**Final Project | Tan-Awa: A Vehicle Surveillance System (VSS) using ESP32-Cam Module**

CpE Elec 1 - Systems and Network Administration 1 / Embedded System 1

February 12, 2025

**Matthew Arni Bendo**

**Loreen Wilmer J. Yboa**

BSCpE 3A

**Engr. Maricris M. Ediza**

Instructor

## **Introduction**

In the fast-changing security-focused landscape of today, an efficient and automated vehicle monitoring system is crucial for controlling entry to restricted areas. Conventional manual security checkpoints frequently experience inefficiencies, human mistakes, and delays, rendering them unreliable in high-security settings. With the progress of embedded systems and IoT-based solutions, the integration of automated surveillance technology has emerged as a practical choice to enhance security and operational effectiveness.

This Vehicle Surveillance System (VSS) aims to address these issues by combining advanced surveillance technology with automated access management. The system uses ESP32-CAM modules for live image capture, ensuring fast and accurate vehicle identification. Using Google Spreadsheet for data processing and a mobile app for remote monitoring and control, the system provides a smooth and secure way to regulate vehicle access.

The importance of this project comes from its capacity to enhance security through the automation of vehicle entry. The system photographs license plates and compares them to a database, allowing access solely to authorized vehicles, thus lowering the likelihood of unauthorized entry. This is particularly beneficial for gated residential areas, office buildings, parking garages, and numerous other secure facilities. Furthermore, the integration of cloud-based data management guarantees the system's availability, real-time updates, and scalability.

Furthermore, the initiative emphasizes the significance of smart surveillance systems in contemporary security structures. The combination of computer vision, cloud computing, and IoT technologies can lead to security solutions that are more efficient and responsive compared to conventional approaches. The systems initiative offers a specific illustration where vehicle access management is improved through embedded systems and remote oversight, contributing to a more secure and smarter security structure.

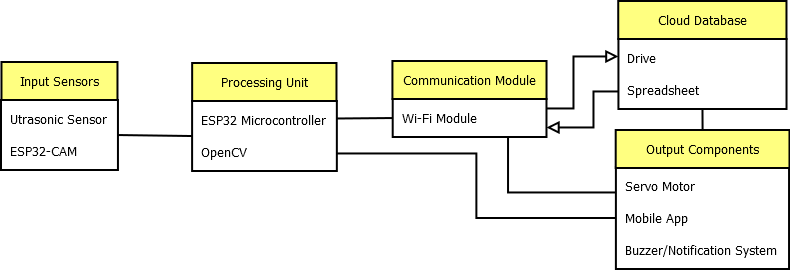
This project seeks to establish a vehicle surveillance system that delivers an automated and dependable security solution catering to the increasing demand for intelligent and interconnected security systems in modern urban and industrial settings.

## **Objectives**

1. **Wi-Fi Module Integration:** This module enables wireless connectivity for real-time data transmission between the ESP32-CAM, Google Spreadsheet, and the mobile application.
2. **Database System (Spreadsheet):** Use Spreadsheet to store and retrieve vehicle data, logs, and user authentication records.
3. **Mobile Application:** Provide a user-friendly mobile app to monitor live video feeds, receive event notifications, and control gate access, enhancing usability and accessibility.

## **Overall Design with Detailed Description of Each Components**

### **Block Diagram**



***Figure 1. UML Diagram***

The Vehicle Surveillance System (VSS) consists of a combination of multiple hardware and software components that work together to provide real-time vehicle monitoring and access control. The system follows a continuous process that starts with vehicle detection and ends with automatic gate operation. The following block diagram represents the main components in the system and their interactions.

#### **Input Sensors**

The procedure starts with an ultrasonic sensor constantly scanning the entrance for incoming vehicles. Once a vehicle enters the detection zone, the sensor sends a trigger signal to the ESP32-CAM module, which starts the image capture process.

#### **Processing Unit**

The ESP32-CAM module takes a photo of the vehicle. The captured image goes through pre-processing methods such as noise reduction and edge detection to improve clarity.

The processed image is passed to the OpenCV algorithm for optical character recognition (OCR). OpenCV library extracts and analyzes license plate characters from the captured image. OCR technology identifies alphanumeric characters and generates a text output representing the vehicle license plate number.

#### **Communication Module**

It enables real-time data transfer between ESP32-CAM, Google Spreadsheet, and mobile applications.

#### **Cloud Database**

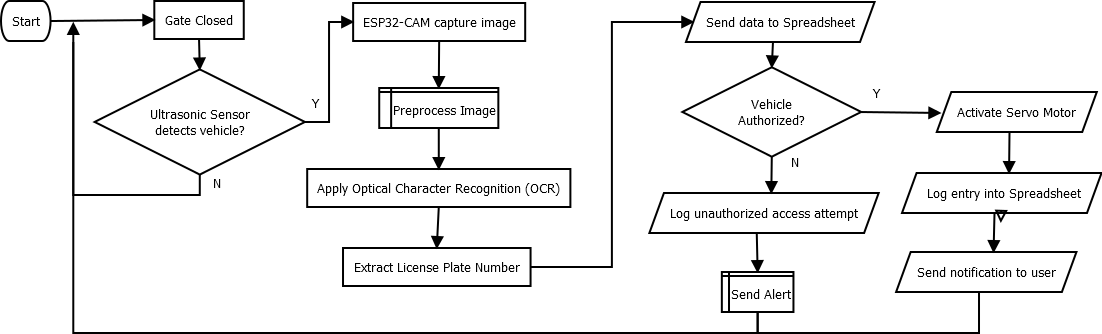
The extracted license plate number is sent to Google Spreadsheet, a cloud-based database, via the Wi-Fi module for validation. Google Spreadsheet compares the received license plate data with its stored list of authorized vehicles. A response signal (access granted or denied) is sent back to the ESP32-CAM.

#### **Output Components**

If the vehicle is authorized, the servo motor is activated to open the gate. If not authorized, the gate will remain closed and a security alert may be triggered by a buzzer and a mobile notification.

The mobile app (developed on Blynk) allows users to remotely monitor the system, view live video feeds, and manually control gate access. When an unauthorized vehicle attempts to enter, the system sends real-time notifications. Users can access vehicle logs stored in Google Spreadsheet to review previous entries.  
  
 This structured block diagram ensures efficient multitasking, real-time monitoring, and safe vehicle access, providing an advanced security solution for restricted areas.

### **System Flowchart**



***Figure 2.1. VSS Flowchart***

Vehicle Surveillance System (VSS) follows a structured process flow to ensure efficient and automated vehicle surveillance. The system integrates multiple components such as ESP32-CAM, Ultrasonic Sensors, Google Spreadsheet Database, and Mobile Applications, all working together to detect, capture, process, and regulate vehicle access. Below is the step-by-step process of the system’s flowchart explained below:

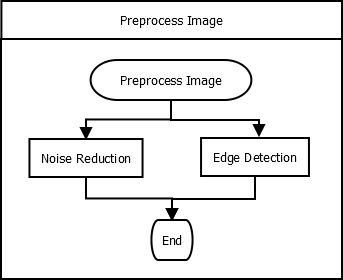
#### **System Initialization**

Once powered on, VSS will initialize all components, including ESP32-CAM, ultrasonic sensor, servo motor, and Wi-Fi module. The system will connect to Google Spreadsheet to ensure real-time data synchronization and remote access.

#### **Vehicle Detection**

The ultrasonic sensor (HC-SR04) continuously scans the entry point for approaching vehicles. This process operates in low power mode to optimize energy efficiency.

When a vehicle is detected within a predefined range, the sensor triggers the ESP32-CAM to capture an image of the vehicle. If no vehicle is detected, the system loops back and remains in scanning mode.



***Figure 2.2. VSS Preprocess Image Function***

#### **Image Capturing and Preprocessing**

Once triggered, the ESP32-CAM module captures an image of the vehicle. The system then applies pre-processing techniques such as:

* Noise reduction removes unwanted visual distortions.
* Edge detection enhances the vehicle's license plate.
* Grayscale conversion enhances contrast to improve OCR accuracy.

#### **License Plate Recognition (OCR Processing)**

The analyzed images are examined using OpenCV to detect the alphanumeric characters on the license plate. The Optical Character Recognition (OCR) system detects and retrieves the license plate and transforms it into textual data.

