SMART PARCEL SORTING CONVEYOR SYSTEM WITH QR BASED IDENTIFICATION & IOT NOTIFICATION

TABLE OF CONTENTS

Chapter	Title	Page. No
1	Abstract	3
2	Introduction	4
3	Review of literature	6
4	Design consideration	7
5	Circuit diagram	12
6	Implementations and results	15
7	Program and user interface	23
8	Advantages and applications	38
9	Conclusions and future scope	39
10	references	40

ABSTRACT

Smart logistics and automated parcel handling are gaining momentum across industries. In this project, we propose and implement a smart sorting conveyor system using an ESP32-CAM module, IoT-based cloud monitoring, and Telegram notifications. The system is designed to read QR codes on packages, extract relevant customer and destination data, and sort parcels automatically using servo-controlled gates on a moving conveyor belt.

The ESP32-CAM captures the QR code on each package and attempts to read it up to 7 times. If the QR code is invalid or unreadable after 3 tries, a flash LED is triggered to assist with reading. Upon successful decoding, the system extracts the name, phone number, and location from the QR data. This information is logged to a Google Sheet and also triggers a Telegram message to the customer indicating that the package has reached the facility. Based on the location, the system dynamically assigns the package to one of six gates spread across three junctions on the conveyor. Each junction is powered by MG90S servos, which either open the appropriate gate or push the parcel into it.

Additionally, the system features Arduino Cloud integration to remotely reset the servos and halt the system when required. Invalid or unrecognized packages are sent to a final section for manual inspection. This system combines mechanical actuation with IoT capabilities to present an efficient, cost-effective, and scalable model for automating logistics workflows.

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The logistics and supply chain industry is evolving rapidly with increasing demand for automation, real-time monitoring, and efficient sorting systems. Traditional manual parcel handling methods are time-consuming, prone to human error, and not scalable. To address these limitations, smart conveyor systems integrated with IoT and computer vision technologies have emerged as a viable solution.

Smart conveyor systems can identify, classify, and route parcels automatically using sensors and microcontrollers. In this project, we employ an ESP32-CAM module to scan QR codes on parcels and make decisions based on the extracted information, thereby reducing human involvement and improving efficiency.

1.2 OBJECTIVES

1. To develop a smart conveyor-based parcel sorting system that uses QR code recognition.

2 To implement real-time communication and cloud-based monitoring using Arduino IoT Cloud.

3 To integrate a Telegram alert system for notifying customers upon package arrival.

4 To provide an efficient fallback mechanism for unrecognized or unreadable QR codes.

5 To log QR data using Google sheets to maintain parcel history.

1.3 RELEVANCE

This project is highly relevant in today’s fast-paced e-commerce and logistics sectors, where accuracy and speed are critical. Automating the sorting process reduces labor costs, increases throughput, and minimizes errors. By incorporating cloud-based control and customer notification systems, this project also enhances user experience and operational transparency.

1.4. REPORT ORGANIZATION

This report is structured into the following chapters:

- **Chapter 1:** Introduction – background, objective and relevance of the project.
- **Chapter 2**: Review of Literature – summarizes related work and foundational technologies.
- **Chapter 3**: Design Considerations – discusses hardware and software design choices.
- **Chapter 4**: Circuit Diagram – presents the wiring and working of the circuit.
- **Chapter 5**: Implementation and Results – shows software and hardware outcomes.
- **Chapter 6**: Programs and User Interface – explains the code, cloud functions, and UI.
- **Chapter 7**: Advantages and Applications – describes real-world utility.
- **Chapter 8**: Conclusion and Future Scope – outlines key insights and possible extensions.

CHAPTER 2
REVIEW OF LITERATURE

1. The paper "Robosort: Vision-Based Sorting System Using ESP32-CAM and Servo Motors" (2021) explores an automated sorting mechanism utilizing vision sensors. It outlines a similar use of ESP32-CAM and servo motors to identify and redirect objects. This work helped us recognize the potential and limitations of onboard image processing for real-time sorting systems.

2. "QR Code Detection and Servo Control Using ESP32-CAM" (2023) presents a lightweight implementation for QR scanning combined with actuation. It confirms the feasibility of using QR payload data to trigger servo motors, reinforcing our decision to design our servo gate logic using similar principles.

3. The IEEE paper "Design and Implementation of a QR Code Based Intelligent Sorting System Using Raspberry Pi" (ICAICE, 2022) describes a larger-scale implementation using a Pi-based camera. While more powerful, Raspberry Pi setups are bulkier and costlier than ESP32-CAM. This helped us prioritize cost-effectiveness without compromising functionality.

4. "ESP32 Camera Module Based QR Code Verification for Output On/Off" (2025) demonstrates basic input-output control based on QR validation. It inspired the logic we adopted to activate downstream operations only after a successful QR decode, enhancing the reliability of our trigger system.

5. The guide "ESP32 PWM/LEDC Servo Multiplexing Guide, using Adafruit library" offers critical insight into running multiple servos using the PCA9685 driver. This enabled our use of six MG90S servos in synchronized control, optimizing hardware efficiency and timing accuracy in the junction sorting logic.

CHAPTER 3

DESIGN CONSIDERATIONS

3.1 INTRODUCTION

Designing a smart sorting conveyor system involves both hardware interfacing and software control. The goal is to reliably read QR codes using a vision module, extract relevant data, and trigger physical actions via motorized gates in real time. Each submodule, such as the ESP32-CAM, motor driver, PWM servo driver, and cloud communication, must be integrated carefully for synchronized operation. The modular design helps in making the system scalable, serviceable, and adaptable for different routing logic in logistics environments.

