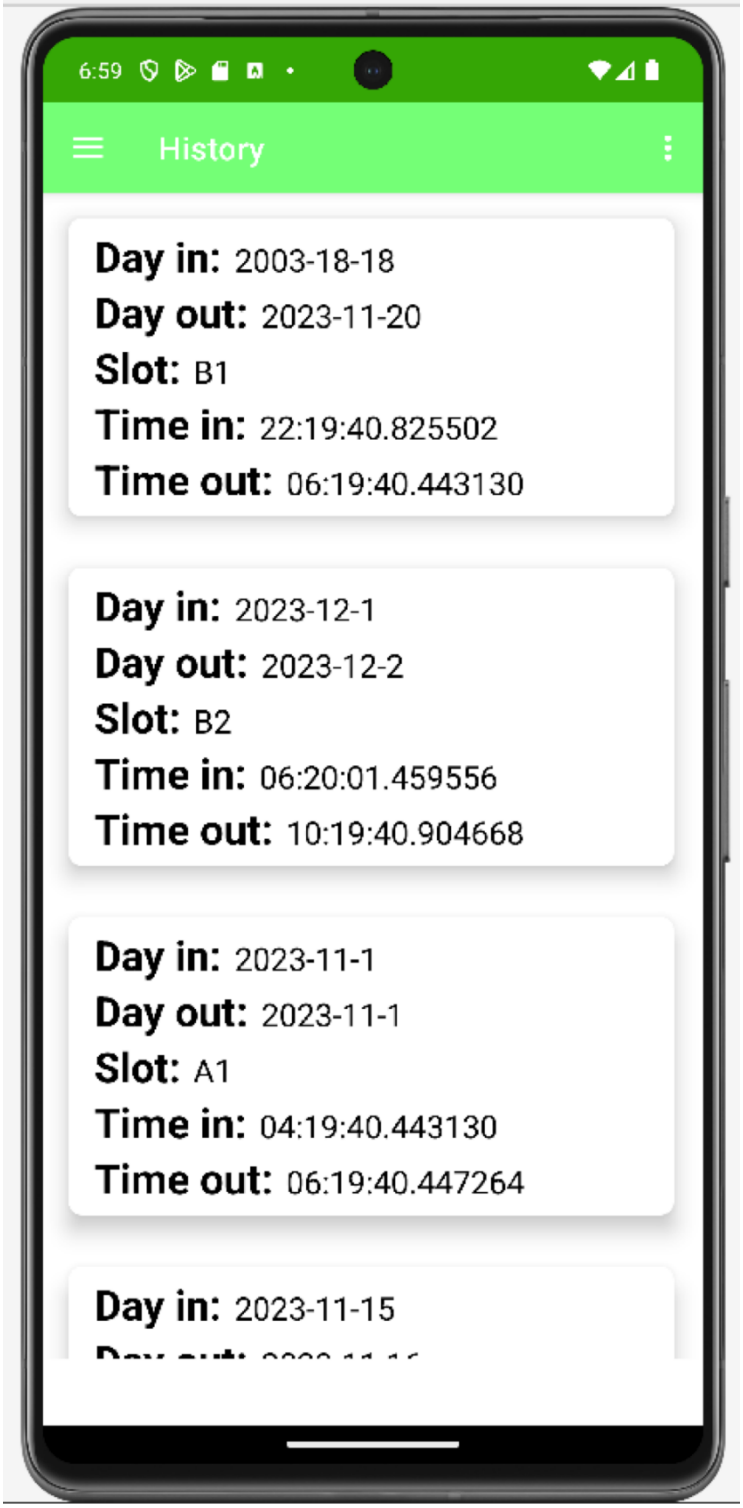
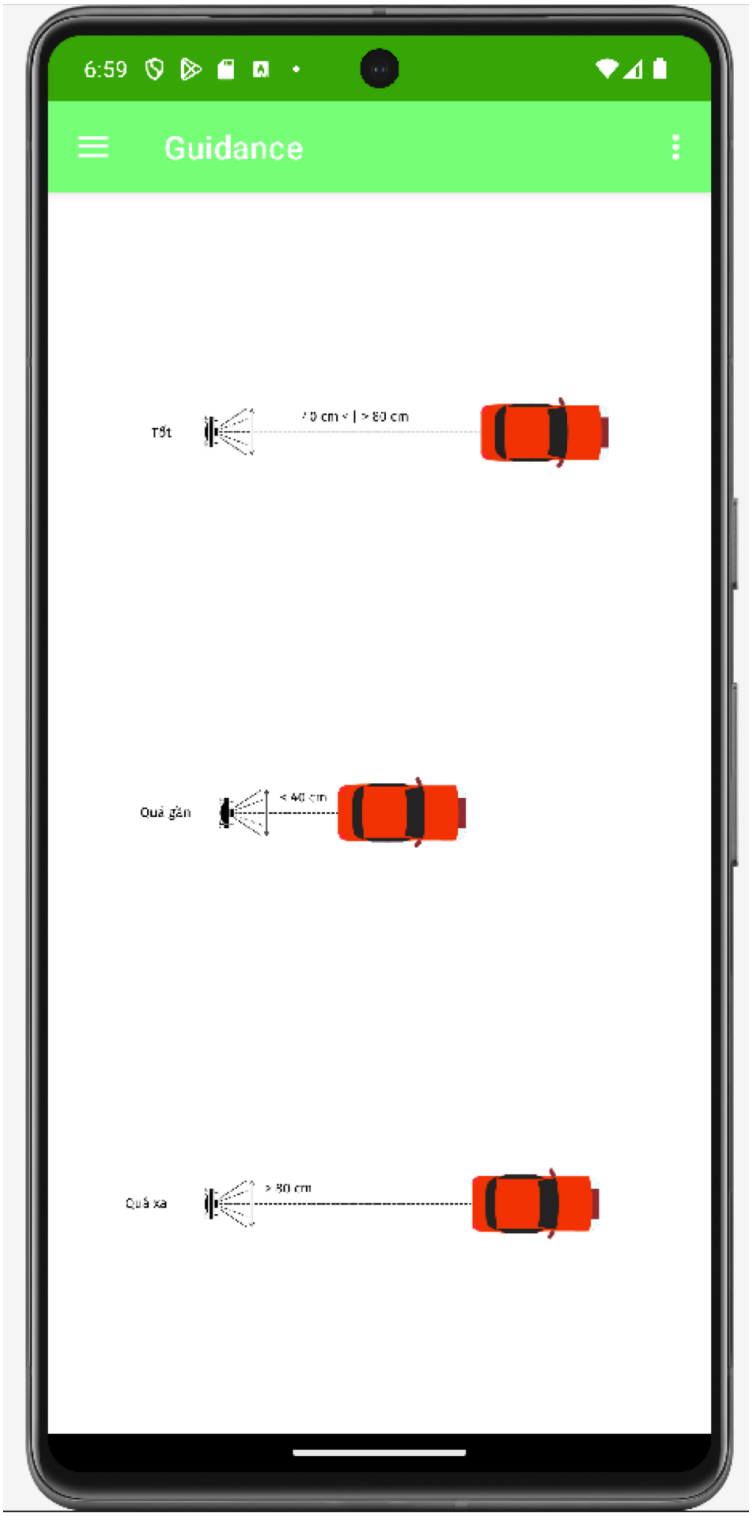


Figure III.3.10: Application History Interface

This tab enables users to review duration details on recent parking sessions.



Additionally, a guidance tab guide how to park the car properly

Figure III.3.11: Application Guidance Interface

On the guidance screen, there are 3 pictures to help the users. It is best to park the car 40cm to 80cm from the camera. If is below 40cm, it is too close for the camera to capture the image. It is too far to capture the image when car is parked 80cm away.

There is also a log out tab, to let user log out of their account

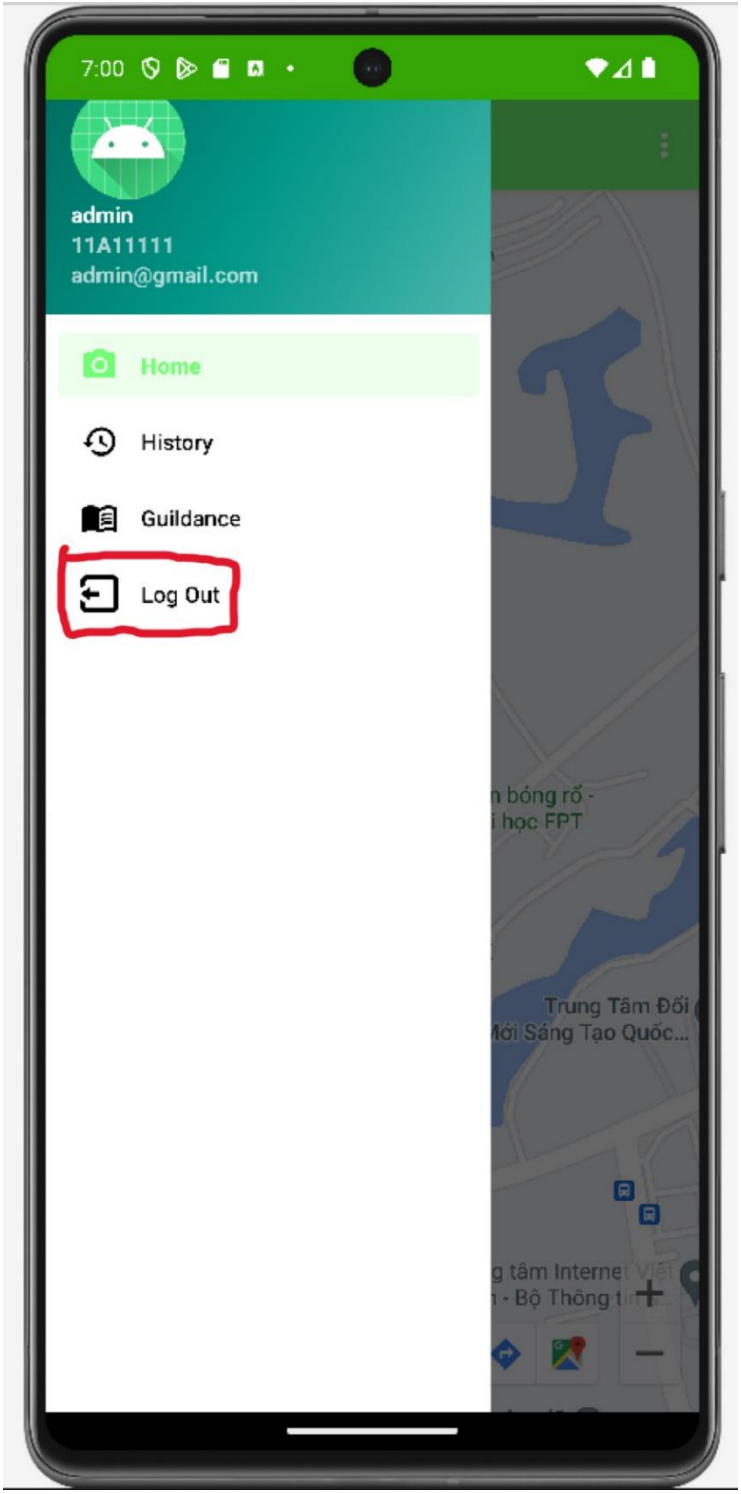


Figure III.3.12: Application Navigation Bar

Another feature is when the user is already has an account and login. When the user exits the app, it will remain log in

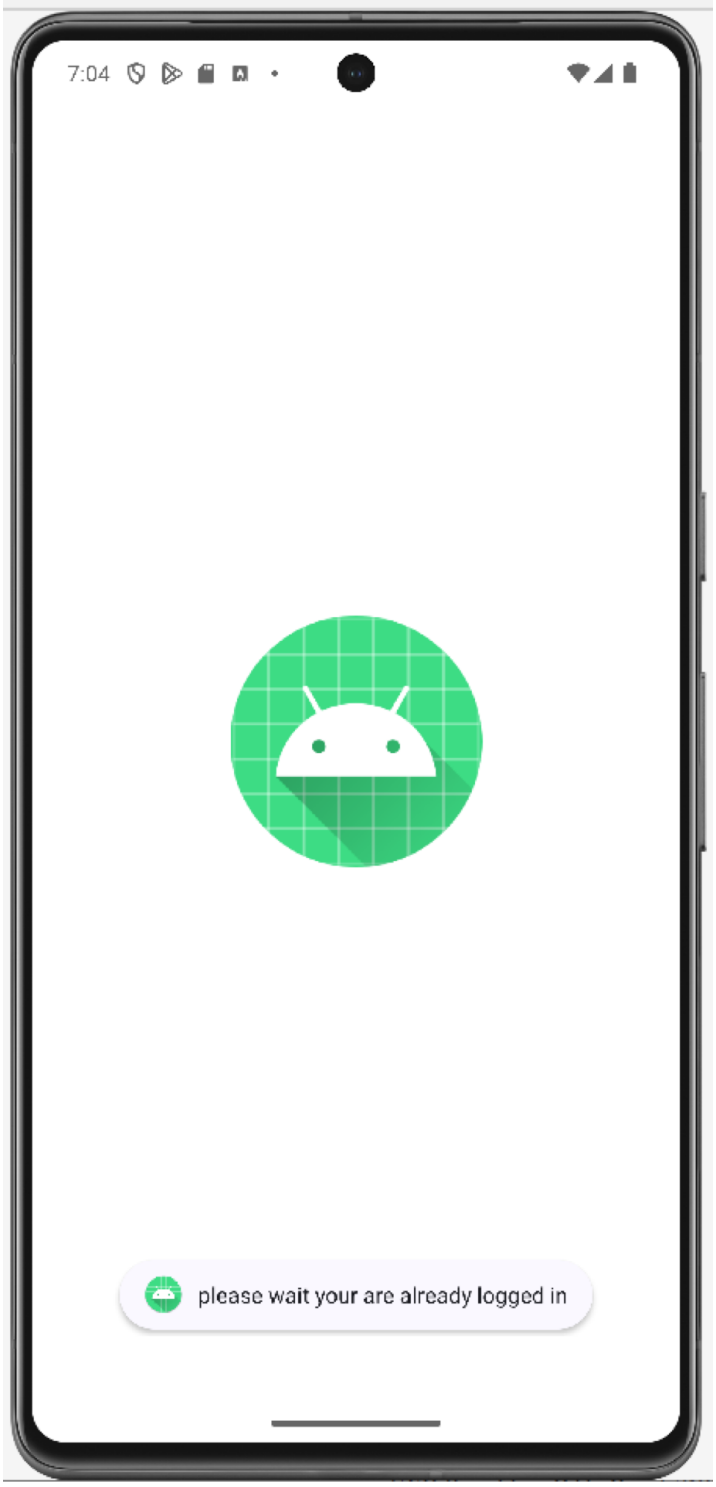


Figure III.3.13: Application Waiting Screen

IV Test Documentation

1. Hardware Test

1.1 Unit Test

1.1.1 Evaluating the Object Detection Capability of Ultrasonic Sensors
Objective: Test the actual measurable distance and stability of the HC-SR04 ultrasonic sensor.

According to published specifications, the HC-SR04 has a maximum operating range of 2cm to 450cm. Using the following code loaded onto an ESP32 CAM:

digitalWrite(trig,0); // turn off trig pin

delayMicroseconds(2);

digitalWrite(trig,1); // send pulse from trig pin

delayMicroseconds(5); // pulse length is 5 microseconds

digitalWrite(trig,0); // turn off trig pin

/* Calculate time */

// Measure the pulse width at the echo pin.

duration = pulseIn(echo,HIGH);

// Calculate distance to object.