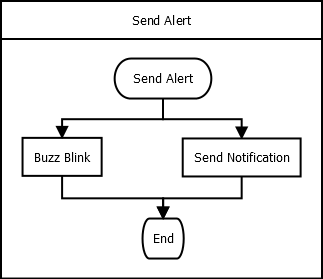
#### **Data Transmission to Google Drive**

Once the license plate number is extracted, the ESP32-CAM transmits the data to Google Drive.

#### **Authorization Check**

The license plate number is compared to determine access permissions.

If a match is found, the vehicle is approved and the system proceeds to the gate operation. If no match is found, access is denied and a warning notification is sent to the user via the mobile application.



***Figure 2.3. VSS Send Alert Function***

#### **Automated Gate Control**

Based on the authorization result mentioned above, the following occurs:

* ***Authorized vehicle:*** The servo motor activates and the gate opens to allow the vehicle to enter. A notification is sent to the user confirming access.
* ***Unauthorized vehicle:*** The gate remains closed and a security alert is triggered to notify the user of possible unauthorized access.

#### **Data Logging in Google Spreadsheet**

Regardless of whether access is approved or denied, the entry attempt is recorded in Google Spreadsheet and includes the following information:

* Timestamp of the event.
* Captured vehicle image.
* License plate details.

This data is accessible through a mobile application, allowing users to monitor past vehicle entry records.

#### **Mobile Application Monitoring & User Control**

The mobile application allows users to remotely:

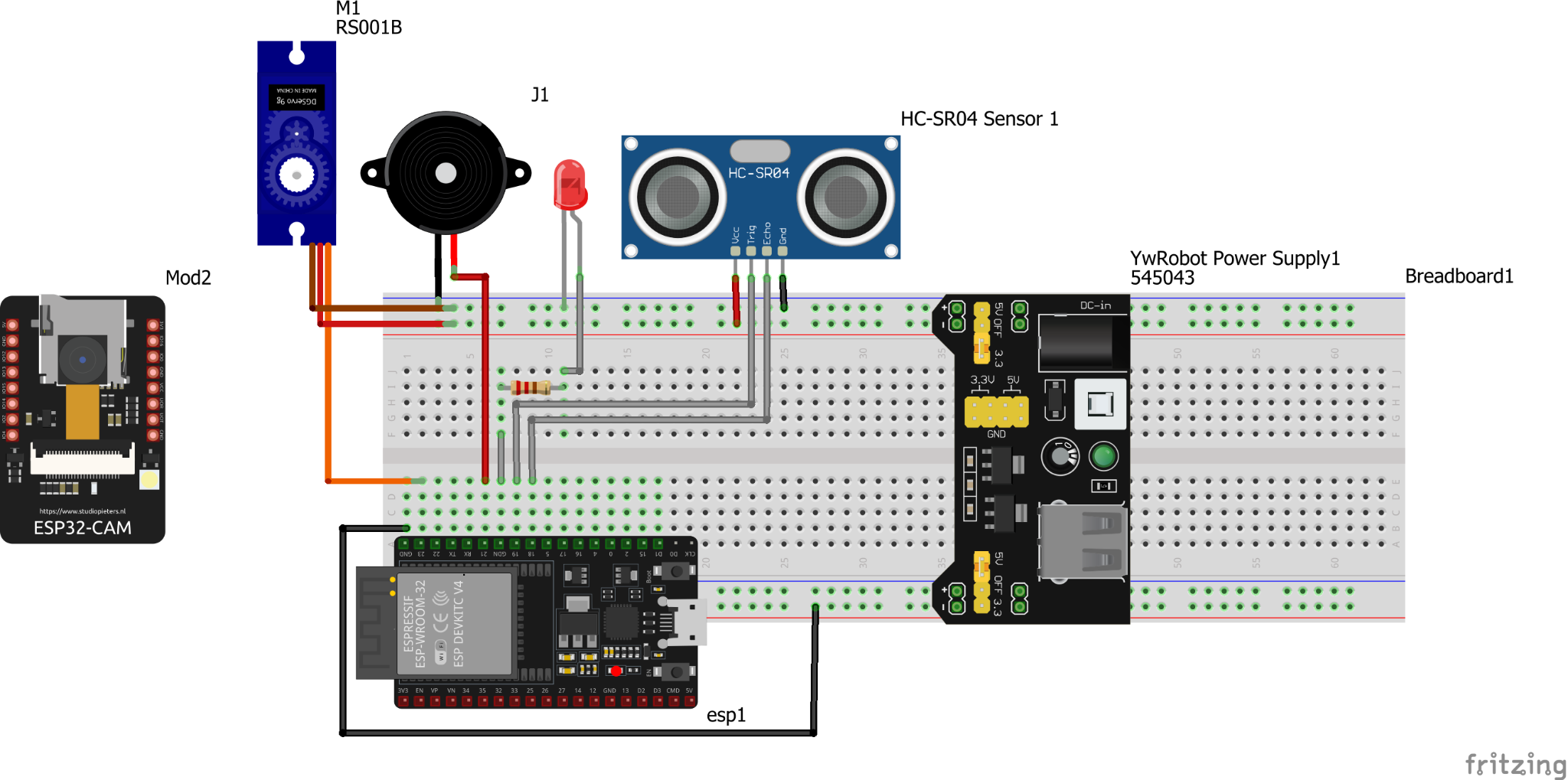
* Receive real-time notifications about detected vehicles.
* Manually override the system to allow or deny access.

If an anomaly is detected, users can take appropriate action, such as blocking unauthorized vehicles or reporting a security concern.

#### **System Continuous Monitoring**

After processing each vehicle entry attempt, the system returns to vehicle detection mode and monitors for the next entering vehicle. This cycle repeats to ensure continuous security monitoring.

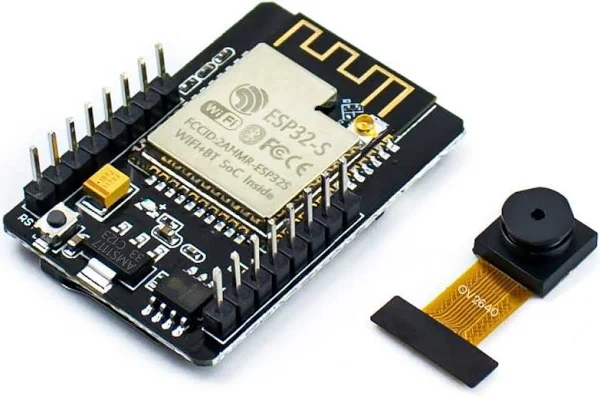
### **System Circuit Design**



***Figure 3. VSS Circuit Design***

Developing a Vehicle Surveillance System (VSS) requires a combination of hardware components, software tools, and cloud services to ensure smooth operation and efficient monitoring. The essential ingredients are:

#### **Hardware Requirements**



***Figure 3.1.1. ESP32-CAM Module***

To capture and process vehicle images, the Vehicle Surveillance System (VSS) uses an ESP32-CAM module with an integrated Wi-Fi module for real-time data transfer. The camera captures images of approaching vehicles and processes them using computer vision algorithms to extract license plate numbers. These images are then processed with pre-processing techniques such as noise reduction and edge detection to enhance clarity and improve Optical Character Recognition (OCR) accuracy. Processed data is sent to a cloud-based database for matching with stored records, enabling automated access control and efficient vehicle monitoring.

Notably, the ESP32-CAM module is connected separately, rather than directly to the breadboard like the other components. This module is used for image and video capture, providing visual feedback to the system. This module operates independently and may communicate wirelessly with the ESP32 over Wi-Fi or another data connection.



***Figure 3.1.2. Ultrasonic Sensor (HC-SR04)***

To detect the presence of a vehicle approaching the gate, the Vehicle Surveillance System (VSS) employs an ultrasonic sensor that continuously monitors the entrance. When a vehicle is detected within a predefined range, the sensor triggers the ESP32-CAM module to capture an image of the vehicle. This detection mechanism allows the system to be active only when necessary, saving power and optimizing processing efficiency. Additionally, the sensor data can be used to regulate gate operations to ensure timely access control and prevent unauthorized or accidental entry.