3.2 COMPONENTS REQUIRED

Components	**Description**
ESP32-CAM Module	Captures and decodes QR codes using an onboard camera and Wi-Fi capabilities
MG90S Servo Motors (6 units)	Controls the opening and pushing gates at sorting junctions.
PCA9685 16-channel PWM Servo Driver Module	Enables simultaneous control of multiple servo motors using I2C.
L298N Dual H-Bridge Motor Driver	rives the 12V DC motor for conveyor movement using control signals.
12V DC Gear Motor	Powers the conveyor belt to move parcels forward
Logic Level Converter (3.3V to 5V)	Matches voltage levels between ESP32 (3.3V) and 5V modules.
Power Supply Modules: 5V and 12V	Provide regulated power to servos, motor, and ESP32-CAM.
Conveyor Belt Assembly with 3 Junctions	Mechanically transports and sorts parcels to designated gates.
Jumper wires, breadboards, mechanical/physical supports	Used for circuit connections and structural setup.

Table 1: Component description

3.3 PIN DIAGRAMS AND CONNECTIONS

ESP32-CAM	**PCA9685**	**L298N**	**12V DC Motor**	**External power**	**Logic Shifter**

GPIO 13	SDA				
GPIO 15	SCL				
GND	GND				
5V	Vcc				
	V+			5V	
	GND			GND	
GPIO 12					3.3V
		IN1			5V
GND		GND			
		12V			12V
		GND			GND
		OUT1	+VE		
		OUT2	-VE		

Table 2: Pin Connections

*Channels 0–5 of PCA9685 are connected to servos controlling 6 gates.

ESP32-CAM:

- GPIO 0: XCLK (camera clock)
- GPIO 4: Flash LED control
- GPIO 13: I2C SDA (to PWM driver)
- GPIO 15: I2C SCL (to PWM driver)
- GPIO 12: Conveyor motor trigger (via motor driver)
- GPIOs 32, 26, 27, 25, 23, etc.: connected to camera sensor

PCA9685 PWM Driver:

- Address: 0x40 (default I2C)
- Channels 0–5: Connected to servos controlling 6 gates
- VCC and GND connected to 5V power supply
- SDA/SCL connected to ESP32-CAM

L298N Motor Driver:

- IN1: ESP32 GPIO 12 via logic level converter
- IN2: GND
- 12V and GND: Power lines to motor

3.4. SUB-CIRCUITS

1. Camera + Flash Subcircuit:
   a. ESP32-CAM module with connected OV2640 camera
   b. GPIO 4 used to control a flash LED via resistor
2. Servo Control Subcircuit:
   a. PCA9685 connected to 6 MG90S servos
   b. Common 5V power rail for servos
   c. I2C from ESP32 to driver
3. Motor Driver Subcircuit:
   a. L298N controls a single 12V DC motor
   b. Triggered by ESP32 digital output pin through logic converter
   c. Motor connected to conveyor belt
4. Cloud and Communication:
   a. Arduino IoT Cloud integration for variables like stopSystem and resetServos
   b. Telegram API for alerts
   c. Google Sheets via HTTP GET using Apps Script

3.5. BLOCK DIAGRAM

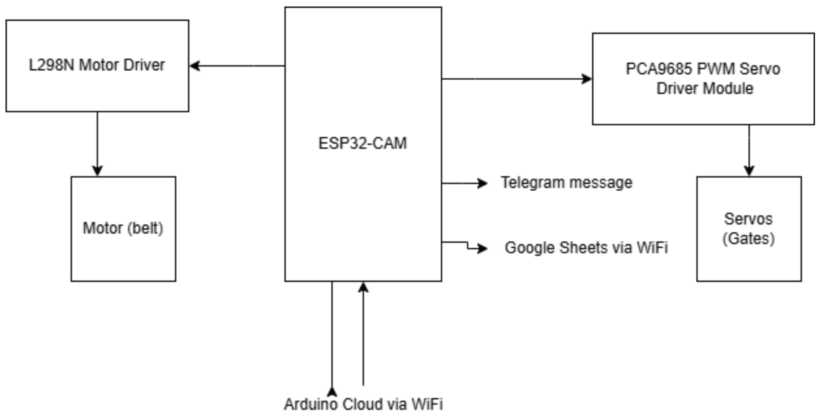


Fig 1:Block Diagram

3.6. SOFTWARE DESIGN

The software controls all system components via the ESP32-CAM using Arduino C++. It integrates QR code scanning, servo actuation, motor control, and IoT functions.

QR Code Scanning
The ESP32-CAM captures and decodes QR codes using the ESP32QRCodeReader library. It retries up to 7 times, turning on the flash after 3 failed attempts. Extracted data includes the customer's name, phone number, and delivery location.

Location Mapping and Gate Logic
A predefined mapping assigns each location to a servo gate and level. Unknown locations are flagged for manual handling. Gate positions are tracked using a gates[] array.

Conveyor and Gate Control
A 12V DC motor drives the conveyor via an L298N motor driver (triggered by GPIO 12 via logic level shifter). Servos (MG90) are controlled using a PCA9685 PWM driver to open gates and push parcels.

Cloud Integration

- Telegram API: Sends delivery notifications using parsed QR data.
- Google Sheets: Logs parcel details via a webhook.
- Arduino Cloud: Allows remote control with two variables:
    - stopSystem to pause operation
    - resetServos to reset servo angles

CHAPTER 4
CIRCUIT DIAGRAM

4.1. INTRODUCTION

The Smart Sorting Conveyor System is an IoT-enabled automation project designed to streamline parcel sorting using computer vision and servo-actuated gates. The system leverages an ESP32-CAM to scan QR codes on parcels, extract recipient information such as name, phone number, and delivery location, and sort them into designated bins. Based on the extracted data, the system controls servos to operate gates at specific conveyor belt junctions. Integration with Telegram API and Google Sheets provides real-time notification and logging functionality. Cloud connectivity is facilitated through Arduino IoT Cloud, enabling remote control and system monitoring.