Distance = int(duration/2/29.412);

The maximum distance detected was 115cm, far short of the datasheet's claims:

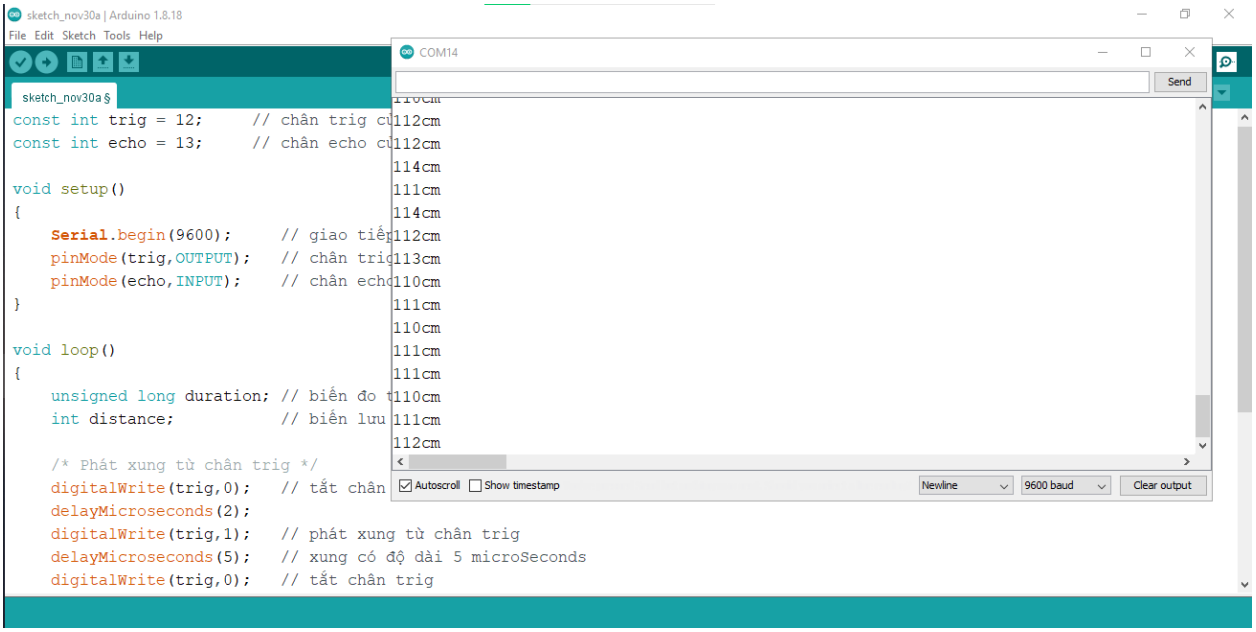


Figure IV.1.1: Image showing serial output with 115cm max distance

Next, we will proceed to measure the stability of the ultrasonic sensor. The data will be collected for 30 minutes under normal operating conditions and then presented in a chart for easy monitoring:

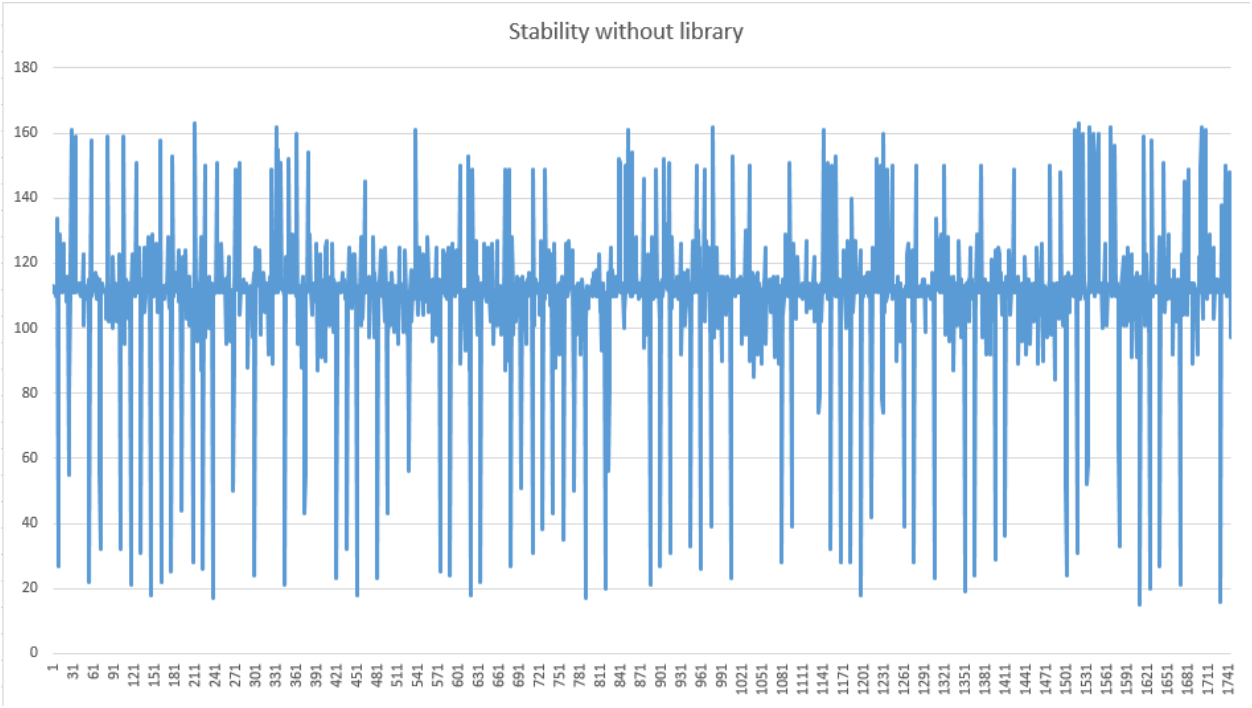


Figure IV.1.2: Image showing the stability of the HC-SR04 when not using the library

The results demonstrate two issues with relying solely on the default HC-SR04 behavior:

1. The maximum distance is only ~100cm versus the datasheet's 450cm claim.
2. The readings fluctuate wildly, indicating poor measurement stability.

To address these problems, the NewPing library was utilized, an optimized library that can be configured to increase the maximum object detection distance and eliminate unnecessary noise signals.

The NewPing library provides simple, fast and powerful communication with ultrasonic sensors. Initially, I was not satisfied with the performance of the ultrasonic sensor. I soon realized that the problem did not come from the sensor but was caused by the available ping and ultrasonic libraries. When I switched to using the NewPing library, I noticed that the problem had been partially solved: The NewPing library allows you to set the maximum distance via a function:

NewPing sonar(trigger_pin, echo_pin [, max_cm_distance])

And some methods of the library:

- *sonar.ping([max_cm_distance])* - Send a ping and get the echo time (in microseconds) as a result. *[max_cm_distance]* allows you to optionally set a new max distance.
- *sonar.ping_in([max_cm_distance])* - Send a ping and get the distance in whole inches. *[max_cm_distance]* allows you to optionally set a new max distance.

- *sonar.ping_cm([max_cm_distance])* - Send a ping and get the distance in whole centimeters. *[max_cm_distance]* allows you to optionally set a new max distance.
- *sonar.ping_median(iterations [, max_cm_distance])* - Do multiple pings (default=5), discard out of range pings and return median in microseconds. *[max_cm_distance*] allows you to optionally set a new max distance.
- *sonar.convert_in(echoTime)* - Convert echoTime from microseconds to inches.
- *sonar.convert_cm(echoTime)* - Convert echoTime from microseconds to centimeters.

Below, we will adjust the max_distance value and use the ping_median method with iteration = 10. The code when using max_distance = 200 and iteration = 5:

```
#include <NewPing.h
#define TRIGGER_PIN 12
#define ECHO PIN 13
#define MAX DISTANCE 300
#define IRETATIONS 5

NewPing sonar(trigPin, echoPin, MAX_DISTANCE) ;

long duration = 0, distance = 0;
void setup(){

// put your setup code here, to run once:

}

void loop() {

// put your main code here, to run repeatedly:

duration = sonar.ping_median (IRETATIONS);

distance = sonar.convert_cm(duration);

Serial.println(distance);

Serial.print ("cm");

}
```

Following is the data of the HC-SR04 ultrasonic sensor recorded for 15 minutes with different Max distances and Iterations.

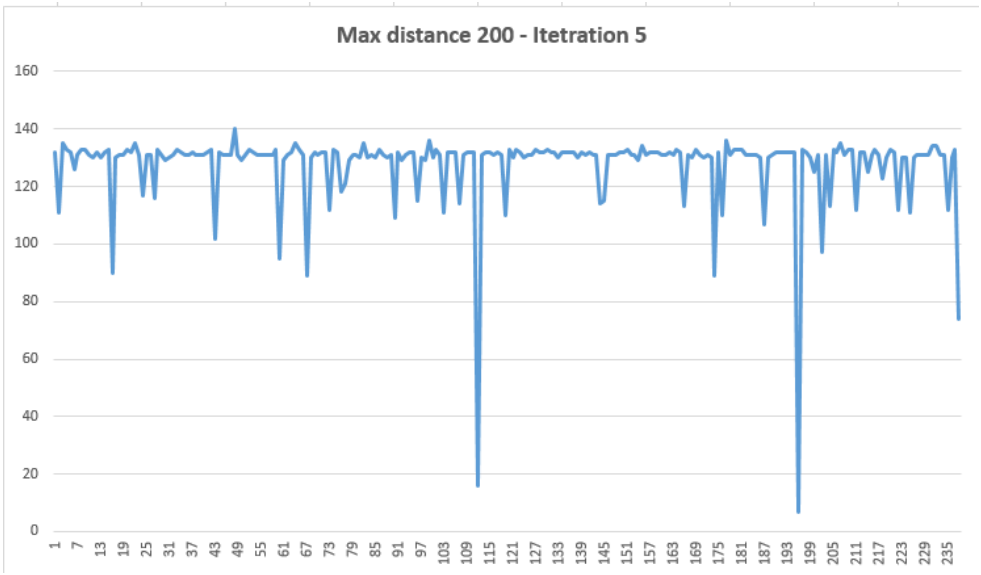


Figure IV.1.3: HC-SR04 data at max distance 200 and iterations 5

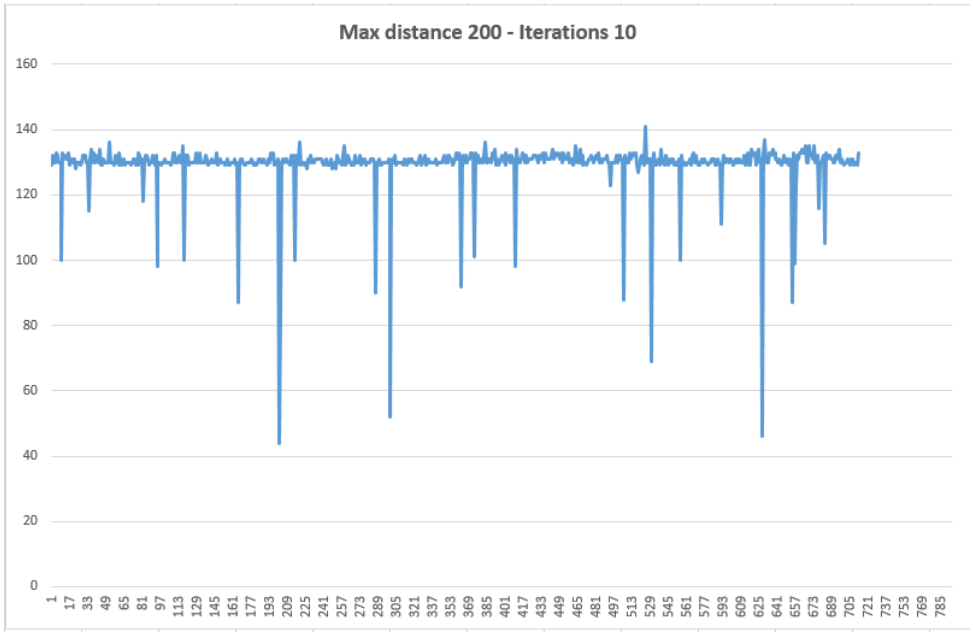


Figure IV.1.4: HC-SR04 data at max distance 200 and iterations 10

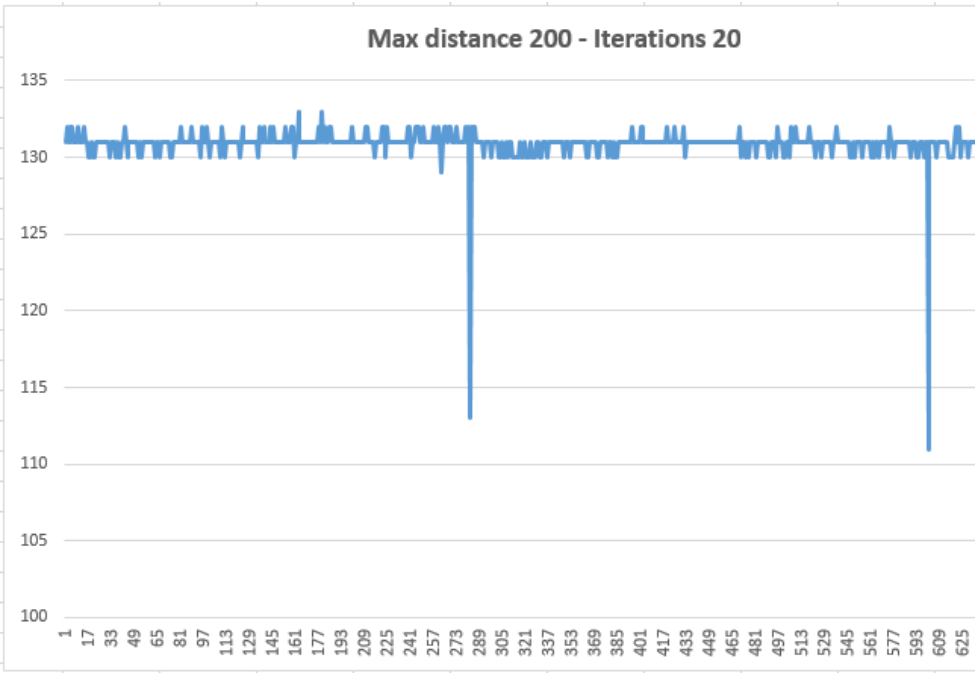


Fig IV.1.5: HC-SR04 data at max distance 200 and iterations 20

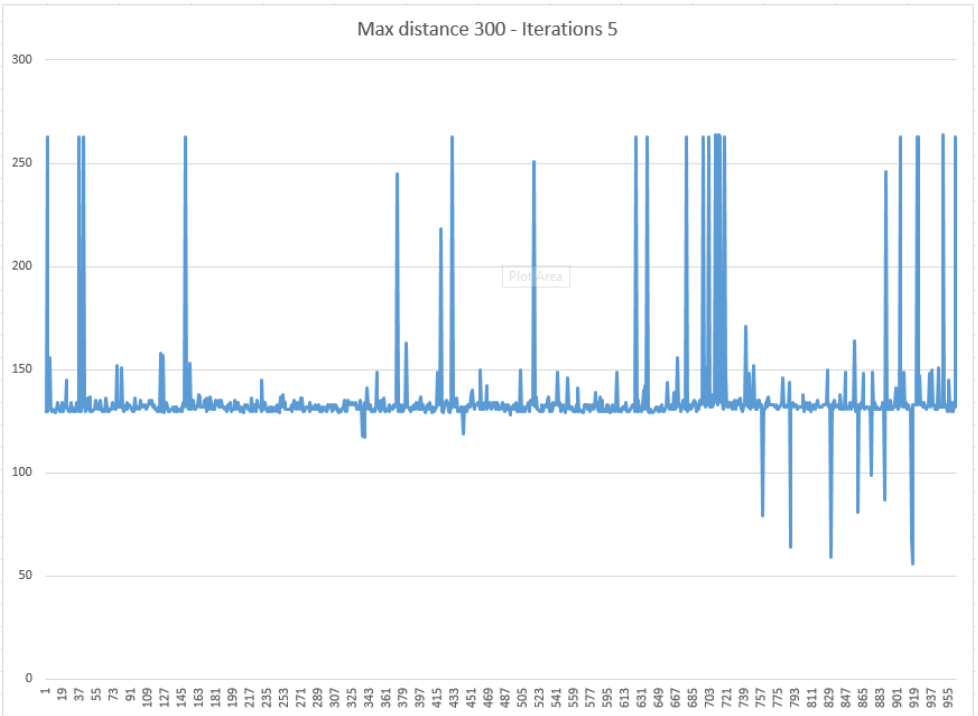


Figure IV.1.6: HC-SR04 data at max distance 300 and iterations 5

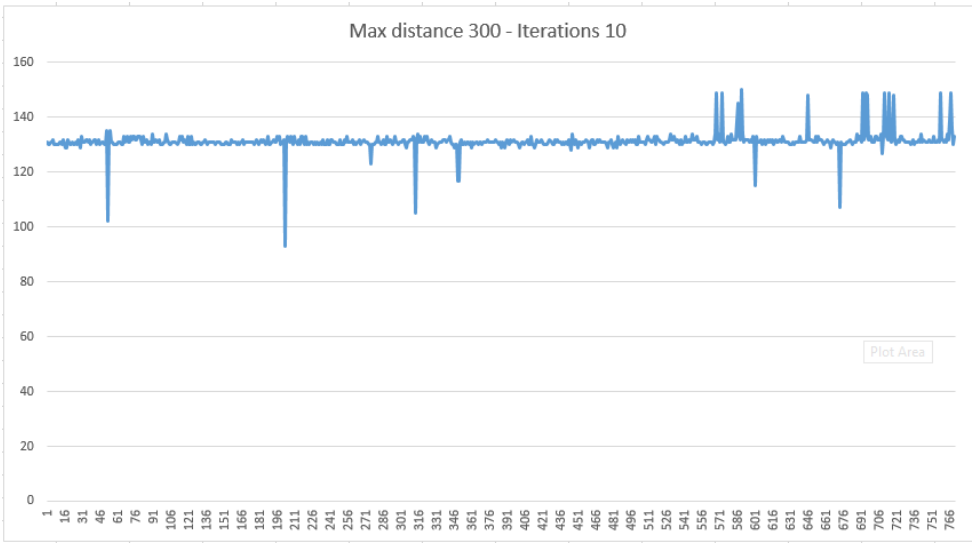


Figure IV.1.7: HC-SR04 data at max distance 300 and iterations 10

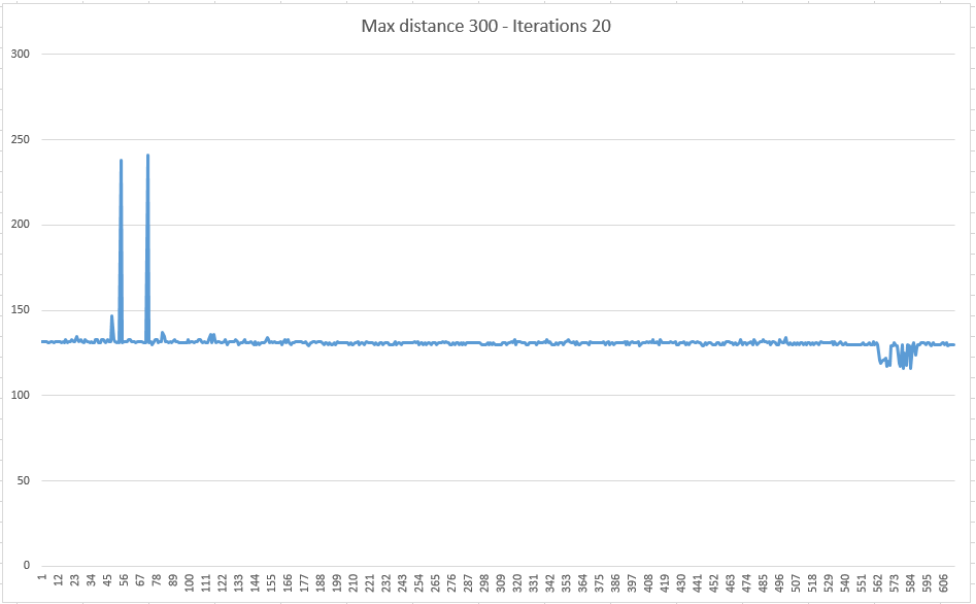


Figure IV.1.8: HC-SR04 data at max distance 300 and iterations 20

Based on the results above, with a max distance setting of 300 and 20 iterations, the best stability is achieved.