***Figure 3.1.3. Servo Motor***

To control the gate mechanism, the Vehicle Surveillance System (VSS) uses servo motors or relay-controlled actuators to open and close the gate based on the vehicle's authorization. Once the vehicle's license plate is captured and matched against a database, the system sends a signal to the motor, telling it to rotate and allow or deny access. If the vehicle is authorized, the motor activates to open the gate, allowing entry; if not, the gate remains closed and a warning may be triggered by a buzzer or notification system. This automated mechanism ensures efficient, hands-free access control and enhances security by preventing unauthorized entry.



***Figure 3.1.4. Buzzer***

A buzzer is also integrated and connected to the ESP32, which is used for warning and notification purposes.



***Figure 3.1.6. Breadboard Power Supply Module***

The power module provides regulated power to the circuit via the breadboard, ensuring stable voltage distribution to the components. The ESP32 board is powered via the breadboard connections and acts as the central processing unit of the system.



***Figure 3.1.7. Power Supply: Lithium Battery Pack***

A lithium battery pack is used to power the system components, providing a reliable and energy-efficient power source for continuous operation. The lithium battery pack has high energy density and is rechargeable, which can extend the operating life of the system while maintaining a stable voltage level.



***Figure 3.1.8. Wi-Fi Module in ESP32***

The integrated Wi-Fi module built into the ESP32 enables seamless connectivity to Google Spreadsheet, enabling real-time data transfer between the system's hardware and cloud-based storage. This wireless communication allows ESP32-CAM to upload captured vehicle images, submit license plate data for verification, and receive approval responses from the database. By leveraging Google Spreadsheet's cloud infrastructure, the system ensures efficient remote access, data synchronization, and secure storage of vehicle logs. The connectivity also supports mobile application integration, allowing users to monitor and control the system from anywhere with an internet connection.

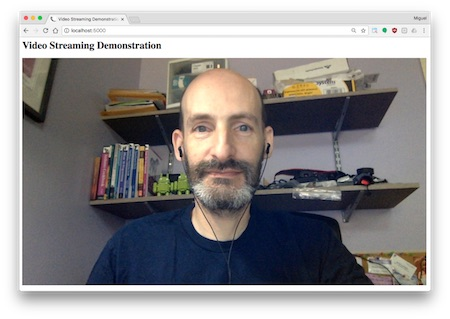
Overall, this circuit design demonstrates an integrated approach to sensor-based control, motor actuation, and vision monitoring using the ESP32 ecosystem.

#### **Software Requirements**



***Figure 3.2.1. Blynk***

Blynk Web Application is used for mobile applications to enable remote monitoring and control of Vehicle Surveillance Systems (VSS). Through Blynk's intuitive interface, users can receive real-time notifications, view captured license plate images, and manage vehicle access permissions directly from their smartphones. The application connects to ESP32 via Wi-Fi and Google Spreadsheet, enabling seamless communication between the hardware and cloud storage. In addition, Blynk provides interactive widgets to control gate mechanisms, view system logs, and customize access settings. This integration provides a smart remote control solution for efficient and safe vehicle monitoring, improving user convenience.



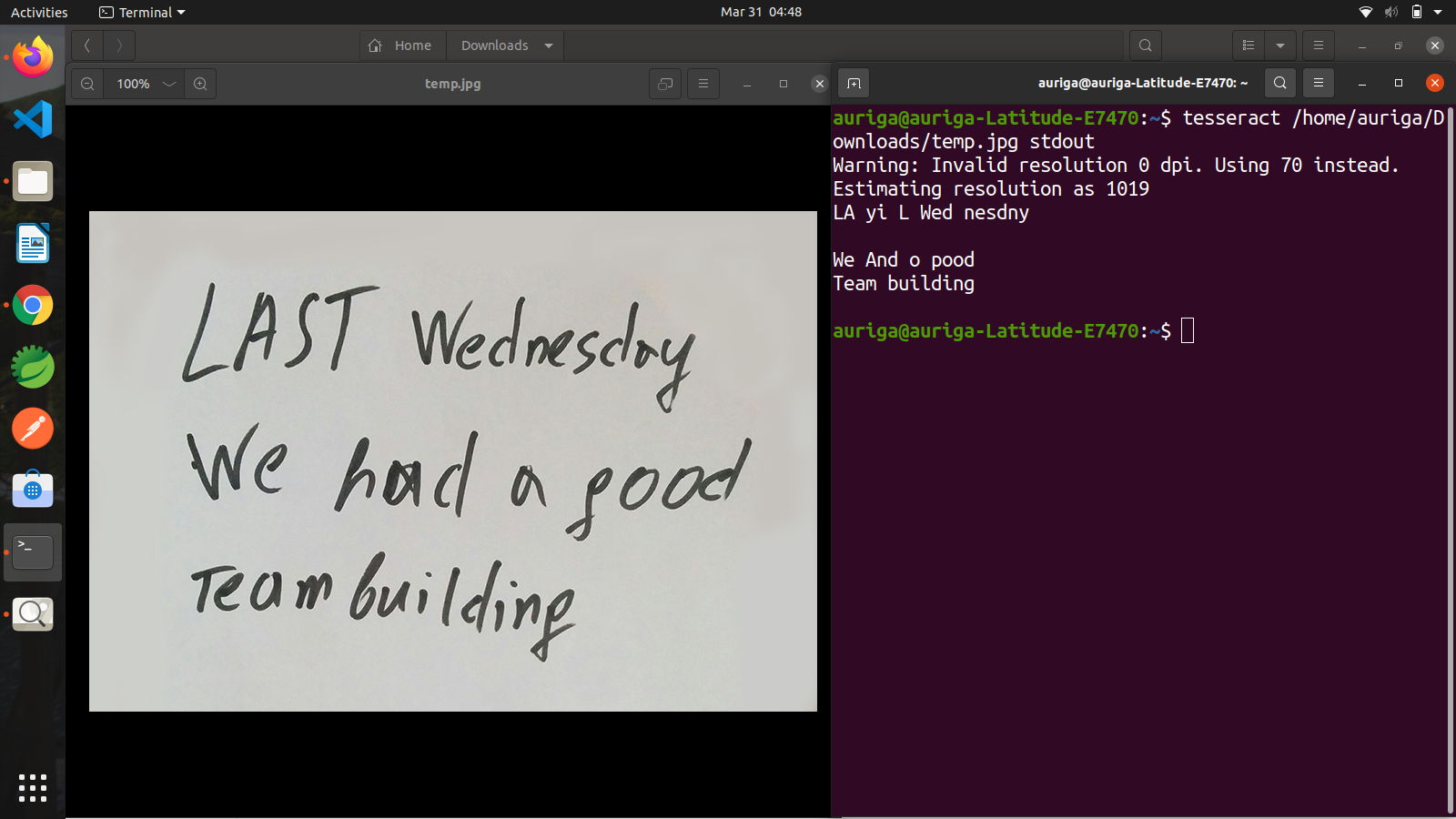
***Figure 3.2.3. Flask***

Integrating Flask with CameraWebServer allows seamless access and management of images captured by ESP32-CAM. Flask, a lightweight Python web framework, is used to access image content from a specified URL and retrieve still images from ESP32-CAM. CameraWebServer hosted on ESP32-CAM streams live video and captures images, which Flask processes and displays within a web interface built using HTML. By obtaining the IPv4 address of the device within the home Wi-Fi network, Flask can continuously capture images with a specific delay or allow the user to manually trigger capture. Captured images are stored locally in a specified directory, with the latest image stored for further processing or analysis. This integration enables real-time remote monitoring and secure image storage, enhancing the capabilities of the surveillance system.



***Figure 3.2.4. OpenCV***

Once the images are processed using OpenCV techniques such as grayscale conversion, noise reduction, and edge detection, the system can further analyze the captured frames to detect and recognize vehicle license plates. Using Optical Character Recognition (OCR), the extracted plate numbers are compared against a pre-registered database. If the detected plate matches an authorized vehicle, the system automatically triggers the Servo Motor to open, enhancing security and convenience in this vehicle surveillance system.