4.2. WORKING

1. **ESP32-CAM Initialization:**

    a. Acts as the main controller.

    b. Initializes camera and connects to Wi-Fi.

    c. Sets up I2C communication with the PCA9685 servo driver (SDA: GPIO 13, SCL: GPIO 15).

2. **QR Code Scanning:**

    a. Uses onboard camera to scan the QR code on the parcel.

    b. Tries up to 7 times; turns on flash LED (connected to GPIO 4) after 3 failed attempts.

    c. Extracts customer name, phone number, and delivery location from the QR data.

3. **Location-to-Gate Mapping:**

    a. Based on the delivery location, the system maps the parcel to a specific servo gate and level.

4. **Google Sheets Logging & Telegram Notification:**

    a. Logs customer data to a Google Sheet using a webhook.

    b. Sends a delivery confirmation message via Telegram API using extracted QR data.

5. **Conveyor Belt Control:**

    a. GPIO 12 is used to control the conveyor motor via an L298N motor driver.

    b. Since ESP32 outputs 3.3V and the L298N requires 5V logic, a **logic level shifter** is used.

    c. IN1 of L298N is connected to GPIO 12 via the level shifter.

    d. IN2 is grounded for unidirectional movement.

    e. The motor is powered by a 12V external supply.

6. **Gate Actuation Using Servos:**

    a. PCA9685 drives 6 servo motors connected to its first 6 PWM channels.

    b. Each junction has 2 servos: one to open the gate and one to push the parcel into the bin.

    c. Servos are powered by an external 5V 2A supply connected to both V+ and VCC pins on the PCA9685.

7. **Common Grounding:**

a. All modules (ESP32-CAM, PCA9685, L298N) share a **common GND** for stable signal referencing.

8. **End of Process:**

a. After sorting, the system waits for the QR code to disappear from the camera view before scanning the next parcel.

4.3. CIRCUIT DIAGRAM

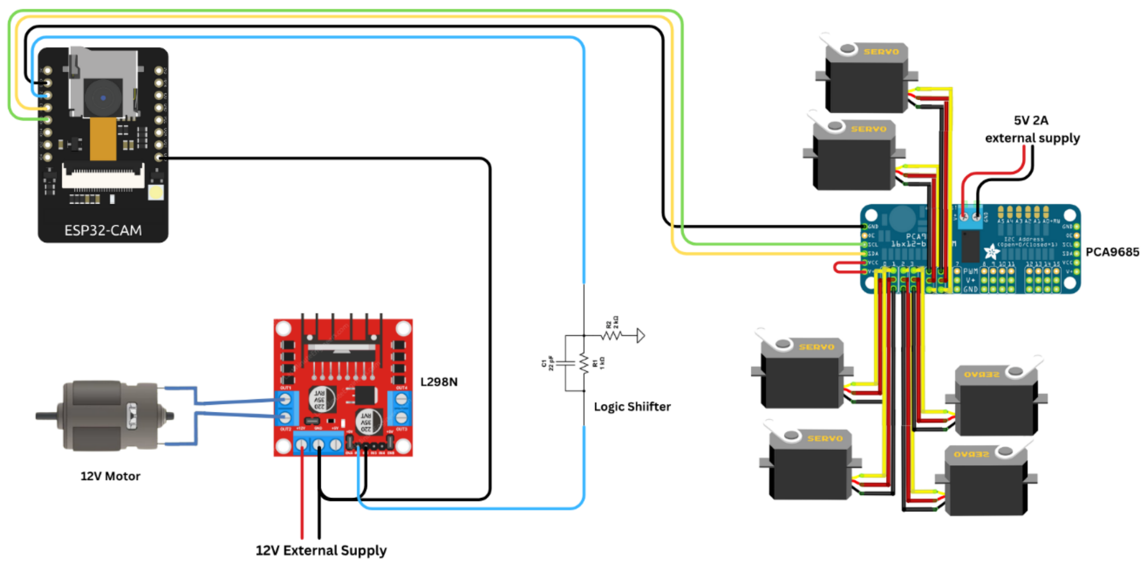


Fig 2: Circuit Diagram

CHAPTER 5

IMPLEMENTATION & RESULTS

5.1. Software Implementation

The software for the Smart Sorting Conveyor System is developed using the **Arduino IDE** and written in **C++** using the **ESP32 platform**. The implementation integrates camera-based QR scanning, motor and servo control, and cloud communication. Below is an overview of the major components of the software:

5.2 Libraries Used

- `ESP32QRCodeReader`: For QR code detection from the ESP32-CAM.
- `esp_camera.h`: To interface with the camera module.
- `Adafruit_PWMServoDriver.h`: To control servos via the PCA9685 driver over I2C.
- `HTTPClient.h`: For sending HTTP requests to Google Sheets and Telegram APIs.
- `ArduinoIoTCloud.h`: For cloud control via Arduino IoT Cloud.

5.3 Key Functional Blocks

1. QR Code Reading

- The ESP32-CAM continuously tries to detect a QR code using its camera.
- If the QR isn't detected in 3 attempts, the flash (GPIO 4) is turned on.
- The payload is parsed to extract **name, phone number, and location**.