1.1.2 Image Capture Capability of ESP32 CAM

Objective: Test the image capture capability of the ESP32 Cam with OV2640 camera and image quality under different lighting conditions.

1.1.2.1 Image Capture Capability Testing

Consecutive test photos were taken with the ESP32 Cam to check for consistency. The first photo captured only a ceiling backdrop. The second introduced a hand gesture in the frame. However, the images obtained from these two shots are hardly different.

This is surprising because It is expected the change in scenery (shooting the ceiling and shooting when raising hand) would create a clear difference in the images, but that was not the case. The fact that the photos look alike shows that retrieving data from the ESP32 Cam is not working as expected.

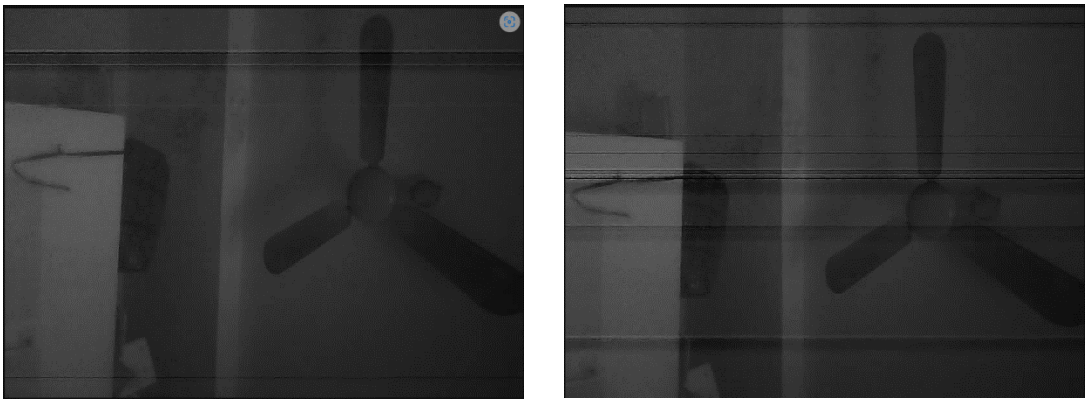


Figure IV.1.9: Images show ESP32 CAM capture error

It had to take 2-3 more consecutive shots before the ESP32 CAM returned the photo with the raised hand

Fig IV.1.10: Photo with the raised hand

This seems to be an effect of the camera operating like a queue; with the frame buffer holding images (fb_count+1).

- When taken a picture it gets added to the frame buffer, This command esp_camera_fb_get() pushing out older images when necessary.

- But the pointer returned from this function will be the oldest image in the frame buffer, not the latest.
- This can get quite extreme, it is entirely possible for an image minutes, hours (or more) old to be returned as the 'latest' if you just don't regularly trigger capturing and aren't streaming at the same time.

One solution that may help is to use a setting called **CAMERA_GRAB_LATEST** in the setup section of the code. By adding the following code line: **config.grab_mode = CAMERA_GRAB_LATEST;**

When applying this line, the ESP32 Cam will return the latest image that the camera has received, helping to get the images in the desired order and avoid getting the oldest image from the buffer.

Code lines added:

```
config.xclk freq hz = 20000000;
config.pixel format = PIXFORMAT_JPEG;

// init with high specs to pre-allocate larger buffers

if (psramFound()) {
config.frame size = FRAMESIZE_SVGA;
config.jpeg quality = 10; //0-63 lower number means higher quality
config.fb_count = 2; config.grab mode = CAMERA_GRAB_LATEST;
} else {
config.frame size = FRAMESIZE_CIF;
config.jpeg quality = 12; //0-63 lower number means higher quality
config.fb_count = 1;
config.grab mode = CAMERA_GRAB_LATEST; }
```

1.1.2.2 Test the image quality under different lighting conditions

1.2.2.1.1 Bright, direct sunlight condition

Fig IV.1.11: Image of front and rear plate

Under bright, direct sunlight: Images exhibited overexposure; license plate unviewable

1.2.2.1.2 Cloudy, shade condition

Figure IV.1.12: Image of a license plate

Figure IV.1.13: Image of a license plate

In cloudy environment display images with accurate color and details like license plate numbers.

1.2.2.1.3 Indoor lighting condition

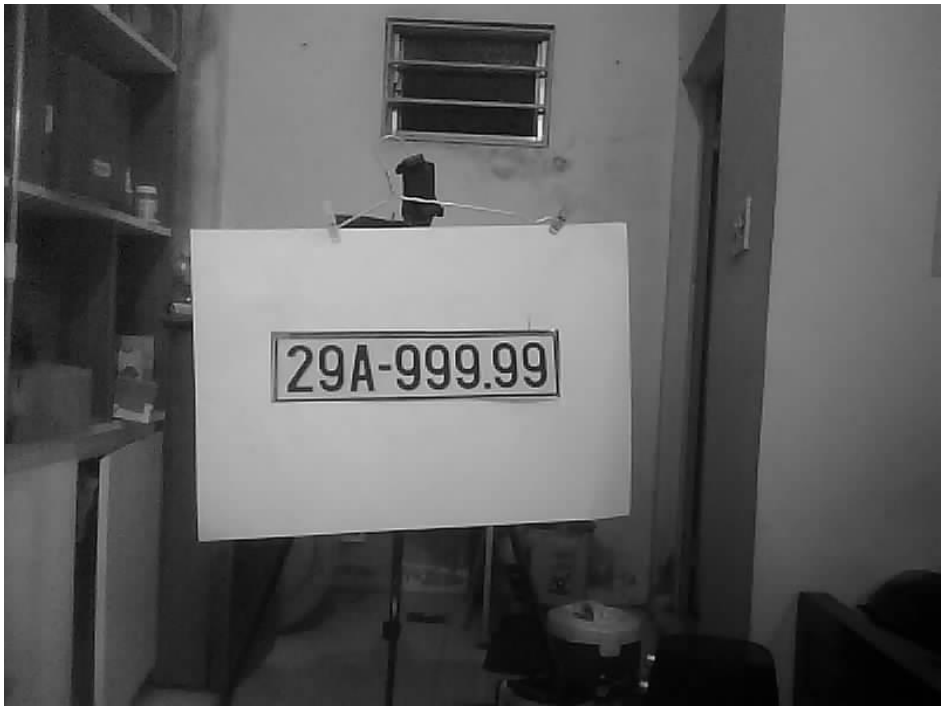


Figure IV.1.14: Picture of a license plate

Figure IV.1.15: Picture of a license plate

With typical indoor condition: Similar to cloud condition, indoor lighting provided quality imaging of license plate

1.2.2.1.4 Low light condition

Figure IV.1.16: Picture of a license plate

Figure IV.1.17: Picture of a license plate

Low light condition led to reduced image clarity because of higher noise. However, license plates remained identifiable.

1.1.3 Evaluation of License Plate Recognition Accuracy

Objective: Assess the accuracy of license plate information extraction under various real-world conditions.

Since license plate heights vary between cars, to optimize information extraction, the idea was to measure the distance from the top and bottom edges of the license plate to the ground. From there, it can calculate the average distance from the license plate to the ground. Applying this method to each vehicle will help to calculate the optimal average height to position the camera, thereby improving the ability to capture images and extract information from license plates most effectively.

Figure IV.1.18: Picture of a car

Survey of license plate heights:

	Font plate height		Back plate height		Average font plate height	Average back plate height
	Lower border	Upper border	Lower border	Upper border		
Sedan (Kia)	42	53	32	48.5	47.5	40.25
SUV(Toyota cross)	40	56.5	100	111	48.25	105.5
Sedan (Camry)	40	51	76	92.5	45.5	84.25
Sedan(Honda)	38	49	78	94.5	43.5	86.25
SUV(Kia Sorento)	46	57	83	96.5	51.5	89.75
Sedan (Kia morning)	43	54	33	49.5	48.5	41.25
Sedan (huyendai)	42	53	76	92.5	47.5	84.25
sedan (camry)	40	51	68	84.5	45.5	76.25
suv(honda)	45	61.5	83	96.5	53.25	89.75
suv (peugout)	44	60.5	34	50.5	52.25	42.25
Sedan (audi)	36	52.5	72	88.5	44.25	80.25
Suv (Mitsu)	43	59.5	68	84.5	51.25	76.25
SUV (huyndai)	40	56.5	78	94.5	48.25	86.25
SUV(huyndai)	46	57	86	102.5	51.5	94.25
Suv (Mitsu)	49	60	87	103.6	54.5	95.3
Sedan(Volvo)	36	47	75	91.5	41.5	83.25
Sedan(Suzuki)	36	52.5	39	55.5	44.25	47.25
sedan(kia)	36	52.5	72	88.5	44.25	80.25
sedan(KN)	36	52.5	72	72-88	44.25	72
sedan(vinfast	28	44.5	49	65.5	36.25	57.25
Sedan(Honda)	34	45	47	53.5	39.5	50.25

Figure IV.1.19: Excel of license plate heights

After determining the ideal camera height, the next step is to choose an appropriate distance to take optimal license plate photos, avoiding interference from the surrounding environment. So choose a range of distances from the camera to the license plates of 50cm to 100cm

After taking photos of license plates at these distances and performed pre-processing to find regions containing license plates. The result was a series of rectangles, so to determine the region accurately containing the license plate.

Figure IV.1.20 Image show after drawing the approximated polygons

Image name	Weight	Height	Area ratio	Edge ratio	License plate type
60 (1)	597	272	7.567248005	2.194852941	Short
60 (2)	593	278	7.453868271	2.133093525	Short
60 (3)	618	321	6.194235248	1.925233645	Short
60 (4)	525	294	7.961127308	1.785714286	Short
60 (5)					Short
60 (6)					Short
60 (7)					Short
60 (8)					Short
60 (9)	958	203	6.318582433	4.719211823	Long
60 (10)					Long
60 (11)					Short
60 (12)	910	229	5.896636115	3.973799127	Long
60 (13)	909	207	6.530508123	4.391304348	Long
60 (14)					Short
60 (15)	643	344	5.555354624	1.869186047	Short
60 (16)	606	291	6.968119492	2.082474227	Short
60 (17)					Short
60 (18)	642	341	5.61295804	1.882697947	Short
60 (19)	1113	253	4.363806825	4.399209486	Long
60 (20)	1121	245	4.474139344	4.575510204	Long
60 (21)	1029	202	5.911728199	5.094059406	Long
60 (22)	1039	203	5.825988422	5.118226601	Long
60 (23)	663	328	5.650590443	2.021341463	Short
60 (24)	656	322	5.817300409	2.037267081	Short

Figure IV.1.21 Image of Excel for record parameters

The maximum and minimum ratio values mean if a rectangle in the image satisfies the area ratio and edge ratio then that rectangle is likely the license plate.

Max value			8.007507038	6.005555556
Min value			4.363806825	1.785714286

Figure IV.1.22: Image of Min Max value in Excel

The final step was to remark on the separated license plate images, was it be able to read the text and accurately recognize it or not

Image name	Weight	Height	Area ratio	Edge ratio	License plate type	Can detect plate	Can read text	Can read exactly
60 (1)	597	272	7.567248005	2.194852941	Short	1	1	0
60 (2)	593	278	7.453868271	2.133093525	Short	1	1	0
60 (3)	618	321	6.194235248	1.925233645	Short	1	1	0
60 (4)	525	294	7.961127308	1.785714286	Short	1	1	0
60 (5)					Short			
60 (6)					Short			
60 (7)					Short			
60 (8)					Short			
60 (9)	958	203	6.318582433	4.719211823	Long	1	1	1
60 (10)					Long			
60 (11)					Short			
60 (12)	910	229	5.896636115	3.973799127	Long	1	1	1
60 (13)	909	207	6.530508123	4.391304348	Long	1	1	1
60 (14)					Short			
60 (15)	643	344	5.555354624	1.869186047	Short	1	1	1
60 (16)	606	291	6.968119492	2.082474227	Short	1	1	0
60 (17)					Short			
60 (18)	642	341	5.61295804	1.882697947	Short	1	1	1
60 (19)	1113	253	4.363806825	4.399209486	Long	1	1	1
60 (20)	1121	245	4.474139344	4.575510204	Long	1	1	1
60 (21)	1029	202	5.911728199	5.094059406	Long	1	1	0
60 (22)	1039	203	5.825988422	5.118226601	Long	1	1	0
60 (23)	663	328	5.650590443	2.021341463	Short	1	1	1
60 (24)	656	322	5.817300409	2.037267081	Short	1	1	1

Figure IV.1.23: Image of Excel for record parameters

The results were quite good but there were still some cases where the license plates could not be identified under certain lighting conditions:

1. Sunny, harsh lighting:

Fig IV.1.24: Image of license plate under harsh lighting condition

After the pre-processing, the program was unable to detect the region containing the license plate.

2. Plate is overexposed:

Figure IV.1.25: A short License Plate

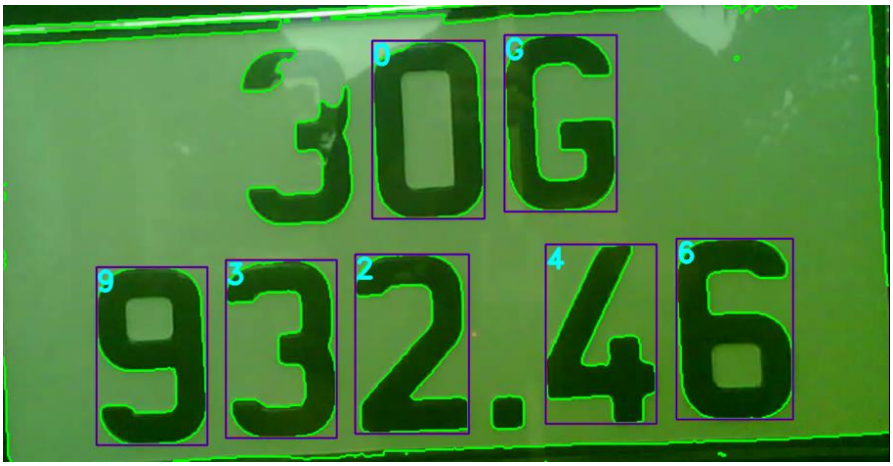


Figure IV.1.26: A short License Plate

Plate is overexposed led to loss of characters

3. Plate is dirty

Figure IV.1.27: A dirty long license plate

Plate is dirty lead to undetectable of the character

4. Mistake in recognizing character

Figure IV.1.28: Some mistakes in recognizing character on the license plate

5. Solution

The solution is retraining the model by adding more font variations of license plates and changing tilt angles. This involves adding variants of letters and numbers tilted at different angles such as 5 degrees, 10 degrees, -5 degrees, -10 degrees. The goal is to increase accuracy when processing images of license plates.