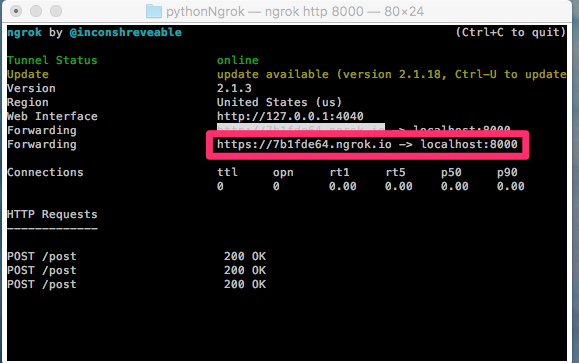
***Figure 3.2.5. pyTesseract***

pyTesseract is used as an Optical Character Recognition (OCR) algorithm to extract alphanumeric text from license plate images captured by ESP32-CAM. Tesseract applies its deep learning-based text recognition capabilities to accurately identify and extract characters from the plate. The extracted license plate number is sent to a Google Spreadsheet database for validation to determine if the vehicle is authorized for access. Tesseract's robust OCR capabilities improve the accuracy and efficiency of Vehicle Surveillance Systems (VSS), ensuring reliable vehicle identification.



***Figure 3.2.6. Google Drive & Sheets***

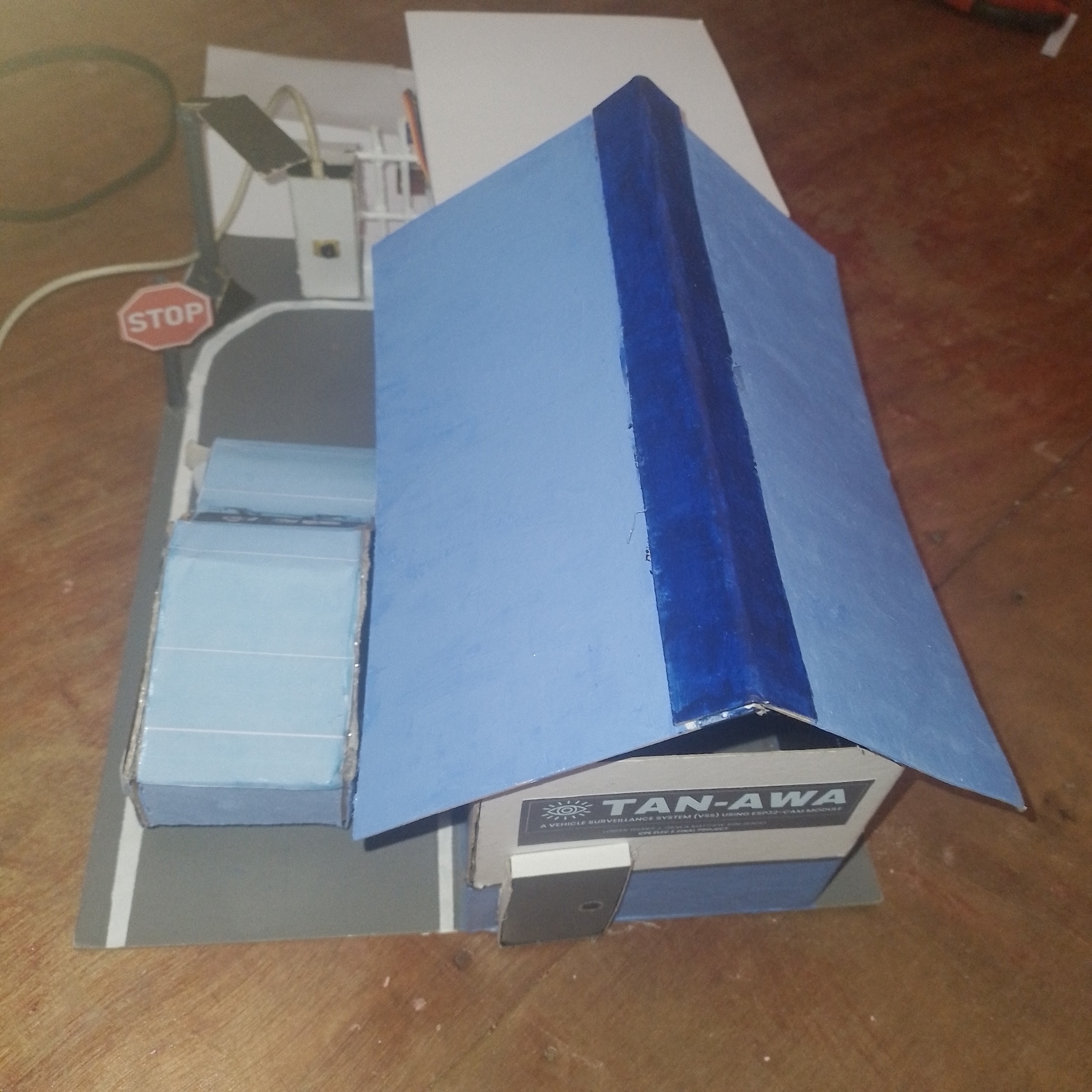
Google Drive serves as the database for storing images captured by the ESP32-CAM, ensuring easy access and retrieval of visual records. Meanwhile, Google Sheets functions as the overall tabular database, maintaining structured records of vehicle details, license plate numbers, timestamps, and the images captured. This integration allows for efficient data management, enabling automated tracking, verification, and monitoring within the system.



***Figure 3.2.7. ngrok | API Gateway***

To facilitate secure remote access and communication between the ESP32-CAM and cloud services, Ngrok is used to expose the local server to the internet, enabling real-time data transmission. Additionally, an API Gateway is implemented to manage and streamline API requests, ensuring secure and efficient interaction between different system components, such as the camera module, database, and automated garage control.

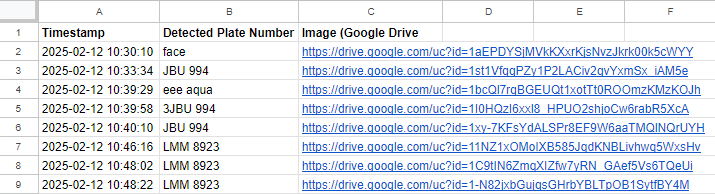
## **Results**

**

***Figure 4.1. Actual Design***

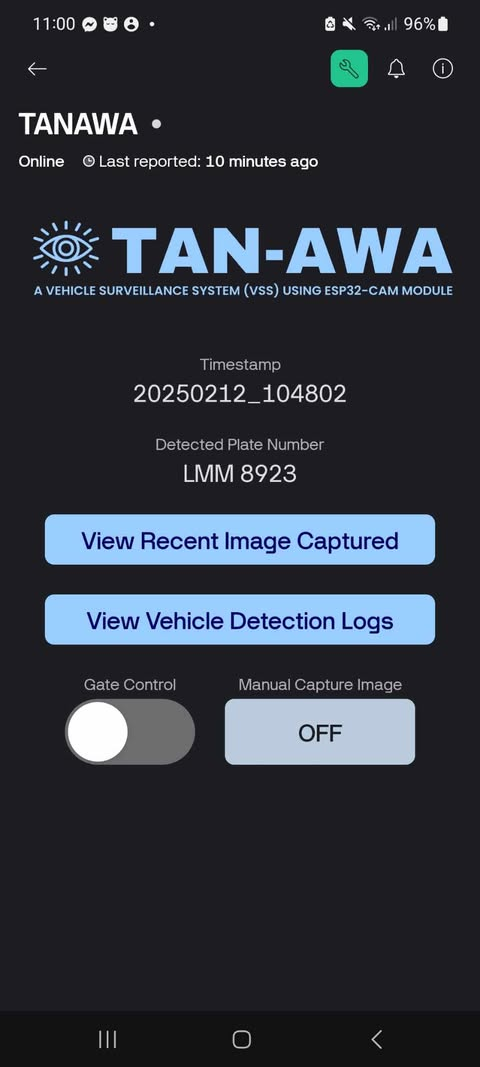
The implementation of a vehicle surveillance system (VSS) demonstrated a highly efficient and automated method for vehicle access monitoring and control. The system successfully integrated ESP32-CAM, Google Spreadsheet, OpenCV, and Tesseract OCR to ensure accurate image capture, real-time data processing, and effective vehicle identification.