***2.* Location-Based Gate Logic**

- The `validateLoc()` function matches the location from QR data to a specific gate index and level.

- A `gates[]` array keeps track of gate-level assignments for each parcel.

3. Conveyor Motor Control

- The conveyor belt motor is controlled by setting GPIO 12 HIGH for a fixed time.
- A logic level shifter is used to safely interface the 3.3V GPIO with the 5V motor driver.

4. Servo Gate Control

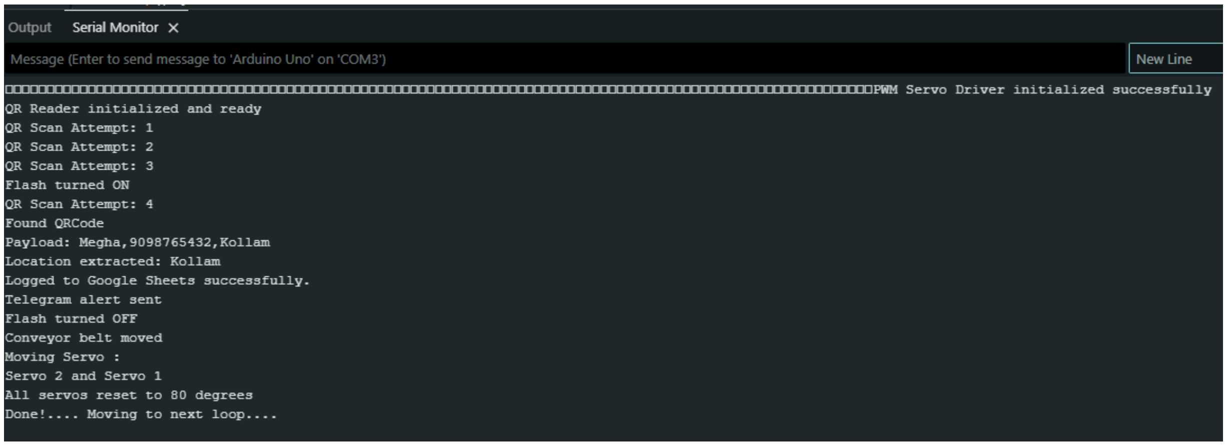
- The `openGate()` function activates two servos at a time:
  - One opens the gate at the correct junction.
  - The other pushes the parcel into the bin.
- Servos are reset to 80° using `resetAllServos()` after each operation.

5. Data Logging and Alerts

- A GET request is sent to a Google Apps Script to log:
  - Name, phone number, and location.
- A Telegram message is sent using the Telegram Bot API to notify the customer.

6. Cloud Control

- The `stopSystem` variable (from Arduino Cloud) pauses the loop until it is toggled off remotely.
- The `resetServos` variable remotely resets all servo positions.


Output Serial Monitor

Message (Enter to send message to 'Arduino Uno' on 'COM3')

New Line

```
PWM Servo Driver initialized successfully
QR Reader initialized and ready
QR Scan Attempt: 1
QR Scan Attempt: 2
QR Scan Attempt: 3
Flash turned ON
QR Scan Attempt: 4
Found QRCode
Payload: Megha,9098765432,Kollam
Location extracted: Kollam
Logged to Google Sheets successfully.
Telegram alert sent
Flash turned OFF
Conveyor belt moved
Moving Servo :
Servo 2 and Servo 1
All servos reset to 80 degrees
Done!.... Moving to next loop....
```

Fig 3: Serial Monitor output

5.4. Google Sheets Implementation

To maintain a real-time cloud-based record of all parcel scans, the Smart Sorting Conveyor System logs every QR code scan to a Google Sheet using Google Apps Script. This sheet serves as a centralized database for tracking parcel details such as customer name, phone number, location, and timestamp.