Figure IV.1.29: Image for retraining the model

After training, the ability to accurately read license plates increased significantly:

	First tranning			Second trainning	
	Can detect plate	Can read text	Can read exactly	Can read text	Can read exactly
Total	32	21	9	33	21
Detection rate	55%	36%	43%	57%	64%

Figure IV.1.30 Image of Excel for License plate character read ratio at 50 cm distance

	First tranning			Second trainning	
	Can detect plate	Can read text	Can read exactly	Can read text	Can read exactly
Total	55	55	22	54	32
Detection rate	79%	79%	40%	77%	59%

Figure IV.1.31: Image of Excel for License plate character read ratio at 60 cm distance

	First tranning			Second trainning	
	Can detect plate	Can read text	Can read exactly	Can read text	Can read exactly
Total	50	50	24	50	37
Tỷ lệ phát hiện	75%	75%	48%	75%	74%

Figure IV.1.32: Image of Excel for License plate character read ratio at 70 cm distance

	First tranning			Second trainning	
	Can detect plate	Can read text	Can read exactly	Can read text	Can read exactly
Total	41	41	23	41	27
Tỷ lệ phát hiện	75.93%	75.93%	42.59%	76%	65.85%

Figure IV.1.33: Image of Excel for License plate character read ratio at 80 cm distance

	First tranning			Second trainning	
	Can detect plate	Can read text	Can read exactly	Can read text	Can read exactly
Total	18	14	8	14	13
Tỷ lệ phát hiện	58.06%	45.16%	57.14%	45%	93%

Figure IV.1.34: Image of Excel for License plate character read ratio at 90 cm distance

	First tranning			Second trainning	
	Can detect plate	Can read text	Can read exactly	Can read text	Can read exactly
Total	17	10	2	9	3
Tỷ lệ phát hiện	61%	36%	20%	32%	33%

Figure IV.1.35: Image of Excel for License plate character read ratio at 100 cm distance

According to the images above, distances of 60-80cm give the best license plate detection and recognition results.

1.1.4 Raspberry Pi capability to access Firebase

```python
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore

cred = credentials.Certificate("/home/pi/Documents/Detect_plate/serviceAccountKey.json")
firebase_admin.initialize_app(cred)

# Kết nối tới Firestore
db = firestore.client()

user_ref = db.collection("user").document("2")
docs = user_ref.get()

if docs.exists:
    print(f"Document dat{docs.to_dict()}")
```

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ /home/pi/Documents/Detect_plate/myenv/bin/python3 /home/pi/Do
cuments/Detect_plate/test_fire_store.py
Document dat{'phone number': '0987654321', 'id': '098765432123', 'password': '12
3123123', 'email': 'nghia37@gmail.com', 'name': 'Ha Minh Nghia', 'plate number':
 '29N15675'}
pi@raspberrypi:~ $
```

Figure IV.1.36 Images of the results returned from Firebase on the Raspberry Pi

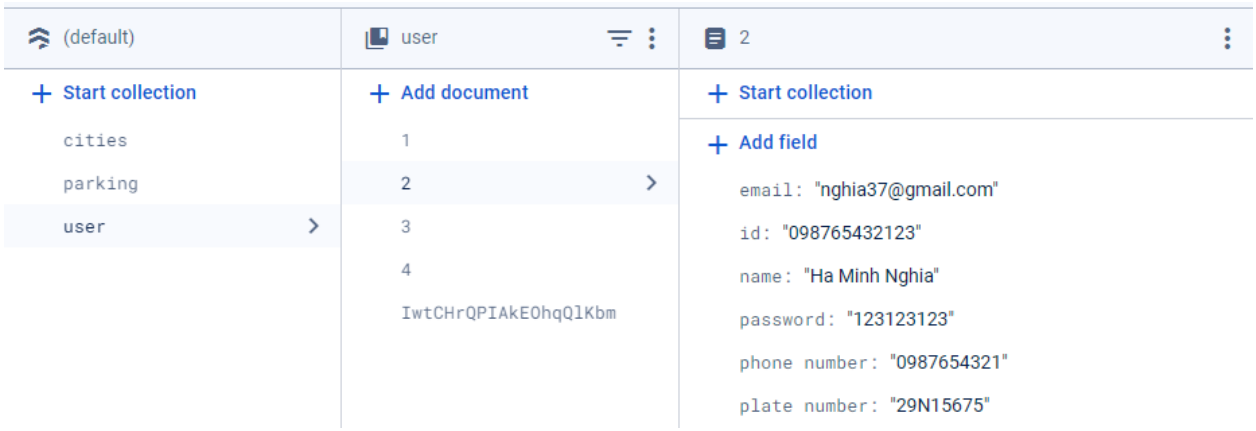


Figure IV.1.37: Data on firebase

When the Raspberry Pi successfully connected to Firebase, it printed the corresponding data from the cloud to the screen.

1.2 Integration Test

1.2.1 Ultrasonic Sensor Integration

Objective: Check if when entering within range, the ESP32 CAM will take a photo

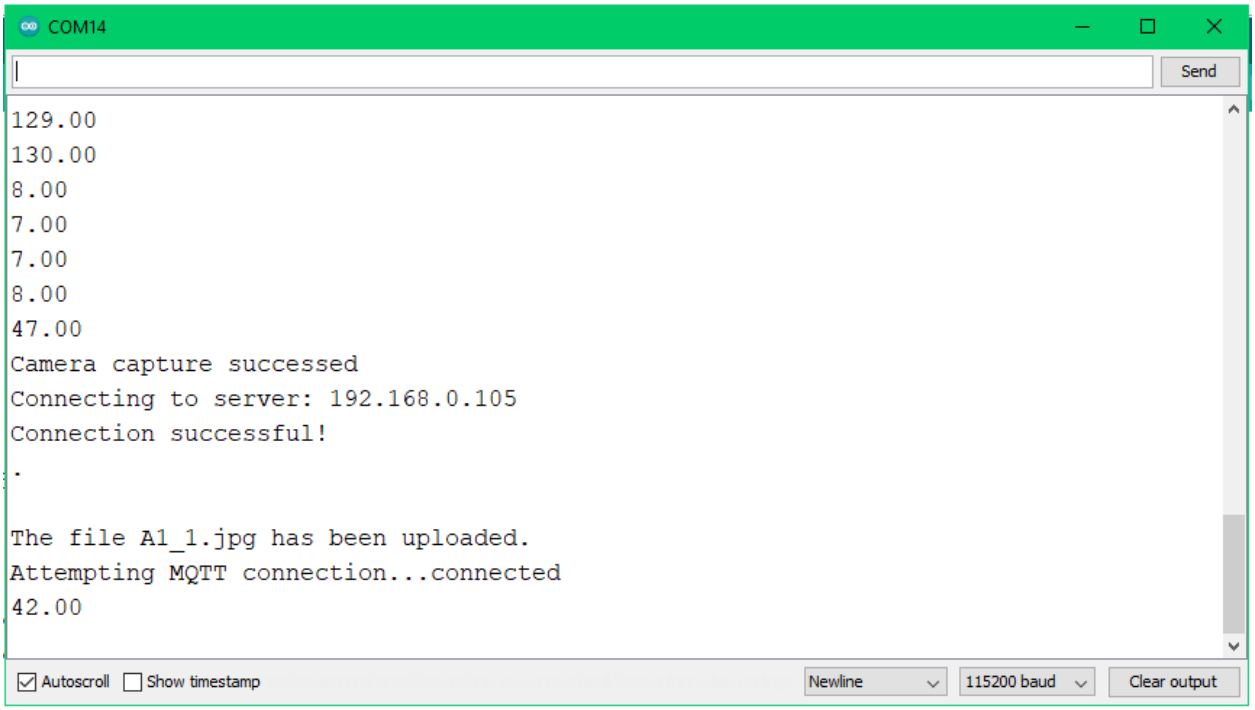


Fig IV.1.38: The interaction between the ESP32 and ultrasonic sensor

When an object is within 40-80 cm range, the ESP32 CAM will take a photo, connect to the internet to send the image to the Raspberry Pi.

1.2.2 LAMP Server Communication

Figure IV.1.39: Image of hand gesture

After the ESP32 CAM takes and sends a photo, this is the image the raspberry received. It proves the ESP32 communicates very well with LAMP.

We can also view the captured images via the IP address of the raspberry through Apache.

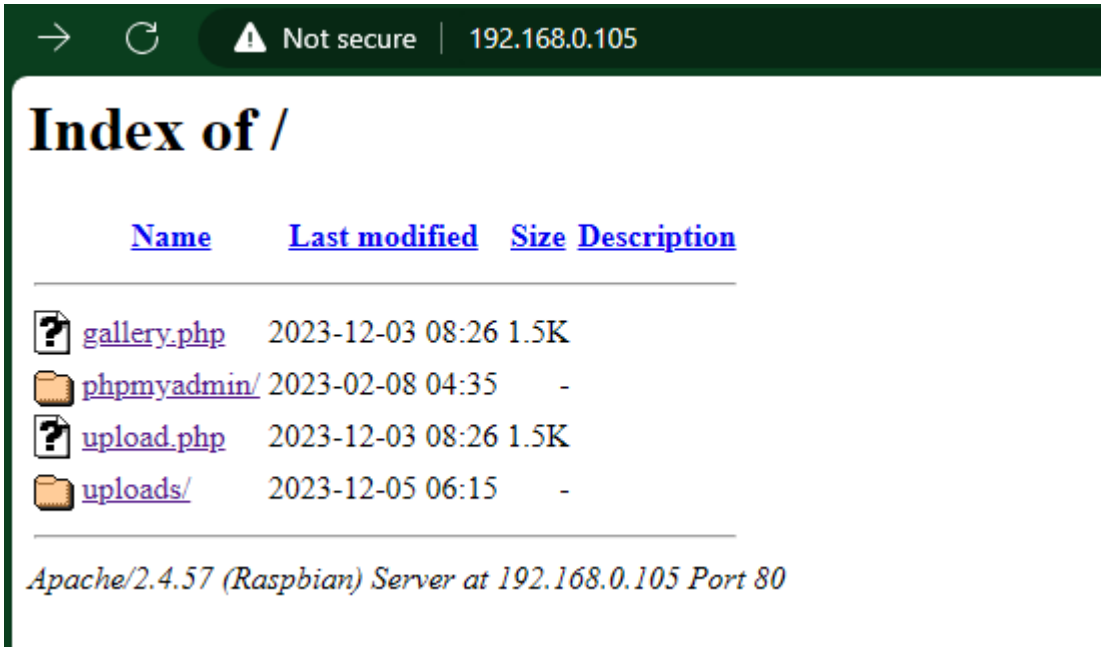


Figure IV.1.40: Image of the screen

Figure IV.1.41: Image of screen

The previous images prove ESP32 communicates very well with LAMP server.

1.2.3 MQTT Messaging

To check if the ESP32 CAM and Raspberry Pi can publish and subscribe to the following message segments and then display them on the screen. Write command code on the raspberry pi to publish a continuous and repeating signal, running from 1 - 20 and with the topic 'rpi/broadcast':

```
k=0
   while True:
   k=k+1 if(k>20):
      k=1

   try:
       msg =str(k)
       pubMsg = client.publish(
            topic='rpi/broadcast',
            payload=msg.encode('utf-8'),
            qos=0,
       )
       pubMsg.wait_for_publish()
       print(pubMsg.is_published())

   except  Exception as e:
       print(e)
time.sleep(2)
```

The ESP32 will receive the corresponding signal

```
Message arrived on topic: rpi/broadcast. Message: 16
130.00
Message arrived on topic: rpi/broadcast. Message: 17
131.00
130.00
Message arrived on topic: rpi/broadcast. Message: 18
130.00
130.00
Message arrived on topic: rpi/broadcast. Message: 19
129.00
Message arrived on topic: rpi/broadcast. Message: 20
129.00
130.00
130.00
Message arrived on topic: rpi/broadcast. Message: 1
130.00
```

Figure IV.1.42: Serial Monitor shows the result of ESP32 CAM received

Conversely, when there is a signal from the ultrasonic sensor, the ESP32 CAM also publishes a 1 0 signal sent out. When subscribing with the corresponding topic, the Raspberry Pi also receives the signal.