In the test phase, the ultrasonic sensor (HC-SR04) reliably detected approaching vehicles and triggered the ESP32-CAM module to capture images. The captured images went through pre-processing techniques such as grayscale conversion, noise reduction, and edge detection to improve the clarity of the license plate before OCR processing. Combined with Tesseract OCR, the OpenCV-based license plate recognition system effectively extracted alphanumeric characters from various license plates and achieved high accuracy in bright conditions. However, challenges arose in low-light environments and when the license plate was distorted, resulting in a slight decrease in recognition efficiency.



***Figure 4.2. Google Spreadsheet database connected with an API***

The integrated Wi-Fi module of the ESP32 allowed the extracted license plate data to be sent in real time to Google Spreadsheet, where it was successfully compared with the pre-stored database of authorized vehicles. The system responded immediately, granting access to the verified vehicles and triggering the servo motor to open the gate. If an unauthorized vehicle was detected, the gate remained closed, a security notification was sent to the mobile application, and a buzzer alarm was activated.

**

***Figure 4.2. Mobile Application on Blynk***

The Blynk mobile application worked effectively, providing users with real-time alerts, live image feeds, and the ability to manually override the system if necessary. Users could remotely monitor vehicle logs, including timestamps, captured images, and access status, ensuring transparency and security of access control. The mobile application also played a key role in notifying users of unauthorized access attempts.

**

***Figure 4.3. Camera Web Server***

Flask and CameraWebServer integration allowed the captured images to be remotely accessed over the local network using the device's IPv4 address. The system saved the latest captured image in a specified local directory, allowing users to retrieve recent surveillance records. Additionally, Tesseract OCR processed license plates efficiently, although performance varied with lighting conditions, image resolution, and plate visibility.

Overall, the VSS system worked as intended, significantly reducing the need for manual vehicle inspections while improving security and access control efficiency. The combination of computer vision, cloud computing, and IoT technologies has resulted in a reliable, real-time, automated security solution suitable for gated communities, office buildings, and secured facilities. Future improvements include enhanced image processing techniques for low-light environments, infrared-assisted recognition, and advanced machine learning algorithms, which may further improve the accuracy and robustness of the system.

## **References**

1. *Santos, S., & Santos, S. (2021, October 30). ESP32: Getting Started with Google Spreadsheet (Realtime Database) | Random Nerd Tutorials. Random Nerd Tutorials.* [*https://randomnerdtutorials.com/esp32-Google Spreadsheet-realtime-database*](https://randomnerdtutorials.com/esp32-firebase-realtime-database)
2. *Santos, S., & Santos, S. (2021a, July 27). ESP32 with HC-SR04 Ultrasonic Sensor with Arduino IDE | Random Nerd Tutorials. Random Nerd Tutorials.* [*https://randomnerdtutorials.com/esp32-hc-sr04-ultrasonic-arduino*](https://randomnerdtutorials.com/esp32-hc-sr04-ultrasonic-arduino)
3. *Santos, S., & Santos, S. (2024, June 13). ESP32 Servo Motor Web Server with Arduino IDE | Random Nerd Tutorials. Random Nerd Tutorials.* [*https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide*](https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide)
4. *Virtual Pins | Blynk Documentation. (n.d.).* [*https://docs.blynk.io/en/blynk-library-firmware-api/virtual-pins*](https://docs.blynk.io/en/blynk-library-firmware-api/virtual-pins)
5. *Tech With Tim. (2021, September 13). Make a Python website as fast as possible! [Video]. YouTube.* [*https://www.youtube.com/watch?v=kng-mJJby8g*](https://www.youtube.com/watch?v=kng-mJJby8g)