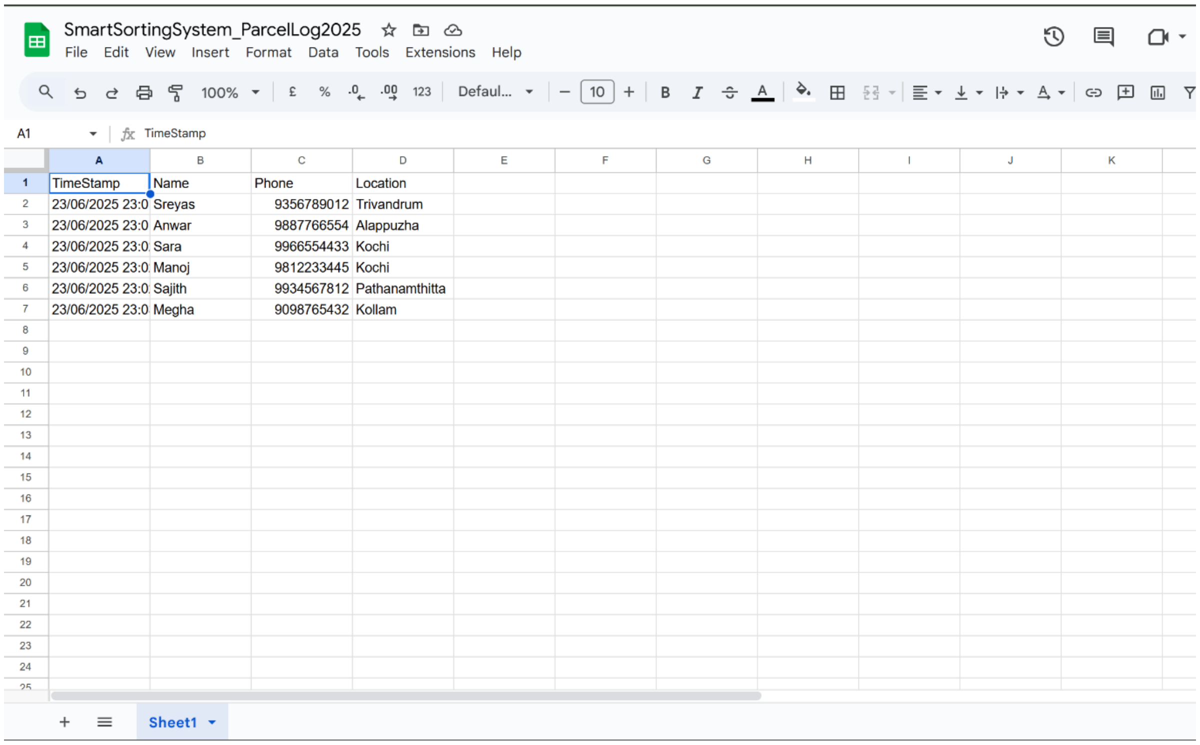
1. Implementation Overview

- When a parcel is scanned successfully, its **QR data** is parsed.
- An HTTP GET request is sent to a **Google Apps Script Web App** URL with the parsed values.
- The script appends the received data to the Google Sheet along with the current timestamp.

2. Sheet Format

The sheet has four columns:

- **TimeStamp**
- **Name**
- **Phone**
- **Location**


SmartSortingSystem_ParcelLog2025
File Edit View Insert Format Data Tools Extensions Help
A1
TimeStamp
Sheet1

TimeStamp	Name	Phone	Location
23/06/2025 23:0	Sreyas	9356789012	Trivandrum
23/06/2025 23:0	Anwar	9887766554	Alappuzha
23/06/2025 23:0	Sara	9966554433	Kochi
23/06/2025 23:0	Manoj	9812233445	Kochi
23/06/2025 23:0	Sajith	9934567812	Pathanamthitta
23/06/2025 23:0	Megha	9098765432	Kollam

3. Google Apps Script Code

Below is the Google Apps Script deployed as a web app to handle data logging:

```
function doGet(e) {
  return handleRequest(e);
}

function doPost(e) {
  return handleRequest(e);
}

function handleRequest(e) {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();

  var timestamp = new Date();
  var name = e.parameter.name || "NA";
  var phone = e.parameter.phone || "NA";
  var location = e.parameter.location || "NA";

  sheet.appendRow([timestamp, name, phone, location]);

  return ContentService.createTextOutput("Success");
}

```

5.5. Hardware Implementation

Smart Sorting Conveyor System Model :