```
pi@raspberrypi:~ $ /home/pi/Documents/Detect_plate/myenv/bin/python3 /home/pi/Do
cuments/Detect_plate/client_sub.py
......client setup complete............
Connected to MQTT server
ESP sensor1 data:  1
ESP sensor1 data:  0
ESP sensor1 data:  1
ESP sensor1 data:  0
```

Figure IV.1.43: Raspberry receives the message from ESP32 CAM

2. Software Test

Use Case

There are only two Interface that have user stories which is Login Interface and Sign-up Interface

2.1 Login Interface

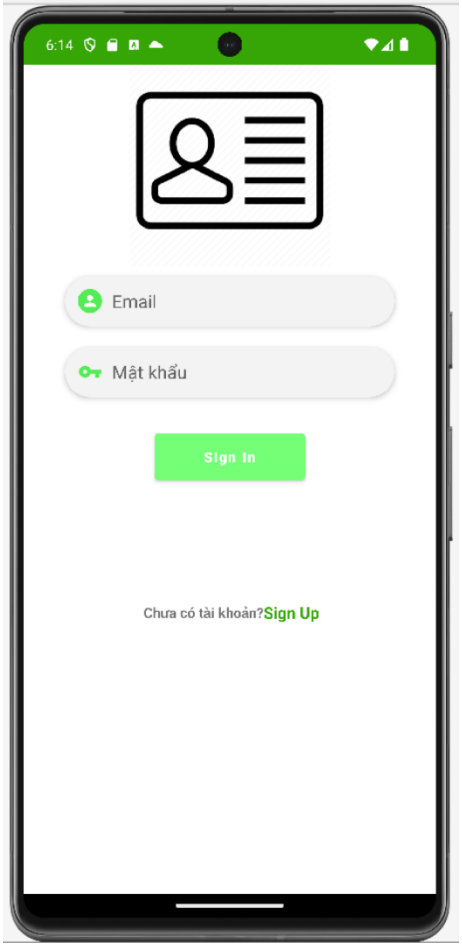


Figure IV.2.1: Application Login Interface

On the login screen, users enter their email address and password to access their existed account, then tap a Sign in button which invokes button verification:

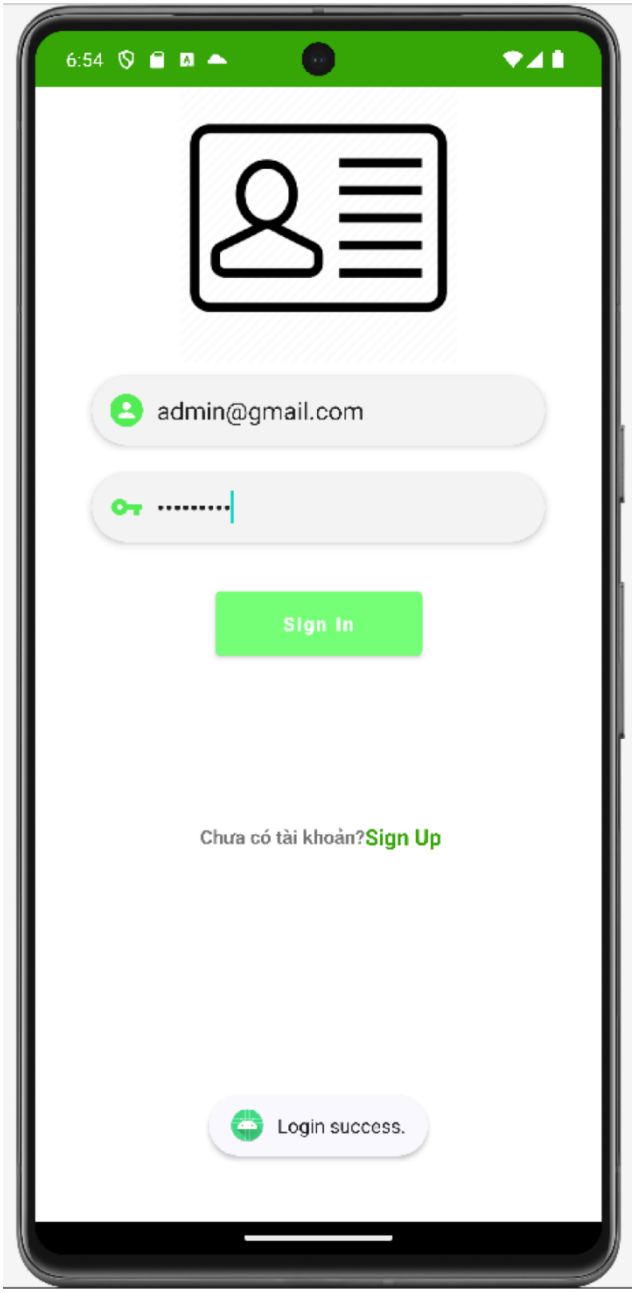


Figure IV.2.2: Login Success on Application Login Interface

If the email and password is matches the existed data, a button displays: “Login Success”

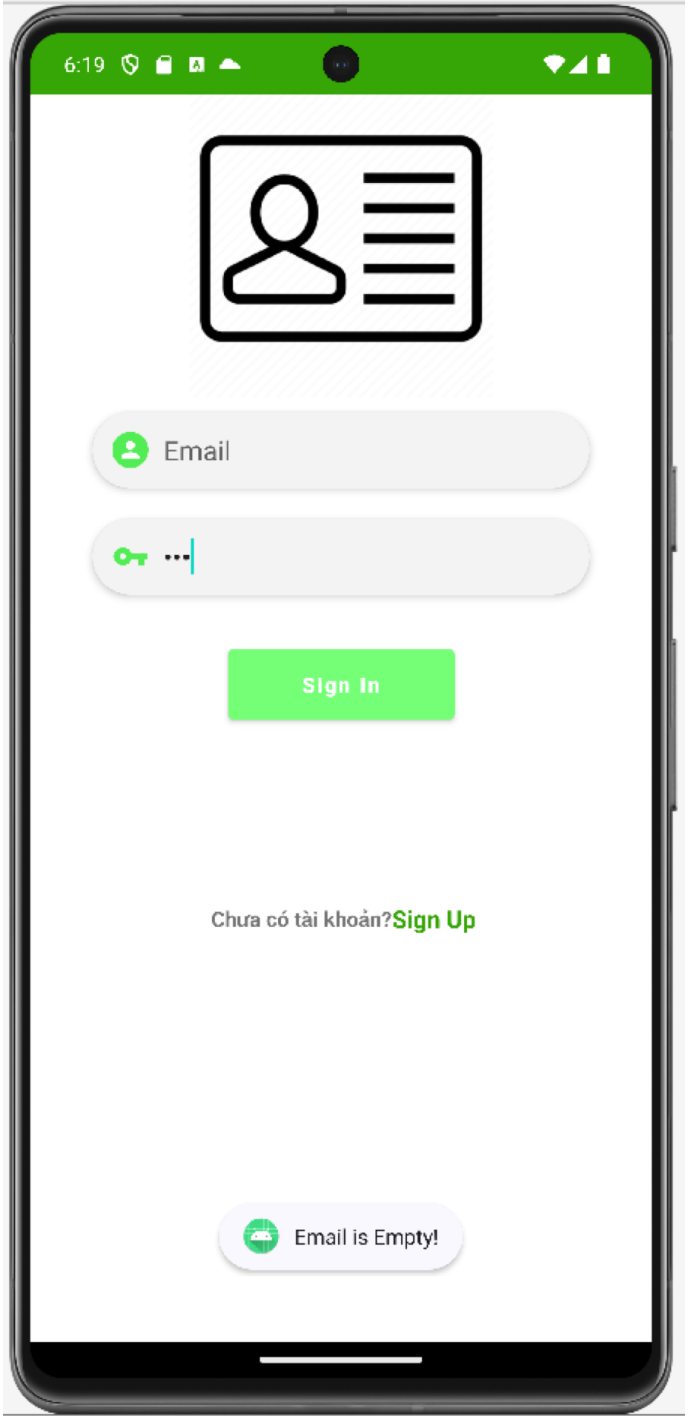


Figure IV.2.3: Application Login Interface

The app will not let user sign in, if the user emptied email section: “Email is Empty!”

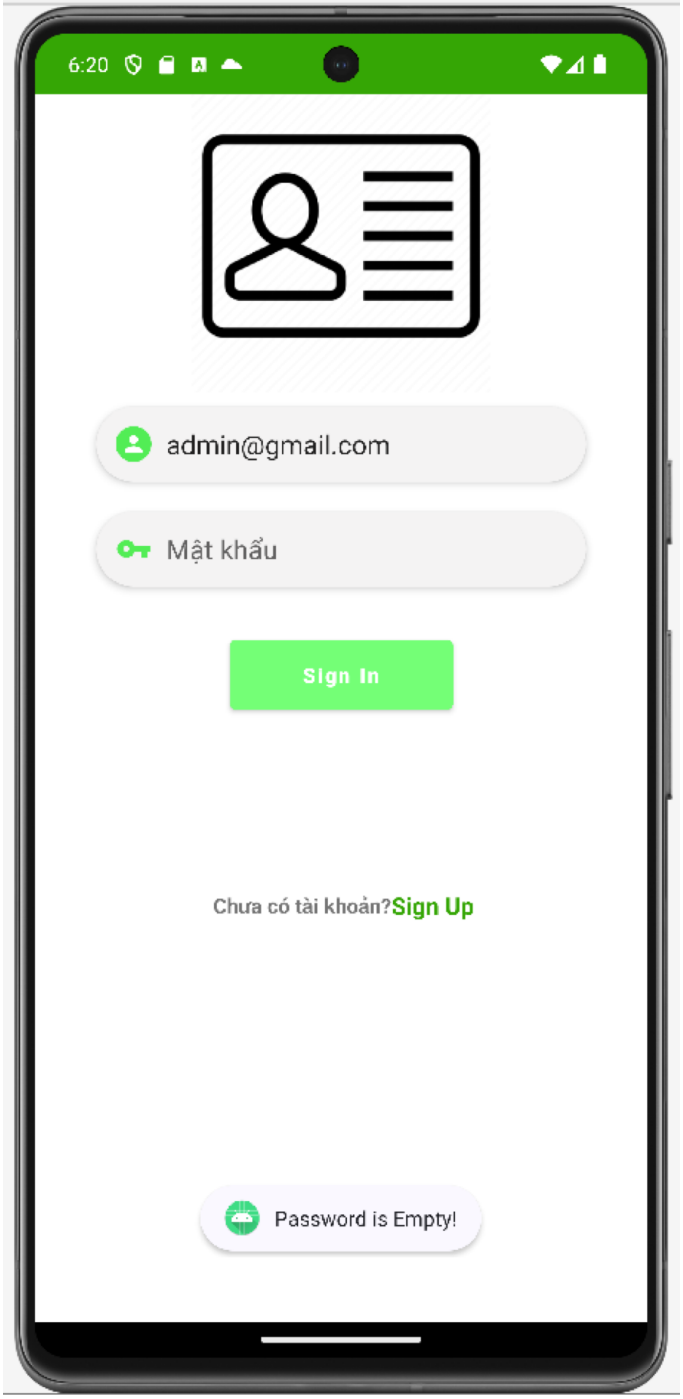


Figure IV.2.4: Application Login Interface

For an unknown password, a message shows: “Password is Empty!”

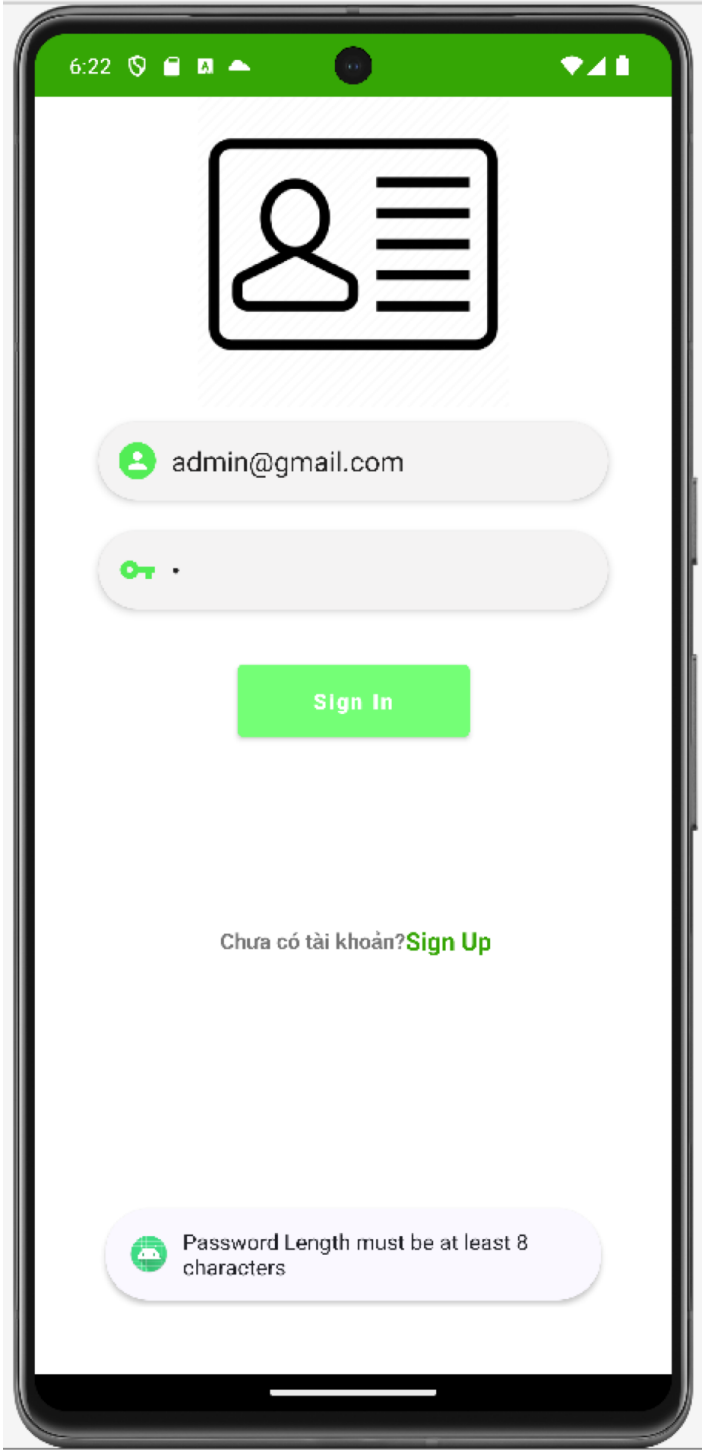


Figure IV.2.5 Application Login Interface

Before verification, password length gets checked client-side. If less than 8 characters, a warning prompts instead of allowing submission: “Password length must be at least 8 characters”

2.2 Sign Up Interface

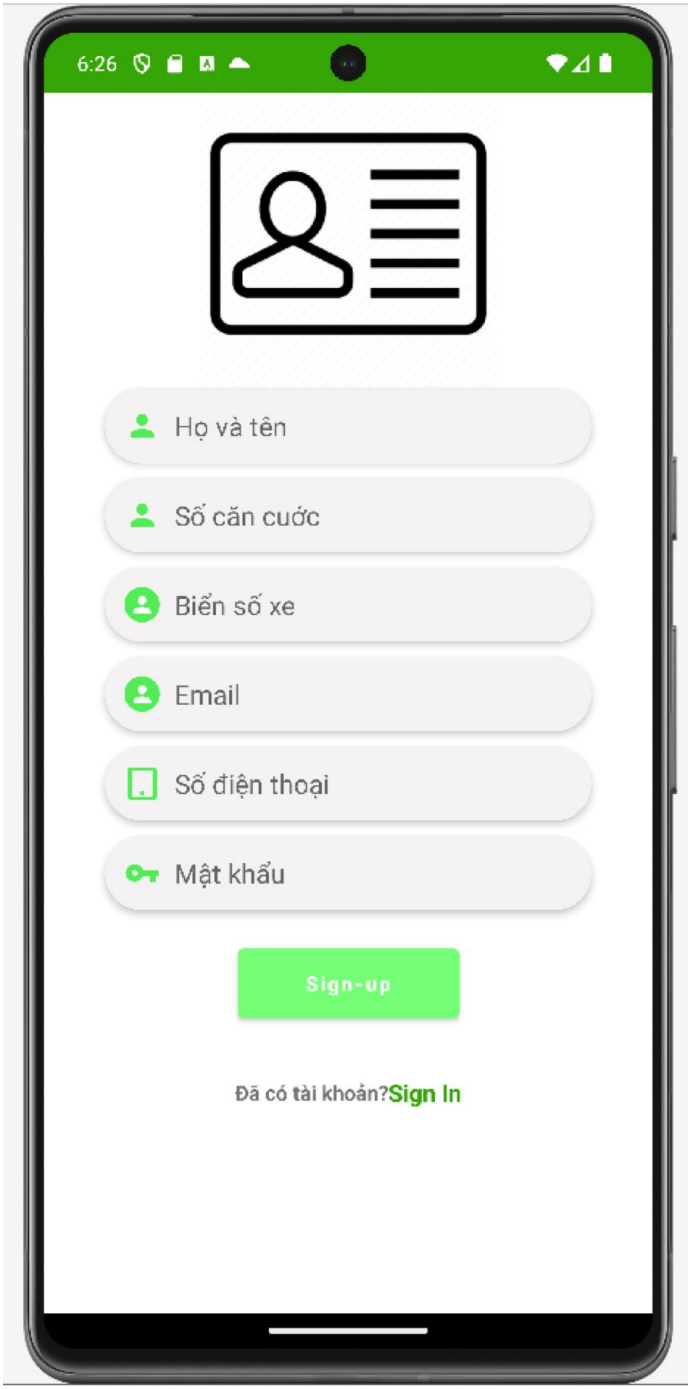


Figure IV.2.6 Application Sign Up Interface

After user choose Registration, it will appear sign up interface, users enter personal details including Name, ID number, License Plate, Email, Phone Number and desired Password to creating a new account. Server-side validation checks info correctness before inserting the data into backend databases to complete sign up. After sign up successfully, the screen will change to the login interface

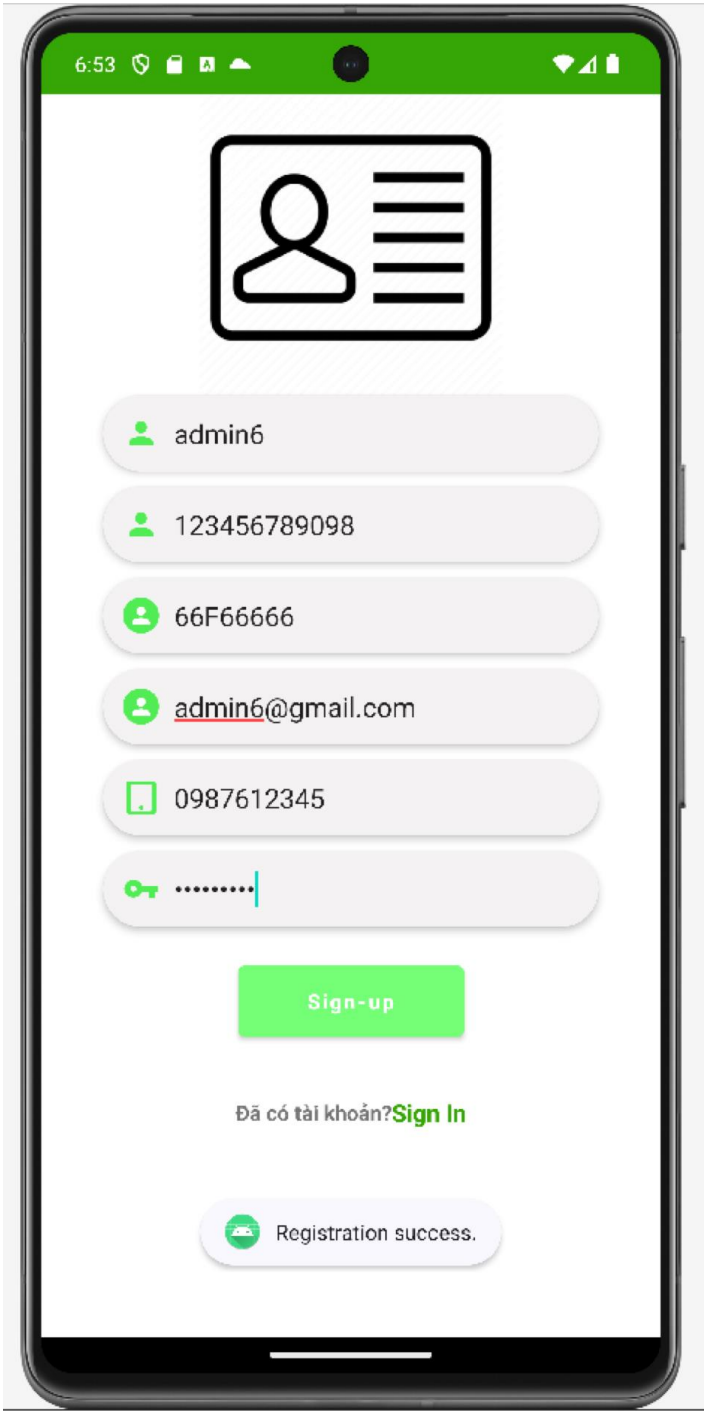


Figure IV.2.7: Application Sign Up Interface

After all details are filled, the notifications: “Registration Success” appear, and it will change the screen to Login Interface

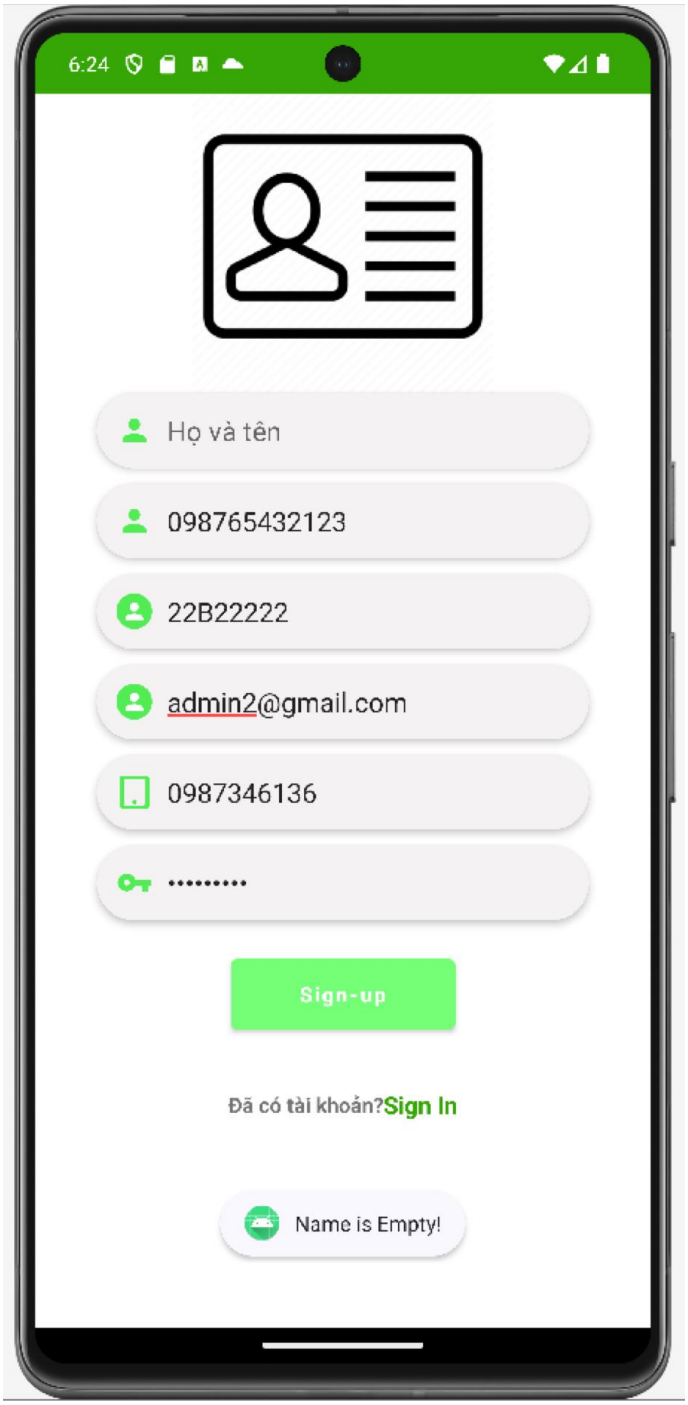


Figure IV.2.8 Application Sign Up Interface

If the user not provide their Name, it will prevent user to the next screen, the screen appears a notification: “Name is Empty!”

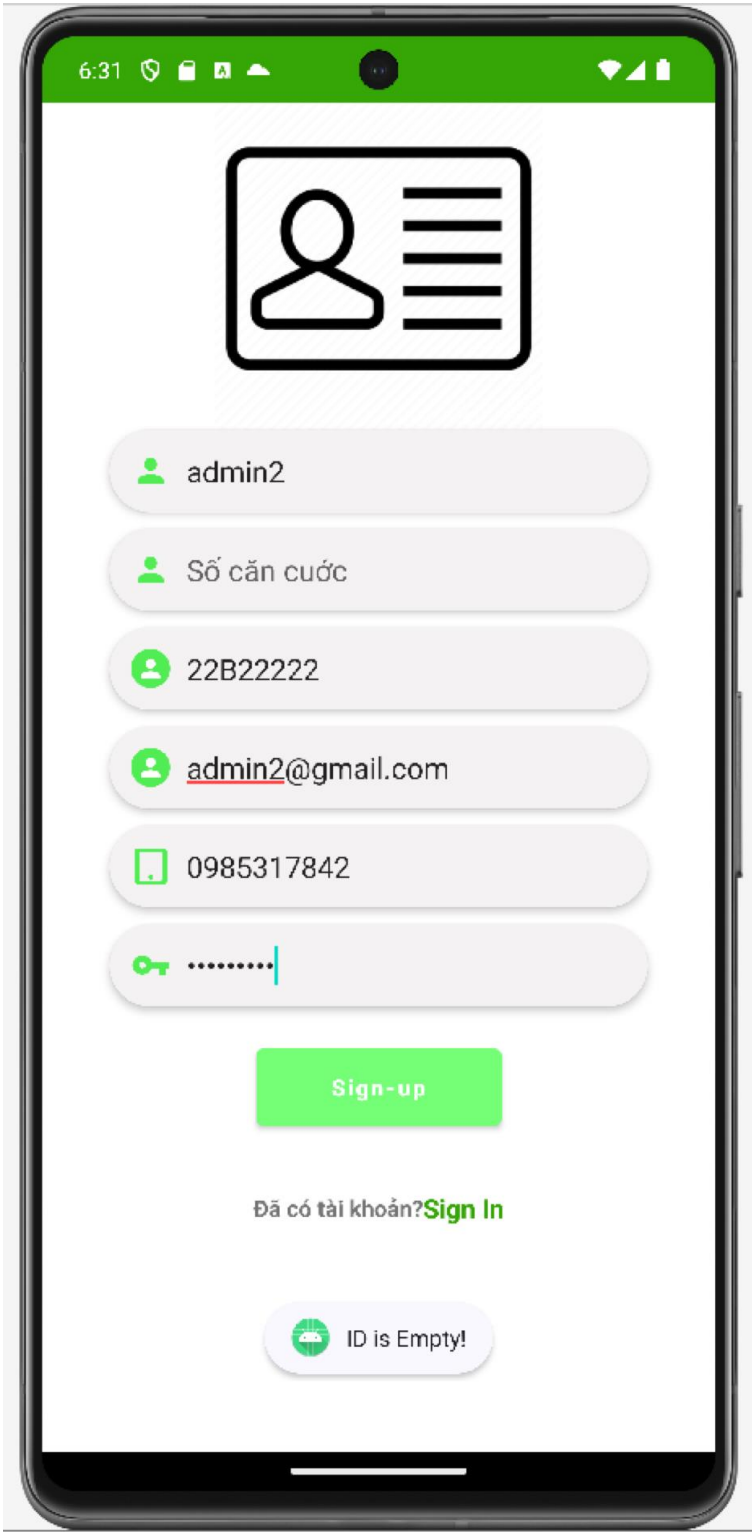