## **Source Code**

#### **Python for Optical Character Recognition (OCR)**

1. **from** google.oauth2.service\_account **import** Credentials
2. **from** googleapiclient.discovery **import** build
3. **import** gspread
4. **import** mimetypes
5. **import** datetime
6. **from** googleapiclient.http **import** MediaFileUpload
7. **from** flask **import** Flask, request, jsonify, send\_from\_directory
8. **import** os
9. **import** re
10. **import** cv2
11. **import** numpy **as** np
12. **import** pytesseract
13. **import** requests
15. *# Flask App*
16. app = Flask(\_name\_)
17. UPLOAD\_FOLDER = "captured\_images"
18. os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)
20. *# Google API Setup*
21. scopes = [
22. "https://www.googleapis.com/auth/spreadsheets",
23. "https://www.googleapis.com/auth/drive"
24. ]
25. creds = Credentials.from\_service\_account\_file("credentials.json", scopes=scopes)
26. client = gspread.authorize(creds)
27. drive\_service = build("drive", "v3", credentials=creds)
29. *# Google Sheets & Drive Info*
30. sheet\_id = "1KcrG1me5UIWw203CqH1q5YWhv3C3LYUIvHsXx8fhbUQ"
31. sheet = client.open\_by\_key(sheet\_id)
32. parent\_folder\_id = "1HMwzXbjakVOM-oVusTzyr1V8RVNEcD29"
34. *# ESP32 & Blynk Config*
35. ESP32\_IP = "http://192.168.254.111"
36. BLYNK\_AUTH = "b189l6OU64UNP8s1R9JgfIJLwqZocuMr"
38. *# Tesseract OCR Config*
39. pytesseract.pytesseract.tesseract\_cmd = r"C:**\P**rogram Files**\T**esseract-OCR**\t**esseract.exe"
41. *# Public URL*
42. PUBLIC\_URL = "https://205b-216-247-24-239.ngrok-free.app"
43. ANNOTATED\_IMAGE\_FILENAME = "latest\_vehicle\_annotated.jpg"
45. **def** upload\_image(file\_path, folder\_id):
46. """Uploads an image to Google Drive and returns the file link."""
47. file\_name = file\_path.split("/")[-1]
48. mime\_type = mimetypes.guess\_type(file\_path)[0] **or** "application/octet-stream"
50. file\_metadata = {
51. "name": file\_name,
52. "parents": [folder\_id]
53. }
54. media = MediaFileUpload(file\_path, mimetype=mime\_type)
55. uploaded\_file = drive\_service.files().create(body=file\_metadata, media\_body=media, fields="id").execute()
57. file\_id = uploaded\_file.get("id")
58. drive\_service.permissions().create(
59. fileId=file\_id,
60. body={"role": "reader", "type": "anyone"}
61. ).execute()
63. **return** f"https://drive.google.com/uc?id={file\_id}"
65. **def** preprocess\_image(image\_path):
66. """Preprocesses the image for better OCR results."""
67. image = cv2.imread(image\_path)
68. gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)
69. blurred = cv2.GaussianBlur(gray, (5, 5), 0)
70. thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)
72. contours, \_ = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)
73. **if** contours:
74. max\_contour = max(contours, key=cv2.contourArea)
75. x, y, w, h = cv2.boundingRect(max\_contour)
76. cropped = gray[y:y+h, x:x+w]
77. **else**:
78. cropped = gray
80. **return** cv2.threshold(cropped, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)[1]
82. **def** extract\_text(image\_path):
83. """Extracts text from the image using OCR."""
84. processed\_image = preprocess\_image(image\_path)
85. raw\_text = pytesseract.image\_to\_string(processed\_image, lang="eng", config="--psm 6").strip()
86. **return** filter\_extracted\_text(raw\_text)
88. **def** filter\_extracted\_text(text):
89. """Filters extracted text to remove unwanted characters."""
90. words = re.findall(r'**\b**[a-zA-Z0-9]{3,} [a-zA-Z0-9]{3,}**\b**', text)
91. **return** " ".join(words)
93. **def** overlay\_text(image\_path, text):
94. """Overlays extracted text on the image."""
95. img = cv2.imread(image\_path)
96. **if** img **is** None:
97. **return** None
99. font = cv2.FONT\_HERSHEY\_DUPLEX
100. font\_scale = 4
101. font\_thickness = 2
102. text\_color = (0, 255, 0)
103. bg\_color = (0, 0, 0)
105. (text\_width, text\_height), baseline = cv2.getTextSize(text, font, font\_scale, font\_thickness)
106. x = img.shape[1] - text\_width - 20
107. y = img.shape[0] - 20
109. cv2.rectangle(img, (x - 5, y - text\_height - 5), (x + text\_width + 5, y + baseline + 5), bg\_color, -1)
110. cv2.putText(img, text, (x, y), font, font\_scale, text\_color, font\_thickness)
112. annotated\_image\_path = os.path.join(UPLOAD\_FOLDER, ANNOTATED\_IMAGE\_FILENAME)
113. cv2.imwrite(annotated\_image\_path, img)
114. **return** annotated\_image\_path
116. @app.route("/capture", methods=["POST"])
117. **def** capture():
118. """Handles image capture, OCR, and data logging."""
119. **try**:
120. image\_data = request.data
121. **if** **not** image\_data:
122. **return** jsonify({"status": "error", "message": "No image received"}), 400
124. timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
125. image\_filename = f"vehicle\_{timestamp.replace(':', '-')}.jpg"
126. image\_path = os.path.join(UPLOAD\_FOLDER, image\_filename)
128. **with** open(image\_path, "wb") **as** image\_file:
129. image\_file.write(image\_data)
131. detected\_text = extract\_text(image\_path)
133. **print**(f"🔎 Extracted Text: {detected\_text}")
135. *# Send extracted text to ESP32*
136. response = requests.get(f"{ESP32\_IP}/receive\_text", params={"text": detected\_text})
137. **print**(f"📡 ESP32 Response: {response.text}")
139. annotated\_image\_path = overlay\_text(image\_path, detected\_text)
140. **if** **not** annotated\_image\_path:
141. **return** jsonify({"status": "error", "message": "Failed to process image"}), 500
143. image\_drive\_link = upload\_image(annotated\_image\_path, parent\_folder\_id)
145. annotated\_image\_url = f"{PUBLIC\_URL}/uploads/{ANNOTATED\_IMAGE\_FILENAME}"
147. *# Save Data to Google Sheets*
148. worksheet = sheet.worksheet("Sheet1")
149. worksheet.append\_rows([[timestamp, detected\_text, image\_drive\_link]])
151. **return** jsonify({
152. "status": "success",
153. "timestamp": timestamp,
154. "extracted\_text": detected\_text,
155. "image\_drive\_link": image\_drive\_link,
156. "annotated\_image\_url": annotated\_image\_url
157. }), 200
158. **except** Exception **as** e:
159. **return** jsonify({"status": "error", "message": str(e)}), 500
161. @app.route("/uploads/latest\_vehicle\_annotated.jpg")
162. **def** get\_annotated\_image():
163. """Serves the latest annotated image."""
164. **return** send\_from\_directory(UPLOAD\_FOLDER, ANNOTATED\_IMAGE\_FILENAME)
166. **if** \_name\_ == "\_main\_":
167. app.run(host="0.0.0.0", port=5000, debug=True)
168. [www.googleapis.com](http://www.googleapis.com)
169. **from** google.oauth2.service\_account **import** Credentials
170. **from** googleapiclient.discovery **import** build
171. **import** gspread
172. **import** mimetypes
173. **import** datetime
174. **from** googleapiclient.http **import** MediaFileUpload
175. **from** flask **import** Flask, request, jsonify, send\_from\_directory
176. **import** os
177. **import** re
178. **import** cv2
179. **import** numpy **as** np
180. **import** pytesseract
181. **import** requests
183. *# Flask App*
184. app = Flask(\_name\_)
185. UPLOAD\_FOLDER = "captured\_images"
186. os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)
188. *# Google API Setup*
189. scopes = [
190. "https://www.googleapis.com/auth/spreadsheets",
191. "https://www.googleapis.com/auth/drive"
192. ]
193. creds = Credentials.from\_service\_account\_file("credentials.json", scopes=scopes)
194. client = gspread.authorize(creds)
195. drive\_service = build("drive", "v3", credentials=creds)
197. *# Google Sheets & Drive Info*
198. sheet\_id = "1KcrG1me5UIWw203CqH1q5YWhv3C3LYUIvHsXx8fhbUQ"
199. sheet = client.open\_by\_key(sheet\_id)
200. parent\_folder\_id = "1HMwzXbjakVOM-oVusTzyr1V8RVNEcD29"
202. *# ESP32 & Blynk Config*
203. ESP32\_IP = "http://192.168.254.111"
204. BLYNK\_AUTH = "b189l6OU64UNP8s1R9JgfIJLwqZocuMr"
206. *# Tesseract OCR Config*
207. pytesseract.pytesseract.tesseract\_cmd = r"C:**\P**rogram Files**\T**esseract-OCR**\t**esseract.exe"
209. *# Public URL*
210. PUBLIC\_URL = "https://205b-216-247-24-239.ngrok-free.app"
211. ANNOTATED\_IMAGE\_FILENAME = "latest\_vehicle\_annotated.jpg"
213. **def** upload\_image(file\_path, folder\_id):
214. """Uploads an image to Google Drive and returns the file link."""
215. file\_name = file\_path.split("/")[-1]
216. mime\_type = mimetypes.guess\_type(file\_path)[0] **or** "application/octet-stream"
218. file\_metadata = {
219. "name": file\_name,
220. "parents": [folder\_id]
221. }
222. media = MediaFileUpload(file\_path, mimetype=mime\_type)
223. uploaded\_file = drive\_service.files().create(body=file\_metadata, media\_body=media, fields="id").execute()
225. file\_id = uploaded\_file.get("id")
226. drive\_service.permissions().create(
227. fileId=file\_id,
228. body={"role": "reader", "type": "anyone"}
229. ).execute()
231. **return** f"https://drive.google.com/uc?id={file\_id}"
233. **def** preprocess\_image(image\_path):
234. """Preprocesses the image for better OCR results."""
235. image = cv2.imread(image\_path)
236. gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)
237. blurred = cv2.GaussianBlur(gray, (5, 5), 0)
238. thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)
240. contours, \_ = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)
241. **if** contours:
242. max\_contour = max(contours, key=cv2.contourArea)
243. x, y, w, h = cv2.boundingRect(max\_contour)
244. cropped = gray[y:y+h, x:x+w]
245. **else**:
246. cropped = gray
248. **return** cv2.threshold(cropped, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)[1]
250. **def** extract\_text(image\_path):
251. """Extracts text from the image using OCR."""
252. processed\_image = preprocess\_image(image\_path)
253. raw\_text = pytesseract.image\_to\_string(processed\_image, lang="eng", config="--psm 6").strip()
254. **return** filter\_extracted\_text(raw\_text)
256. **def** filter\_extracted\_text(text):
257. """Filters extracted text to remove unwanted characters."""
258. words = re.findall(r'**\b**[a-zA-Z0-9]{3,} [a-zA-Z0-9]{3,}**\b**', text)
259. **return** " ".join(words)
261. **def** overlay\_text(image\_path, text):
262. """Overlays extracted text on the image."""
263. img = cv2.imread(image\_path)
264. **if** img **is** None:
265. **return** None
267. font = cv2.FONT\_HERSHEY\_DUPLEX
268. font\_scale = 4
269. font\_thickness = 2
270. text\_color = (0, 255, 0)
271. bg\_color = (0, 0, 0)
273. (text\_width, text\_height), baseline = cv2.getTextSize(text, font, font\_scale, font\_thickness)
274. x = img.shape[1] - text\_width - 20
275. y = img.shape[0] - 20
277. cv2.rectangle(img, (x - 5, y - text\_height - 5), (x + text\_width + 5, y + baseline + 5), bg\_color, -1)
278. cv2.putText(img, text, (x, y), font, font\_scale, text\_color, font\_thickness)
280. annotated\_image\_path = os.path.join(UPLOAD\_FOLDER, ANNOTATED\_IMAGE\_FILENAME)
281. cv2.imwrite(annotated\_image\_path, img)
282. **return** annotated\_image\_path
284. @app.route("/capture", methods=["POST"])
285. **def** capture():
286. """Handles image capture, OCR, and data logging."""
287. **try**:
288. image\_data = request.data
289. **if** **not** image\_data:
290. **return** jsonify({"status": "error", "message": "No image received"}), 400
292. timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
293. image\_filename = f"vehicle\_{timestamp.replace(':', '-')}.jpg"
294. image\_path = os.path.join(UPLOAD\_FOLDER, image\_filename)
296. **with** open(image\_path, "wb") **as** image\_file:
297. image\_file.write(image\_data)
299. detected\_text = extract\_text(image\_path)
301. **print**(f"🔎 Extracted Text: {detected\_text}")
303. *# Send extracted text to ESP32*
304. response = requests.get(f"{ESP32\_IP}/receive\_text", params={"text": detected\_text})
305. **print**(f"📡 ESP32 Response: {response.text}")
307. annotated\_image\_path = overlay\_text(image\_path, detected\_text)
308. **if** **not** annotated\_image\_path:
309. **return** jsonify({"status": "error", "message": "Failed to process image"}), 500
311. image\_drive\_link = upload\_image(annotated\_image\_path, parent\_folder\_id)
313. annotated\_image\_url = f"{PUBLIC\_URL}/uploads/{ANNOTATED\_IMAGE\_FILENAME}"
315. *# Save Data to Google Sheets*
316. worksheet = sheet.worksheet("Sheet1")
317. worksheet.append\_rows([[timestamp, detected\_text, image\_drive\_link]])
319. **return** jsonify({
320. "status": "success",
321. "timestamp": timestamp,
322. "extracted\_text": detected\_text,
323. "image\_drive\_link": image\_drive\_link,
324. "annotated\_image\_url": annotated\_image\_url
325. }), 200
326. **except** Exception **as** e:
327. **return** jsonify({"status": "error", "message": str(e)}), 500
329. @app.route("/uploads/latest\_vehicle\_annotated.jpg")
330. **def** get\_annotated\_image():
331. """Serves the latest annotated image."""
332. **return** send\_from\_directory(UPLOAD\_FOLDER, ANNOTATED\_IMAGE\_FILENAME)
334. **if** \_name\_ == "\_main\_":
335. app.run(host="0.0.0.0", port=5000, debug=True)
336. www.googleapis.com