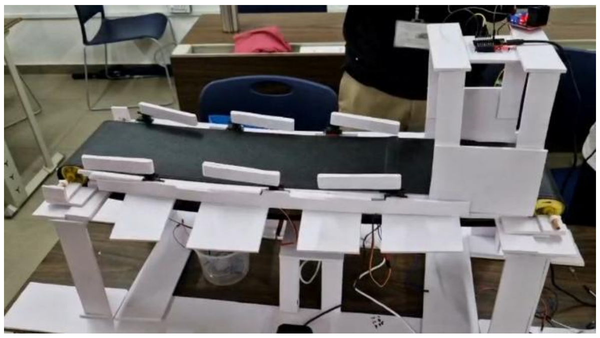


Fig 4: Model

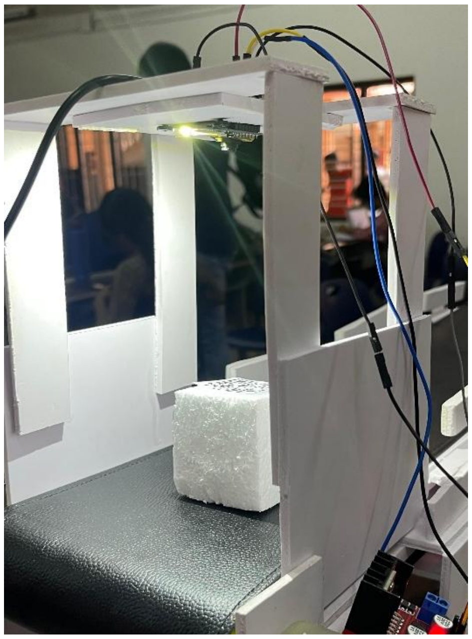


Fig 5: Working of Flash during scanning

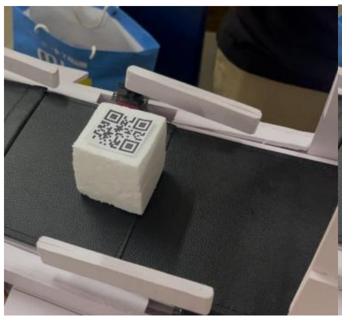


Fig 6: Parcel arrives at gate

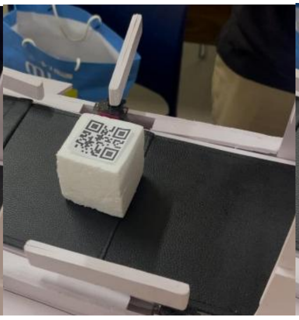


Fig 7: Gate opens

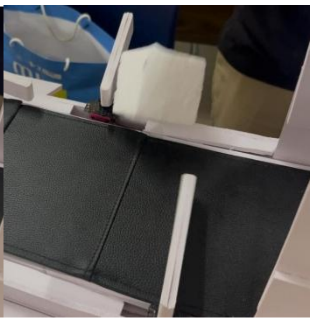


Fig 8: Parcel gets pushed out

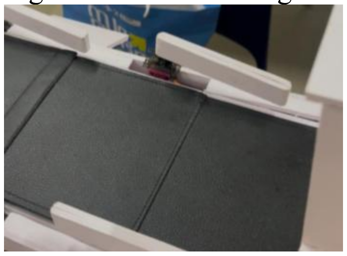


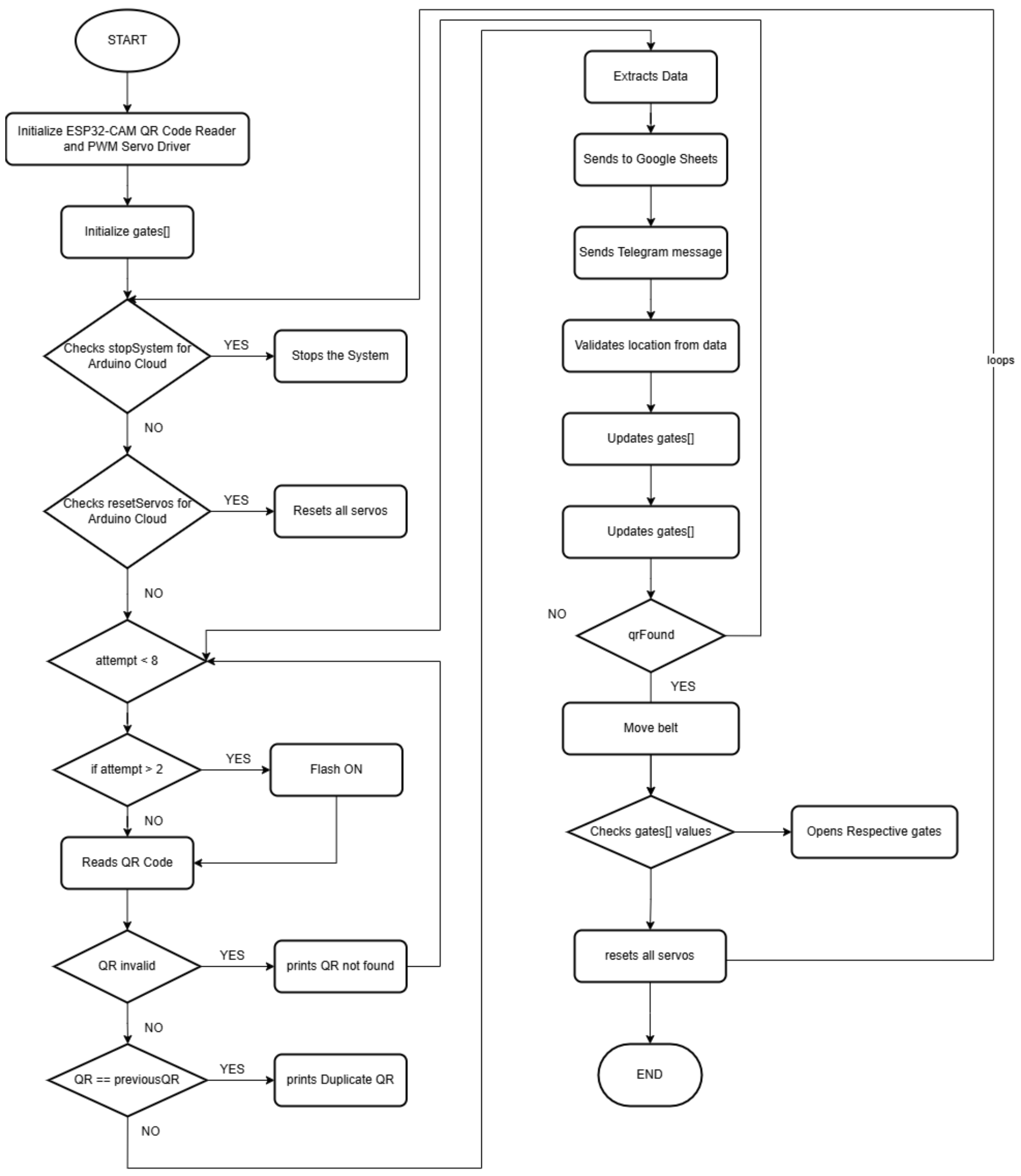
Fig 9; Gate returns to position

CHAPTER 6

PROGRAMS & USER INTERFACE

6.1. PROGRAM WORKFLOW

The following flowchart illustrates the working logic of the Smart Sorting Conveyor System controlled by the ESP32-CAM. Initially, the system initializes the camera, QR code reader, PWM servo driver, and the gates[ ] array. It then checks for cloud control flags—stopSystem pauses the system remotely, while resetServos reinitializes all servo angles. The ESP32 attempts to scan the QR code up to 8 times; if unsuccessful after 3 attempts, the camera flash is turned on. Upon successful QR detection, the data is parsed to extract the customer's name, phone number, and destination. This data is then logged to Google Sheets and sent as a Telegram message. The location is validated and mapped to the appropriate servo-controlled gate. The conveyor belt moves the parcel forward, and at the correct junction, two servos are activated—one to open the gate and the other to push the parcel. After sorting, all servos are reset, and the system is ready to process the next parcel. The flowchart effectively represents the step-by-step decision-making and cloud-integrated automation involved in the smart parcel sorting process.