Figure IV.2.9 Application Sign Up Interface

ID missing - notification: "ID is Empty!"

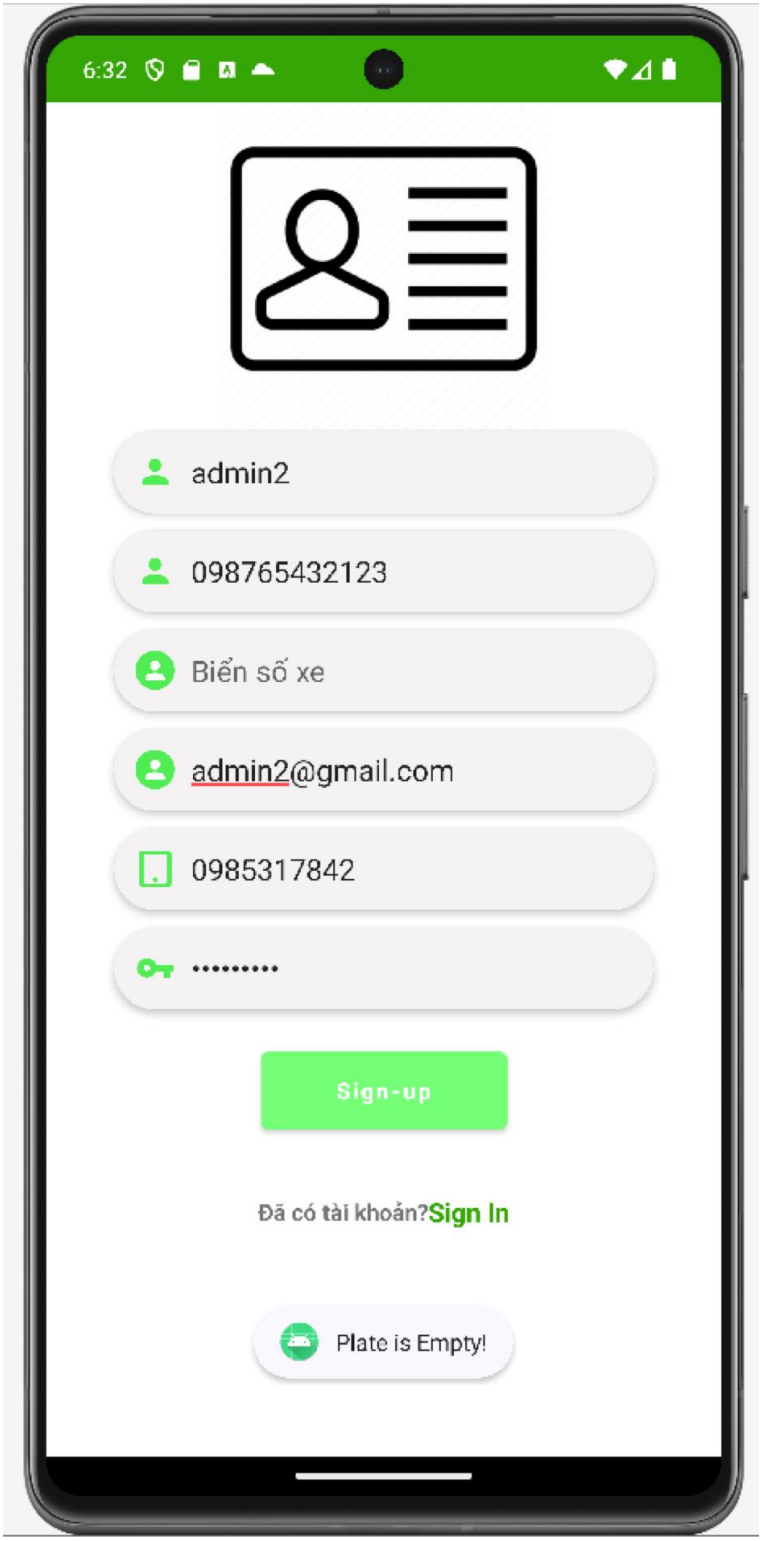


Figure IV.2.10: Application Sign Up Interface

Plate missing - notification: "Plate is Empty!"

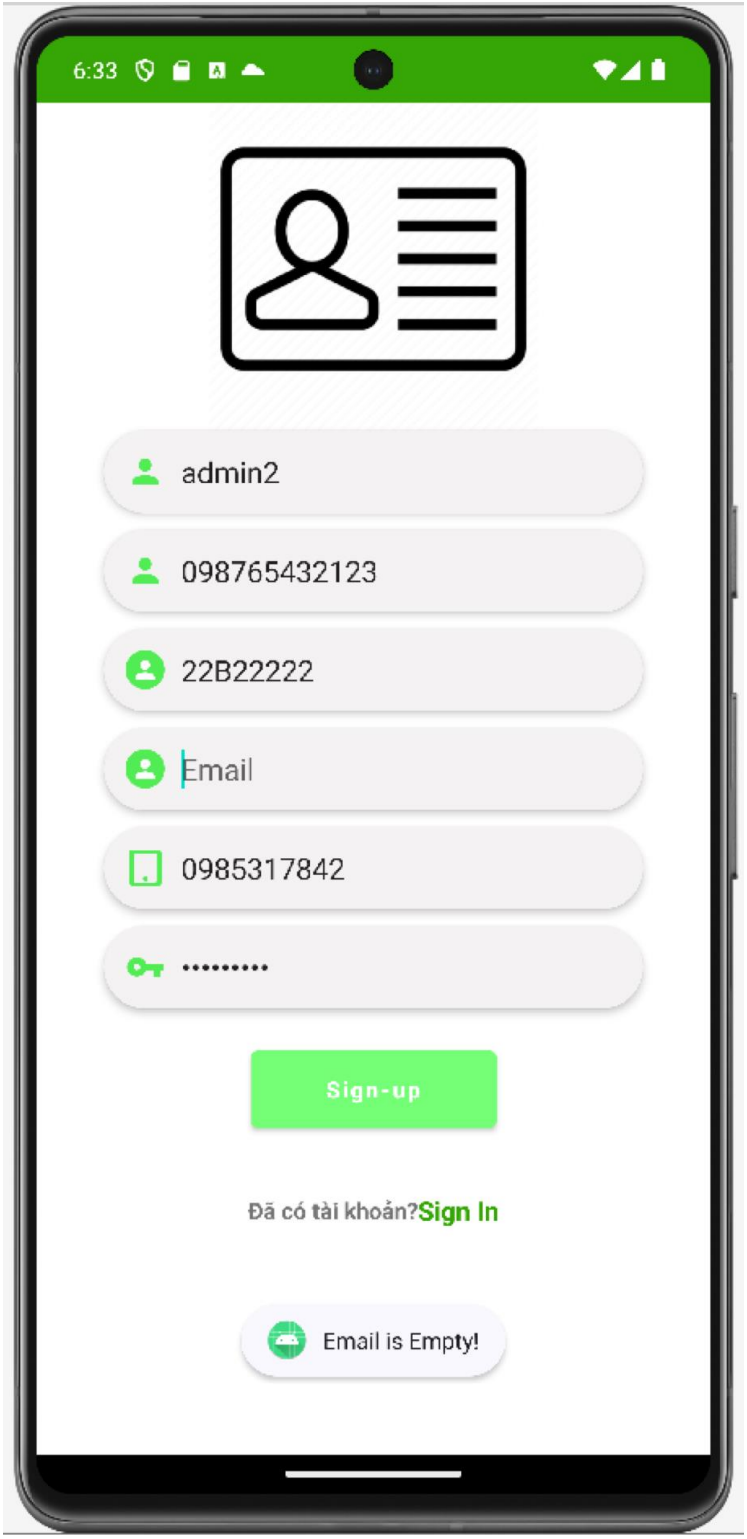


Figure IV.2.11: Application Sign Up Interface

Email missing - notification: "Email is Empty!”

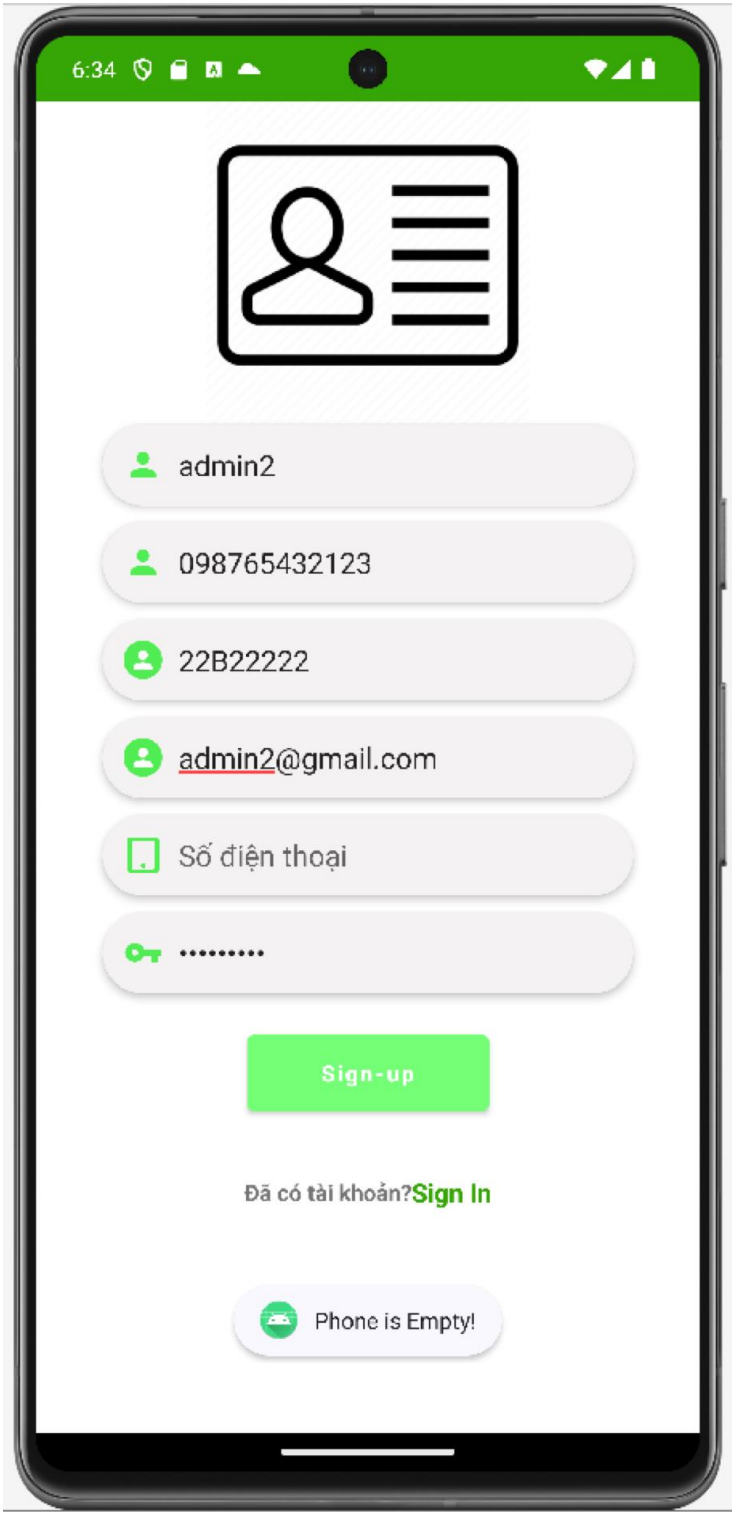


Figure IV.2.12: Application Sign Up Interface

Phone Number missing – notification “Phone is Empty”

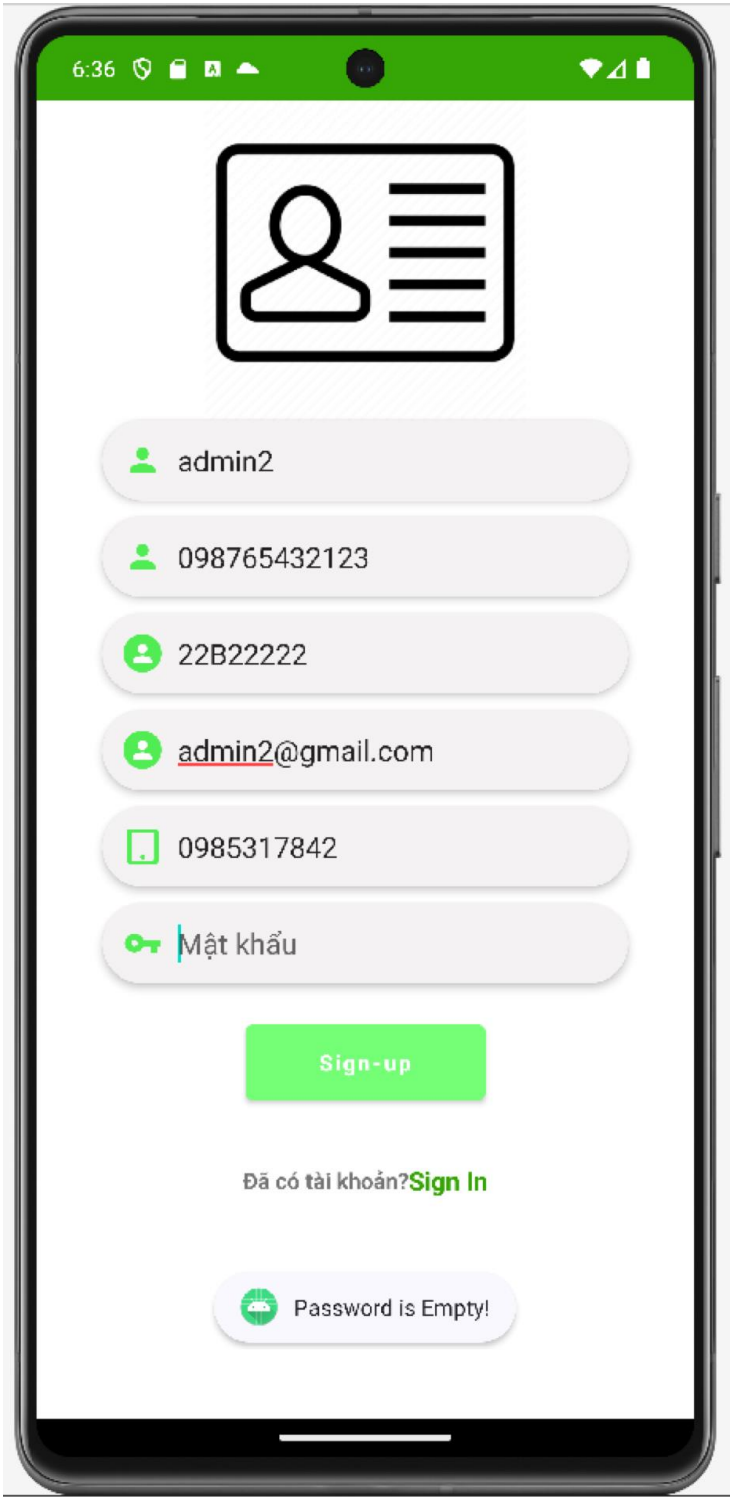


Figure IV.2.13: Application Sign Up Interface

Password missing - notification: "Password is Empty”

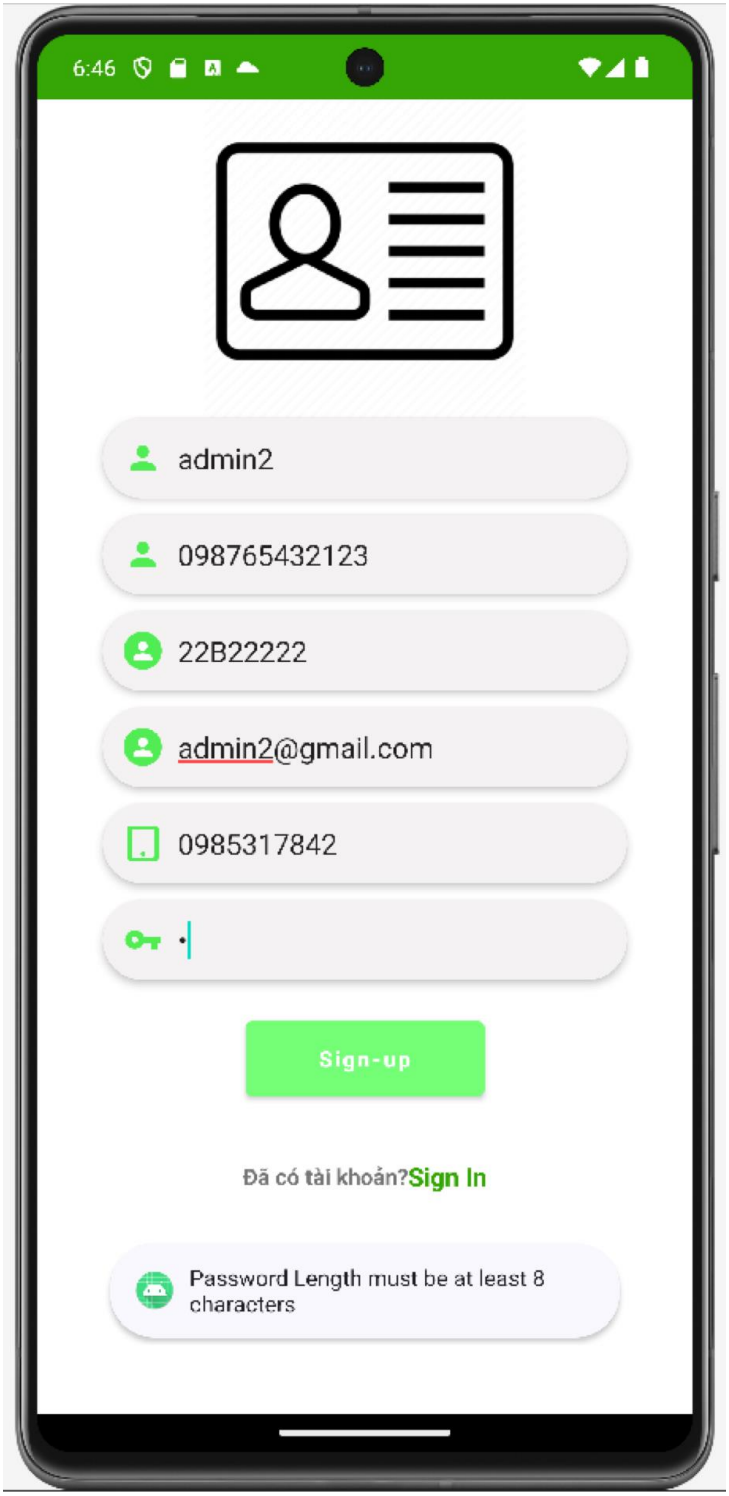


Figure IV.2.14: Application Sign Up Interface

Password under 8 characters: notification: “Password Length must be at least 8 characters"

V. System Set Up Guide

1. Set Up Protocol

MQTT stands for Message Queuing Telemetry Transport. MQTT is a simple messaging protocol, designed for constrained devices with low bandwidth. So, it's the perfect solution to exchange data between multiple IoT devices.

MQTT communication works as a publish and subscribe system. Devices publish messages on a specific topic. All devices that are subscribed to that topic receive the message.

The MQTT broker is responsible for receiving all messages, filtering the messages, deciding who is interested in them, and then publishing the message to all subscribed clients.

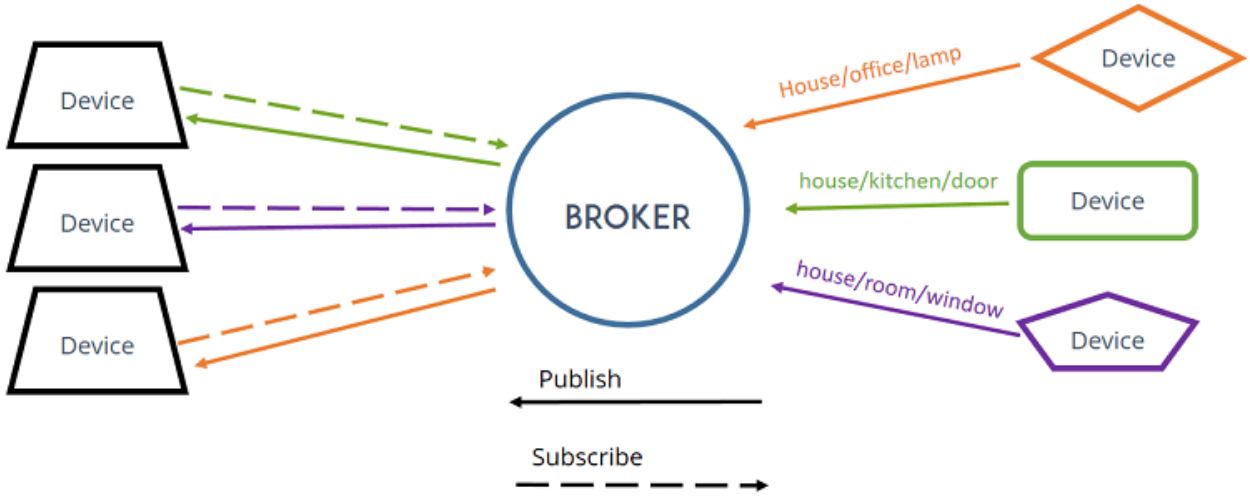


Fig V.1.1: MQTT Protocol

Installing MQTT

First of all, we need a machine which can run the broker (also called MQTT Server). Raspberry Pi is a good choice to perform this task. It consumes low power and can run for days/months without any performance degradation. I am going to use "Mosquitto" as MQTT broker. It can be installed easily on Raspberry Pi OS or any other Linux machine. Use the following commands to install Mosquitto:

1) Open a new Raspberry Pi terminal window.

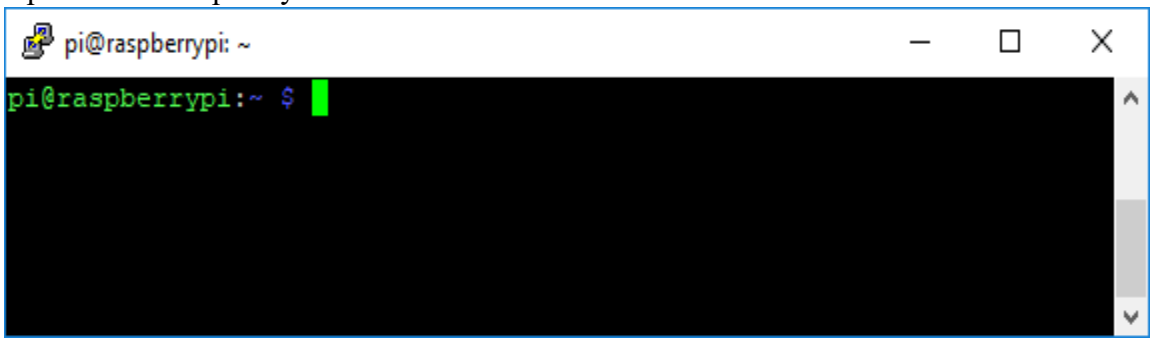


Figure V.1.2: Raspberry Pi terminal window

2) Run the following command to upgrade and update your system:

   sudo apt update && sudo apt upgrade –y

3) To install the Mosquitto Broker enter these next commands:

   sudo apt install -y mosquito

   sudo apt install -y mosquitto-clients

4) To make Mosquitto auto start when the Raspberry Pi boots, you need to run the following command (this means that the Mosquitto broker will automatically start when the Raspberry Pi starts):