#### **ESP32 Code (main())**

1. #define BLYNK\_TEMPLATE\_ID "TMPL6tSitw7kx"
2. #define BLYNK\_TEMPLATE\_NAME "Tanawa Template"
3. #define BLYNK\_AUTH\_TOKEN "3f9t3xoMaZgWy2MLqTUG0ulcXDcvXjyh"
5. #define BLYNK\_PRINT Serial
6. #include <BlynkSimpleEsp32.h>
7. #include <HTTPClient.h>
8. #include <Arduino.h>
9. #include <WiFi.h>
10. #include "time.h"
11. #include <ESP32Servo.h>
12. #include <WebServer.h>
14. WebServer server(80); *// Create an HTTP server on port 80*
16. char auth[] = BLYNK\_AUTH\_TOKEN;
17. BlynkTimer timer;
19. *// Wi-Fi credentials*
20. const char\* ssid = ""; *// Wifi and Password are hidden for data privacy purposes*
21. const char\* password = "";
22. const char\* ntpServer = "time.google.com"; *// Using Google NTP server*
24. const long gmtOffset\_sec = 3600 \* 8; *// Philippines Time (UTC +8)*
25. const int daylightOffset\_sec = 0; *// No DST in the Philippines*
27. *// Pin definitions*
28. const int trigPin = 5;
29. const int echoPin = 18;
30. const int ledPin = 19;
31. const int buzzerPin = 21;
32. const int servoPin = 23;
34. *// Constants for sound speed and distance conversion*
35. #define SOUND\_SPEED 0.034
36. #define CM\_TO\_INCH 0.393701
38. long duration;
39. float distanceCm;
40. float distanceInch;
41. String receivedText;
43. *// Global variables for task synchronization*
44. QueueHandle\_t alarmQueue;
46. Servo servo1;
48. String time\_info;
49. int alarm\_info;
50. bool is\_servo\_fixed\_control;
52. *// Global flag to ensure only one capture request is sent per detection event.*
53. volatile bool captureTriggered = **false**;
55. #define SWITCH\_VPIN V2 // Use V2 for the switch in the Blynk app
56. #define CAPTURE\_VPIN V3
58. *// Blynk function to handle the switch state*
59. BLYNK\_WRITE(SWITCH\_VPIN) {
60. int switchState = param.asInt();
61. if (switchState == 1) {
62. is\_servo\_fixed\_control = **true**;
63. servo1.write(0);
64. Serial.println("Servo moved to 0°");
65. } else {
66. is\_servo\_fixed\_control = **false**;
67. servo1.write(90);
68. Serial.println("Servo moved to 90°");
69. }
70. }
72. BLYNK\_WRITE(CAPTURE\_VPIN) {
73. int captureState = param.asInt();
74. if (captureState == 1) {
75. bool alarmSignal = **true**;
76. xQueueSend(alarmQueue, &alarmSignal, portMAX\_DELAY);
77. }
78. }
80. void controlServoTask(void \*pvParameters) {
81. while (**true**) {
82. if (receivedText == "LMM 8923") { *// Check if the plate number matches*
83. Serial.println("✅ Authorized Vehicle Detected! Opening Gate...");
84. servo1.write(0); *// Open the gate*
85. vTaskDelay(pdMS\_TO\_TICKS(5000)); *// Keep gate open for 5 seconds*
86. Serial.println("❌ Closing Gate...");
87. servo1.write(90); *// Close the gate*
88. receivedText = ""; *// Reset the received text to avoid re-triggering*
89. }
90. vTaskDelay(pdMS\_TO\_TICKS(500)); *// Check every 500ms*
91. }
92. }