6.2. ESP32 CODE USED

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

#include <WiFi.h>
#include <ESP32CameraPins.h>
#include <ESP32QRCodeReader.h>
#include <esp_camera.h>
```

```
#include <HTTPClient.h>
#include "time.h"
#include "thingProperties.h"

#define SDA_PIN 13
#define SCL_PIN 15
#define flash 4

#define CAMERA_MODEL_AI_THINKER

#if defined(CAMERA_MODEL_AI_THINKER)
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22
#endif

#define WIFI_SSID "Gagan_s_Galaxy "
#define WIFI_PASSWORD "posj8411"

const char* unauthorizedBotToken = "7693189511:AAEU1Bm8uY77IibFq_-
MJoUml70eXJqU65k";
const char* unauthorizedChatID = "1450588860";

#define CONVEYOR_PIN 12

#define SERVO_MIN 102
#define SERVO_MAX 512
```

```
ESP32QRCodeReader reader;
struct QRCodeData qrCodeData;
bool isConnected = false;

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x40);

#define SERVO1_CH 0
#define SERVO2_CH 1
#define SERVO3_CH 2
#define SERVO4_CH 3
#define SERVO5_CH 4
#define SERVO6_CH 5

int gates[6] = {-1, -1, -1, -1, -1, -1};

int secondGateJump = 1;
int thirdGateJump = 1;

String lastQR = "";  // To prevent duplicate scans

void setServoAngle(uint8_t channel, int angle) {
  int pulse = map(angle, 0, 180, SERVO_MIN, SERVO_MAX);
  pwm.setPWM(channel, 0, pulse);
}

void moveBelt() {
  digitalWrite(CONVEYOR_PIN, HIGH);
  delay(1020);
  digitalWrite(CONVEYOR_PIN, LOW);
  Serial.println("Conveyor belt moved");
  delay(3500);
  if (gates[0] != -1) gates[0] -= 1;
  if (gates[1] != -1) gates[1] -= 1;
  if (gates[2] != -1) gates[2] -= secondGateJump;
  if (gates[3] != -1) gates[3] -= secondGateJump;
  if (gates[4] != -1) gates[4] -= thirdGateJump;
  if (gates[5] != -1) gates[5] -= thirdGateJump;
}

void sendTelegramAlert(String message, bool unauthorized) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    const char* token = unauthorizedBotToken;
    const char* chatID = unauthorizedChatID;

    String url = "https://api.telegram.org/bot" + String(token) +
                 "/sendMessage?chat_id=" + String(chatID) +
                 "&text=" + urlEncode(message.c_str());

    http.begin(url);
    int httpCode = http.GET();
    if (httpCode > 0) {
      Serial.println("Telegram alert sent");
    } else {
      Serial.println("Failed to send Telegram alert");
    }
    http.end();
  } else {
    Serial.println("WiFi Disconnected");
  }
}

void resetAllServos() {
  for (int i = 0; i < 6; i++) {
    setServoAngle(i, 80);
  }
  Serial.println("All servos reset to 80 degrees");
}

String urlEncode(const char* str) {
  String encoded = "";
  char c;
  char code0, code1;
  for (int i = 0; i < strlen(str); i++) {
    c = str[i];
    if (isalnum(c)) {
      encoded += c;
    } else {
      code1 = (c & 0xf) + '0';
      if ((c & 0xf) > 9) code1 = (c & 0xf) - 10 + 'A';
      c = (c >> 4) & 0xf;
      code0 = c + '0';
      if (c > 9) code0 = c - 10 + 'A';
      encoded += '%';
      encoded += code0;
      encoded += code1;
    }
  }
  return encoded;
}

void validateLoc(String loc, int &level, int &curr_ind){
  if (loc == "Trivandrum" || loc == "Thiruvananthapuram") {
    level = 1; curr_ind = 0;
  } else if (loc == "Kollam") {
    level = 1; curr_ind = 1;
  } else if (loc == "Alappuzha") {
    level = 2; curr_ind = 2;
  } else if (loc == "Kottayam") {
    level = 2; curr_ind = 3;
  } else if (loc == "Pathanamthitta") {
    level = 3; curr_ind = 4;
  } else if (loc == "Kochi" || loc == "Ernakulam") {
    level = 3; curr_ind = 5;
  } else {
    Serial.println("Unknown Location, package requires physical analysis...");
    level = 0; curr_ind = -1;
  }
}

void openGate(uint8_t gateA, uint8_t gateB) {
  Serial.print("Moving servos: ");
  Serial.print("Servo ");
  Serial.print(gateA);
  Serial.print(" and Servo ");
  Serial.println(gateB);

  setServoAngle(gateA, 180);
  delay(500);
  setServoAngle(gateB, 0);
  delay(500);
  setServoAngle(gateB, 80);
  delay(800);
  setServoAngle(gateA, 80);
  delay(60);
}


void logToGoogleSheet(String name, String phone, String loc) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    String url = "https://script.google.com/macros/s/AKfycbyYvmE9o-
SNpot63K7RBqWr4BAqvgloI_SIrzCh6bSnE4VWI9Mp0EopuwHT6o1Lfdo/exec";
    url += "?name=" + name;
    url += "&phone=" + phone;
    url += "&location=" + loc;

    http.begin(url);
    int httpCode = http.GET();
    if (httpCode > 0) {
      Serial.println("Logged to Google Sheets successfully.");
    } else {
      Serial.println("Failed to log to Google Sheets.");
    }
    http.end();
  }
}


bool initPWMDriver() {
  Wire.beginTransmission(0x40);
  return (Wire.endTransmission() == 0);
}