   sudo systemctl enable mosquitto.service

Mosquitto Broker Enable Remote Access (Authentication: user and password)

1) Run the following command, but replace YOUR_USERNAME with the username you want to use:

   sudo mosquitto_passwd -c /etc/mosquitto/passwd YOUR_USERNAME

When you run the preceding command with the desired username, you'll be asked to enter a password. No characters will be displayed while you enter the password. Enter the password and memorize the user/pass combination, you'll need it later in your projects to make a connection with the broker.

2) Run the following command to edit the configuration file:

   sudo nano /etc/mosquitto/mosquitto.conf

3) Add the following three lines to allow connection for authenticated users and tell Mosquitto where the username/password file is located.

   allow_anonymous false

   listener 1883

   password_file /etc/mosquitto/passwd

Your configuration file will look as follows:

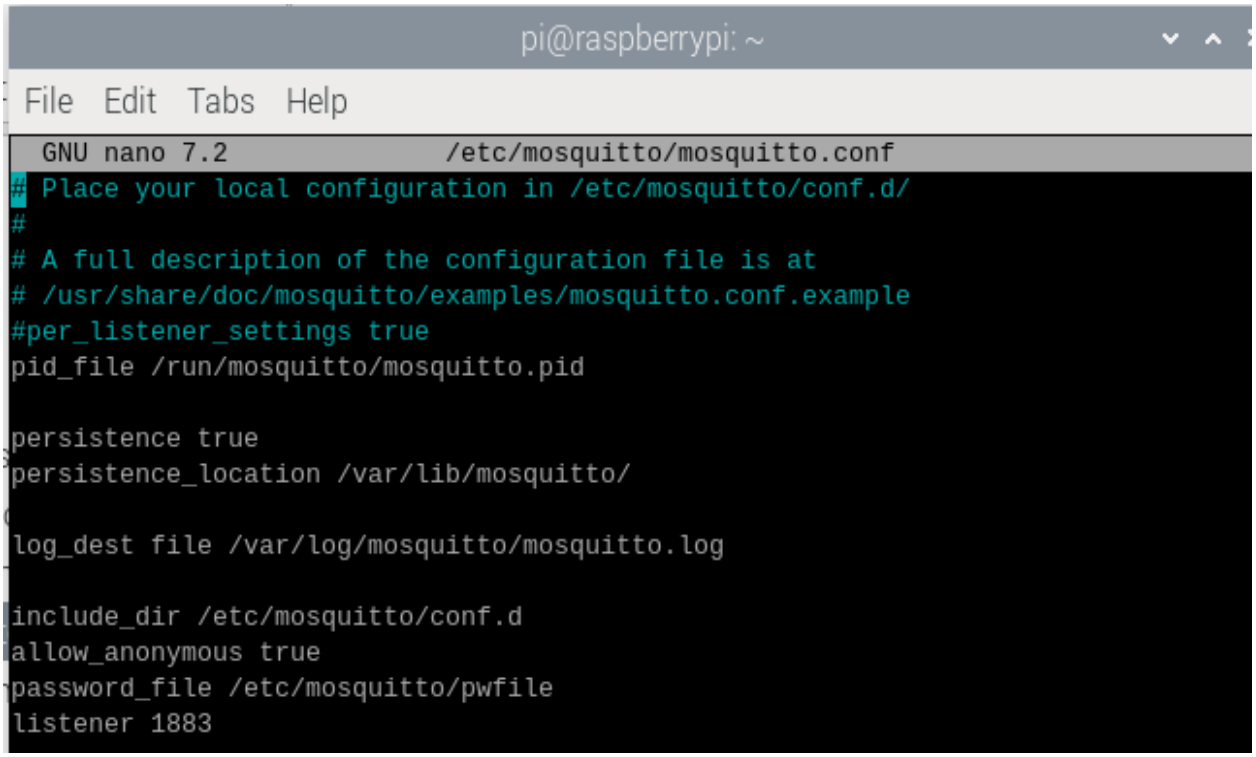


Figure V.1.3: Terminal window

4) Press CTRL-X, then Y, and finally press Enter to exit and save the changes.
5) Restart Mosquitto for the changes to take effect.

sudo systemctl restart mosquitto

To check if Mosquitto is actually running, you can run the following command:

sudo systemctl status mosquitto

Now, you have authentication with username and password enabled.

6) In order to write Python scripts that can connect to the broker as clients, you need to install paho-mqtt.

sudo pip3 install paho-mqtt

After that in the scripts you can import it:

import paho.mqtt.client as mqtt

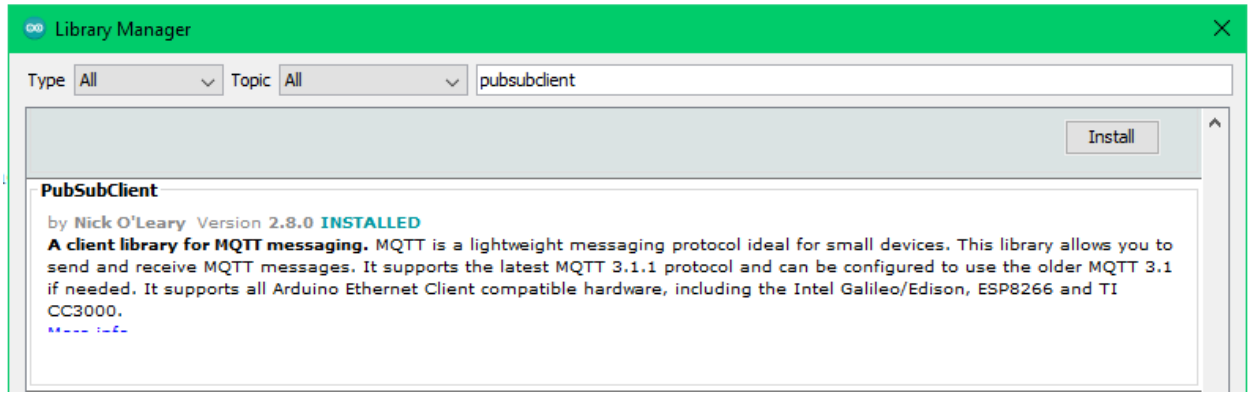


Figure V.1.4: PubSubClient on Arduino IDE Library Manager

For ESP32 CAM on Arduino IDE. You need to install a pubsubclient library to compile this code. The library contains utility functions to perform task such as connecting to a MQTT server, publish /subscribe to topics.

Just declare and write like bellow scripts to be able to use MQTT

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "yourSSID";
const char* password = "yourPassword";

const char* mqtt server = "your MOTT Broker IP address";
const char* mqttUser = "**********";
const char* mqttPassword = "**********";

WiFiClient espClient;
PubSubClient client (mqtt_server, 1883, espClient);
void setup () {
//put your setup code here, to run once:
client.connect (clientId.c_str (), mqttUser, mqttPassword)
}

void loop () {
// put your main code here, to run repeatedly:

}
```