95. void sendCaptureRequest() {
96. HTTPClient http;
97. Serial.println("Sending capture request to Flask server...");
99. http.begin("http://192.168.254.112/capture");
100. http.addHeader("Content-Type", "application/json");
101. int httpResponseCode = http.GET();
103. if (httpResponseCode > 0) {
104. Serial.print("Capture Request Sent Successfully, Response code: ");
105. Serial.println(httpResponseCode);
106. } else {
107. Serial.print("Capture Request Failed: ");
108. Serial.println(http.errorToString(httpResponseCode).c\_str());
109. }
111. http.end();
112. }
114. void sendSensor() {
115. struct tm timeinfo;
116. if (!getLocalTime(&timeinfo)) {
117. Serial.println("Failed to obtain time");
118. return;
119. }
120. char formattedTime[20];
121. strftime(formattedTime, sizeof(formattedTime), "%Y%m%d\_%H%M%S", &timeinfo);
122. time\_info = String(formattedTime);
123. xQueueReceive(alarmQueue, &alarm\_info, portMAX\_DELAY);
124. Blynk.virtualWrite(V0, time\_info);
125. Blynk.virtualWrite(V1, receivedText);
126. }
128. void sendplateNumber() {
129. Blynk.virtualWrite(V1, receivedText);
130. }
132. void printLocalTime() {
133. struct tm timeinfo;
134. if (!getLocalTime(&timeinfo)) {
135. Serial.println("Failed to obtain time");
136. return;
137. }
138. char formattedTime[20];
139. strftime(formattedTime, sizeof(formattedTime), "%Y%m%d\_%H%M%S", &timeinfo);
140. Serial.print(formattedTime);
141. }
143. *// Task to measure distance using the ultrasonic sensor*
144. void measureDistanceTask(void \*pvParameters) {
145. unsigned long objectDetectedTime = 0;
146. while (**true**) {
147. digitalWrite(trigPin, **LOW**);
148. delayMicroseconds(2);
149. digitalWrite(trigPin, **HIGH**);
150. delayMicroseconds(10);
151. digitalWrite(trigPin, **LOW**);
153. duration = pulseIn(echoPin, **HIGH**);
154. if (duration > 0 && duration < 30000) {
155. distanceCm = duration \* SOUND\_SPEED / 2;
156. distanceInch = distanceCm \* CM\_TO\_INCH;
158. if (distanceInch < 6) {
159. if (objectDetectedTime == 0) {
160. objectDetectedTime = millis();
161. } else if (millis() - objectDetectedTime >= 3000) {
162. bool alarmSignal = **true**;
163. xQueueSend(alarmQueue, &alarmSignal, portMAX\_DELAY);
164. }
165. } else {
166. objectDetectedTime = 0;
167. bool alarmSignal = **false**;
168. xQueueSend(alarmQueue, &alarmSignal, portMAX\_DELAY);
169. }
170. }
171. vTaskDelay(pdMS\_TO\_TICKS(100));
172. }
173. }
175. void alarmTask(void \*pvParameters) {
176. bool lastAlarm = **false**;
178. while (**true**) {
179. bool alarm = **false**;
180. if (xQueueReceive(alarmQueue, &alarm, portMAX\_DELAY)) {
181. if (alarm && !lastAlarm) {
182. lastAlarm = **true**;
183. Serial.print("Date/Time: ");
184. printLocalTime();
185. Serial.println(" - VEHICLE DETECTED!");
187. receivedText = "";
188. Serial.println("🔄 Reset receivedText for new detection.");
190. for (int i = 0; i < 3; i++) {
191. digitalWrite(ledPin, **HIGH**);
192. digitalWrite(buzzerPin, **HIGH**);
193. vTaskDelay(pdMS\_TO\_TICKS(100));
194. digitalWrite(ledPin, **LOW**);
195. digitalWrite(buzzerPin, **LOW**);
196. vTaskDelay(pdMS\_TO\_TICKS(100));
197. }
199. if (!captureTriggered) {
200. sendCaptureRequest();
201. captureTriggered = **true**;
202. }
203. } else if (!alarm && lastAlarm) {
204. lastAlarm = **false**;
205. captureTriggered = **false**;
206. }
207. }
208. vTaskDelay(pdMS\_TO\_TICKS(100));
209. }
210. }
212. void setup() {
213. Serial.begin(115200);
215. pinMode(trigPin, **OUTPUT**);
216. pinMode(echoPin, **INPUT**);
217. pinMode(ledPin, **OUTPUT**);
218. pinMode(buzzerPin, **OUTPUT**);
220. servo1.attach(servoPin);
222. Serial.print("Connecting to ");
223. Serial.println(ssid);
224. WiFi.begin(ssid, password);
225. while (WiFi.status() != WL\_CONNECTED) {
226. delay(500);
227. Serial.print(".");
228. }
229. Serial.println("**\n**WiFi connected.");
231. configTime(gmtOffset\_sec, daylightOffset\_sec, ntpServer);
232. struct tm timeinfo;
233. int attempts = 0;
234. while (!getLocalTime(&timeinfo) && attempts < 5) {
235. Serial.println("Failed to obtain time, retrying...");
236. delay(2000);
237. attempts++;
238. }
240. if (attempts == 5) {
241. Serial.println("Failed to obtain time.");
242. } else {
243. Serial.println("Time synchronized!");
244. printLocalTime();
245. Serial.println();
246. }
248. alarmQueue = xQueueCreate(5, sizeof(bool));
250. Blynk.begin(auth, ssid, password);
251. timer.setInterval(10000L, sendSensor);
253. server.on("/receive\_text", HTTP\_GET, []() {
254. if (server.hasArg("text")) {
255. receivedText = server.arg("text");
256. Serial.print("📌 Extracted Text Received: ");
257. Serial.println(receivedText);
258. sendSensor();
259. }
260. server.send(200, "text/plain", "Text received");
261. });
263. server.begin();
264. Serial.println("🔹 HTTP Server Started on ESP32");
266. xTaskCreate(measureDistanceTask, "Measure Distance", 4096, NULL, 2, NULL);
267. xTaskCreate(alarmTask, "Alarm", 4096, NULL, 1, NULL);
268. xTaskCreate(controlServoTask, "Control Servo", 4096, NULL, 1, NULL);
269. }
271. void loop() {
272. Blynk.run();
273. timer.run();
274. server.handleClient();
276. if (alarm\_info == 1) {
277. Blynk.logEvent("vehicle\_detected", "Vehicle Detected! It wants to enter.");
278. }
279. }

#### **ESP32CAM Module Separate Code**

1. #include "esp\_camera.h"
2. #include <WiFi.h>
3. #include <HTTPClient.h>
4. #include <WebServer.h>
6. #define CAMERA\_MODEL\_AI\_THINKER
7. #include "camera\_pins.h"
9. *// WiFi credentials*
10. const char\* ssid = "AOBY\_2.4";
11. const char\* password = "PASSWORD\_1995";
13. *// Flask server URL*
14. const char\* flaskServerUrl = "http://192.168.254.108:5000/capture";
16. *// Web server on port 80*
17. WebServer server(80);
19. void handleCapture() {
20. Serial.println("[INFO] Received HTTP request on /capture");
22. *// Capture an image*
23. camera\_fb\_t \*fb = esp\_camera\_fb\_get();
24. if (!fb) {
25. Serial.println("[ERROR] Camera capture failed");
26. return;
27. }
28. Serial.printf("[INFO] Captured image size: %d bytes**\n**", fb->len);

31. *// Send POST request to Flask server*
32. HTTPClient http;
33. http.begin(flaskServerUrl);
34. http.addHeader("Content-Type", "application/octet-stream");
36. Serial.println("[INFO] Sending image to Flask server...");
37. int httpResponseCode = http.POST(fb->buf, fb->len);
39. if (httpResponseCode > 0) {
40. String response = http.getString();
41. Serial.printf("[SUCCESS] Flask server response: %s**\n**", response.c\_str());
42. server.send(200, "application/json", response);
43. } else {
44. Serial.printf("[ERROR] HTTP POST failed: %s**\n**", http.errorToString(httpResponseCode).c\_str());
45. server.send(500, "text/plain", "Error sending image to Flask");
46. }
48. http.end();
49. esp\_camera\_fb\_return(fb); *// Release buffer only after HTTP request completes*
50. }
52. void setup() {
53. Serial.begin(115200);
54. Serial.println("**\n**[INFO] Starting ESP32-CAM...");
56. *// Initialize Camera*
57. camera\_config\_t config;
58. config.ledc\_channel = LEDC\_CHANNEL\_0;
59. config.ledc\_timer = LEDC\_TIMER\_0;
60. config.pin\_d0 = Y2\_GPIO\_NUM;
61. config.pin\_d1 = Y3\_GPIO\_NUM;
62. config.pin\_d2 = Y4\_GPIO\_NUM;
63. config.pin\_d3 = Y5\_GPIO\_NUM;
64. config.pin\_d4 = Y6\_GPIO\_NUM;
65. config.pin\_d5 = Y7\_GPIO\_NUM;
66. config.pin\_d6 = Y8\_GPIO\_NUM;
67. config.pin\_d7 = Y9\_GPIO\_NUM;
68. config.pin\_xclk = XCLK\_GPIO\_NUM;
69. config.pin\_pclk = PCLK\_GPIO\_NUM;
70. config.pin\_vsync = VSYNC\_GPIO\_NUM;
71. config.pin\_href = HREF\_GPIO\_NUM;
72. config.pin\_sccb\_sda = SIOD\_GPIO\_NUM;
73. config.pin\_sccb\_scl = SIOC\_GPIO\_NUM;
74. config.pin\_pwdn = PWDN\_GPIO\_NUM;
75. config.pin\_reset = RESET\_GPIO\_NUM;
76. config.xclk\_freq\_hz = 20000000;
77. config.pixel\_format = PIXFORMAT\_JPEG;
79. if (psramFound()) {
80. config.frame\_size = FRAMESIZE\_UXGA;
81. config.jpeg\_quality = 10;
82. config.fb\_count = 2;
83. } else {
84. config.frame\_size = FRAMESIZE\_SVGA;
85. config.jpeg\_quality = 12;
86. config.fb\_count = 1;
87. }
89. if (esp\_camera\_init(&config) != ESP\_OK) {
90. Serial.println("[ERROR] Camera initialization failed!");
91. return;
92. }
93. Serial.println("[INFO] Camera initialized");
95. *// Connect to WiFi*
96. WiFi.begin(ssid, password);
97. Serial.print("[INFO] Connecting to WiFi...");
98. while (WiFi.status() != WL\_CONNECTED) {
99. delay(500);
100. Serial.print(".");
101. }
103. Serial.println("**\n**[INFO] WiFi connected!");
104. Serial.print("[INFO] ESP32-CAM IP Address: ");
105. Serial.println(WiFi.localIP());
107. *// Web Server*
108. server.on("/capture", HTTP\_GET, handleCapture);
109. server.begin();
110. Serial.println("[INFO] Web server started!");
111. }

114. void loop() {
115. server.handleClient();
116. }
117. 192.168.254.108