2. Setup Raspberry Pi LAMP server

Figure V.2.1: LAMP Server

Install Apache2 on Raspberry Pi

Apache2 is the most widely used web server software. Briefly, a web server is the software that handles requests to access a web page. Then, depending on the page you have requested, the server will generate the document to serve you (.html, .php, etc.).

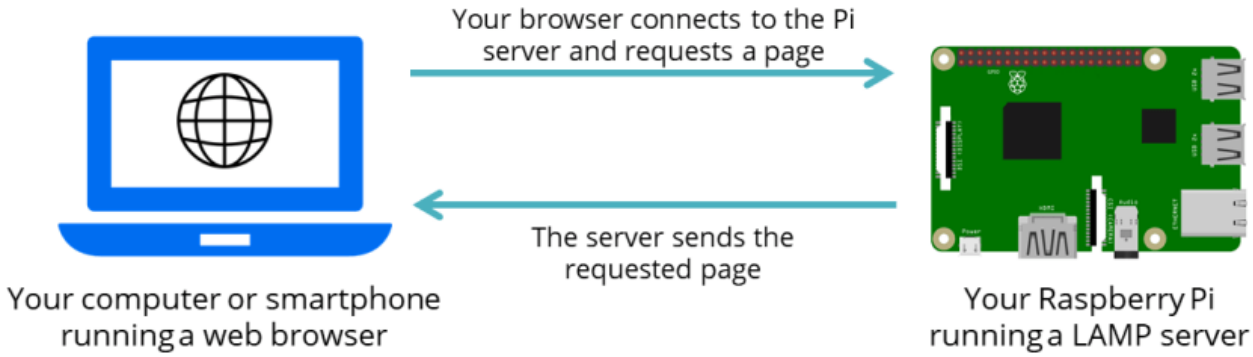


Figure V.2.2 Image Shows How Apache2 Work

To install Apache2 on your Raspberry Pi, run the next command:

pi@raspberrypi:~ $ sudo apt install apache2 –y

Install PHP on Raspberry Pi

PHP is a server side scripting language. PHP (Hypertext Preprocessor) is used to develop dynamic web applications. A PHP file contains <?php … ?> tags and ends with the extension "*.php*".

To install PHP on Raspberry Pi, run:

pi@raspberrypi:/var/www/html $ sudo apt install php –y

Finally, restart Apache2:

pi@raspberrypi:/var/www/html $ sudo service apache2 restart

Remove index.php file from the /var/www/html directory:

pi@raspberrypi:/var/www/html $ sudo rm index.php

Install MySQL (MariaDB Server) on Raspberry Pi

MySQL (often pronounced My S–Q–L) is a popular open source relational database.

Install the MySQL Server (MariaDB Server) and PHP-MySQL packages by entering the following command:

pi@raspberrypi:/var/www/html $ sudo apt install mariadb-server php-mysql -y

pi@raspberrypi:/var/www/html $ sudo service apache2 restart

After installing MySQL (MariaDB Server), it's recommend to run this command to secure your MySQL installation:

pi@raspberrypi:/var/www/html $ sudo mysql_secure_installation

This should appear in your Terminal window:

- You will be asked **Enter current password for root** (type a secure password): press Enter
- Type in **Y** and press **Enter** to Set root password
- Type in a password at the New password: prompt, and press Enter. Important: remember this root password, as you will need it later
- Type in **Y** to Remove anonymous users
- Type in **Y** to Disallow root login remotely
- Type in **Y** to Remove test database and access to it
- Type in **Y** to Reload privilege tables now

Install phpMyAdmin on Raspberry Pi

phpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL using a web interface.

To install phpMyAdmin on a Raspberry Pi, type the following command into the terminal:

pi@raspberrypi:/var/www/html $ sudo apt install phpmyadmin –y

PHPMyAdmin installation program will ask you few questions. We'll use the **dbconfig-common.**

Select **Apache2** when prompted and press the **Enter** key

Configuring **phpmyadmin**? **OK**

Configure database for phpmyadmin with **dbconfig-common**? **Yes**

Type your **password** and press **OK**

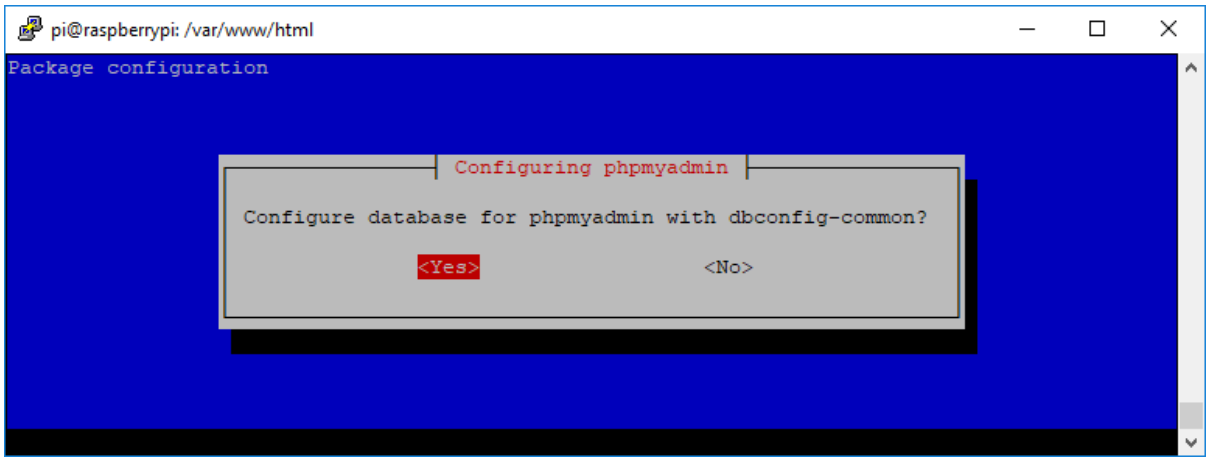


Figure V.2.3: Example Screen

Enable the PHP MySQLi extension and restart Apache2 for changes to take effect:

pi@raspberrypi:/var/www/html $ sudo phpenmod mysqli

pi@raspberrypi:/var/www/html $ sudo service apache2 restart

HTTP POST Request Method

The Hypertext Transfer Protocol (HTTP) works as a request-response protocol between a client and server. Here's an example:

- The ESP32 (client) submits an HTTP request to a Server (for example: local RPi Lamp Server or example.com);
- The server returns a response to the ESP32 (client);

HTTP POST is used to send data to a server to create/update a resource. For example, publish an image to a server.

POST /upload.php HTTP/1.1

Host: example.com

Content-Type: image/jpeg

3. Setup opencv for Raspberry Pi

Step 1: Expand the filesystem

Figure V.3.1: Terminal Shell of Raspberry Pi

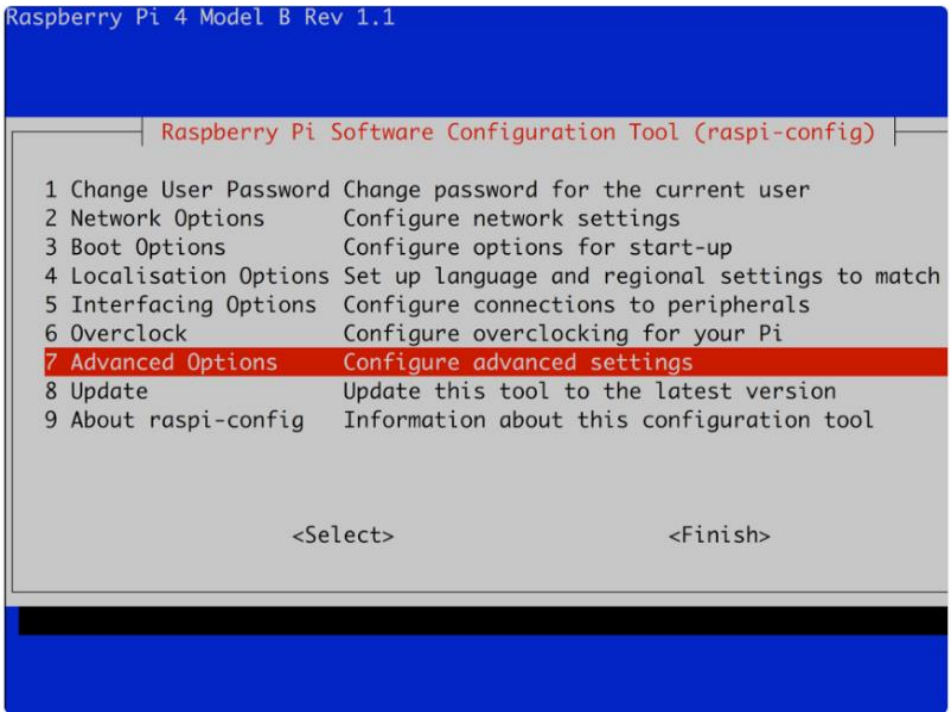


Figure V.3.2: after sudo raspi-config

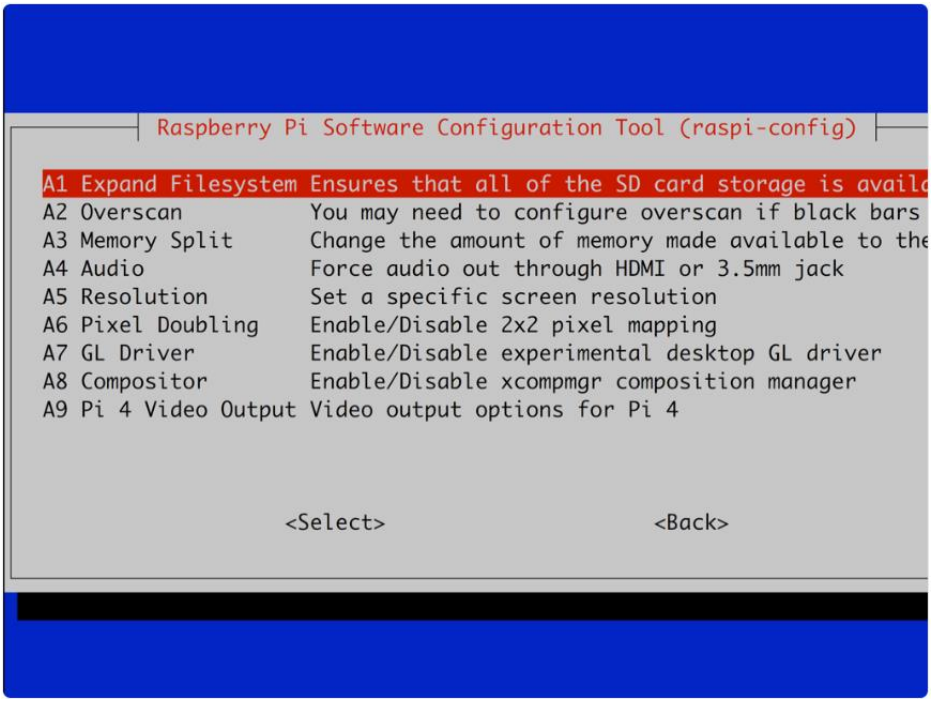


Figure V3.3: Raspberry Pi Config after 7

Type **sudo raspi-config** on Terminal Shell

Navigate to **Advanced Options** and press enter

Navigate to **A1 Expand Filesystem** and press enter

Navigate to the **<Finish>** button and then reboot the Raspberry Pi.

Step 2**:** update and upgrade any existing packages with:\

sudo apt-get update && sudo apt-get full-upgrade -y

Step 3: Install dependencies

sudo apt-get install build-essential cmake pkg-config libjpeg-dev libtiff5-dev libjasper-dev libpng-dev -y

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libxvidcore-dev libx264-dev libfontconfig1-dev -y

sudo apt-get install libcairo2-dev libgdk-pixbuf2.0-dev libpango1.0-dev libgtk2.0-dev libgtk-3-dev libatlas-base-dev gfortran libhdf5-dev libhdf5-serial-dev libhdf5-103 python3-pyqt5 python3-dev -y

sudo apt-get install libv4l-dev -y

Step 4: Install numpy

Check python version. Python –version
Install corresponding numpy

1. **Python 3.6 works with numpy 1.19 and shap 0.39.**
2. **Python 3.9 fails with numpy 1.19 and shap 0.39**
3. **Python 3.9 works with numpy 1.20 and shap 0.39**
4. **Python 3.9 works with numpy 1.19 and shap 0.38**

Next, uninstall the default numpy version if available

pip uninstall numpy

then install the numpy version corresponding to your python version

pip install numpy == <version>

Step 5: Install Pip (Package Management Tool)

sudo apt-get install python3-pip

Step 6. Install OpenCV

Pip install opencv-contrib-python

VII. References

[1] Anonymous. *Grayscaling of Images using OpenCV*. (2019, April 15). GeeksforGeeks. https://www.geeksforgeeks.org/python-grayscaling-of-images-using-opencv/

[2] Adrian Rosebrock. *OpenCV Morphological Operations*. (2021, April 28). https://pyimagesearch.com/2021/04/28/opencv-morphological-operations/

[3] Anonymous. *Morphological Operations*. (2022, March 16). GeeksforGeeks.

https://www.geeksforgeeks.org/python-opencv-morphological-operations/

[4] GMO-Z.com Vietnam Lab Center Technology Blog| https://blog.vietnamlab.vn/xu-ly-anh-voi-opencv-tut-2-chuyen-doi-anh-mau/

[5] Adrian Rosebrock | *OpenCV Edge Detection (cv2.Canny)*| (2021, May 12)| PyImageSearch| https://pyimagesearch.com/2021/05/12/opencv-edge-detection-cv2-canny/

[6] VinBigData | Canny – Phát hiện cạnh trong OpenCV: Hướng dẫn chi tiết từng bước |(2022, November 02)| https://vinbigdata.com/kham-pha/canny-phat-hien-canh-trong-opencv-huong-dan-chi-tiet-tung-buoc.html

[7] *OpenCV: Contours : Getting Started*. (2019). Opencv.org. https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html

[8] Pandey, D. (2021, April 19). *Contours and Convex Hull in OpenCV Python*. Analytics Vidhya. https://medium.com/analytics-vidhya/contours-and-convex-hull-in-opencv-python-d7503f6651bc

[9] *OpenCV: Contour Features*. (n.d.). Docs.opencv.org. https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html

[10] CodeLearn. (n.d.). *Thuật Toán K-Nearest Neighbors (KNN) Siêu Cơ Bản*. Codelearn.io. https://codelearn.io/sharing/thuat-toan-k-nearest-neighbors-knn

[11] VoHungVi. (2017, May 20). Nhận diện ký tự bằng KNN. Thigiacmaytinh.com. https://thigiacmaytinh.com/nhan-dien-ky-tu-bang-knn/

[12] Nguyen Thi Hop. *KNN (K-Nearest Neighbors) #1*. (2019, July 16). Viblo.asia. https://viblo.asia/p/knn-k-nearest-neighbors-1-djeZ14ejKWz