void setup() {
  Serial.begin(115200);
  initProperties();
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) delay(500);
  reader.setup(CAMERA_MODEL_AI_THINKER);
  reader.begin();
  Serial.println("QR Reader initialized and ready");
  Wire.begin(SDA_PIN, SCL_PIN);
  pwm.begin();
  pwm.setPWMFreq(50);
  if (initPWMDriver()) {
    pwm.begin();
    pwm.setPWMFreq(50);
    Serial.println("PWM Servo Driver initialized successfully");
  }
  else {
    Serial.println("Failed to detect PWM Servo Driver at address 0x40");
  }
  delay(10);
  pinMode(CONVEYOR_PIN, OUTPUT);
  pinMode(flash, OUTPUT);
  delay(1000);
}

void loop() {
  ArduinoCloud.update();

  if (stopSystem) {
    Serial.println("System stopped via Arduino Cloud.");
    while (stopSystem) {
      ArduinoCloud.update();
      delay(500);
    }
  }

  if (resetServos) {
    resetAllServos();
    resetServos = false;
  }

  String qrData;
  int level = 0;
  int curr_ind = -1;
  bool qrFound = false;
  bool flashOn = false;

  for (int attempt = 0; attempt < 8; attempt++) {
    Serial.print("QR Scan Attempt: ");
    Serial.println(attempt + 1);

    if (attempt > 2 && !flashOn) {
      digitalWrite(flash, HIGH);
      Serial.println("Flash turned ON");
      flashOn = true;
    }

    if (reader.receiveQrCode(&qrCodeData, 100)) {
      if (qrCodeData.valid) {
        qrData = (const char *)qrCodeData.payload;

        if (qrData == lastQR) {
          Serial.println("Duplicate QR - Ignored");
          delay(2500);
          continue;
        }

        lastQR = qrData;
        qRData = qrData;

        Serial.println("Found QRCode");
        Serial.print("Payload: ");
        Serial.println(qrData);

        int firstComma = qrData.indexOf(',');
        int secondComma = qrData.indexOf(',', firstComma + 1);
        String name = qrData.substring(0, firstComma);
        String phone = qrData.substring(firstComma + 1, secondComma);
        String loc = qrData.substring(secondComma + 1);
        logToGoogleSheet(name, phone, loc);

        String message = "Dear " + name + ", your packaged order to " + loc + "
has arrived at our Amritapuri facility. Thank you for your order.";
        sendTelegramAlert(message, true);

        validateLoc(loc, level, curr_ind);
        Serial.print("Location extracted: ");
        Serial.println(loc);

        if (curr_ind >= 0) {
          if (gates[curr_ind] >= 0) {
            gates[curr_ind] += level;
          } else {
            gates[curr_ind] = level;
          }
        }

        if (gates[2] == 3 || gates[3] == 3){
          secondGateJump = 2;
        }
        else if ((gates[4] >= 4 && gates[4] <= 5) || (gates[5] >= 4 && gates[5]
<= 5)){
          thirdGateJump = 2;
        }
        else if (gates[4] == 6 || gates[5] ==6){
          thirdGateJump = 3;
        }

        qrFound = true;
        break;
      }
    } else {
      Serial.println("No QR found, retrying...");
    }
    delay(2500);
  }

  if (flashOn) {
    digitalWrite(flash, LOW);
    Serial.println("Flash turned OFF");
  }

  if (qrFound) {
    moveBelt();

    if (gates[0] == 0) openGate(SERVO1_CH, SERVO2_CH);
    else if (gates[1] == 0) openGate(SERVO2_CH, SERVO1_CH);
    else if ((gates[2] == 0) || (gates[2] == 1 && secondGateJump == 2)){
      openGate(SERVO3_CH, SERVO4_CH);
      secondGateJump = 1;
    }
    else if ((gates[3] == 0) || (gates[3] == 1 && secondGateJump == 2)){
      openGate(SERVO4_CH, SERVO3_CH);
      secondGateJump = 1;
    }
    else if (gates[4] == 0) openGate(SERVO5_CH, SERVO6_CH);
    else if (((gates[4] == 1) || (gates[4] == 2)) && (thirdGateJump == 2)) {
      openGate(SERVO5_CH, SERVO6_CH);
      thirdGateJump = 1;
    }
    else if ((gates[4] == 3) && (thirdGateJump == 3)) {
      openGate(SERVO5_CH, SERVO6_CH);
      thirdGateJump = 2;
    }
    else if (gates[5] == 0) openGate(SERVO6_CH, SERVO6_CH);
    else if (((gates[5] == 1) || (gates[5] == 2)) && (thirdGateJump == 2)){
      openGate(SERVO6_CH, SERVO6_CH);
      thirdGateJump = 1;
    }
    else if ((gates[5] == 3) && thirdGateJump == 3) {
      openGate(SERVO6_CH, SERVO6_CH);
      thirdGateJump = 2;
    }

    resetAllServos();

    int noQRCount = 0;
    const int maxNoQRFrames = 10;

    while (noQRCount < maxNoQRFrames) {
      if (reader.receiveQrCode(&qrCodeData, 100)) {
        if (!qrCodeData.valid || String((const char*)qrCodeData.payload) !=
lastQR) {
          noQRCount++;
        } else {
          noQRCount = 0;
        }
      } else {
```

```
    noQRCount++;
   }
   delay(250);
  }

  Serial.println("QR code no longer visible. Ready for next scan.");
 } else {
  moveBelt();
 }

 delay(500);
 Serial.println("Done!.... Moving to next loop....");
}

void onResetServosChange() {
 // Add the code here that you want to execute when onResetServosChange() is
called.
 // For example:
 Serial.println("Reset Servos Change detected!");
 // Your actual servo reset logic would go here
}

void onStopSystemChange() {
 // Add the code here that you want to execute when onStopSystemChange() is
called.
 // For example:
 Serial.println("Stop System Change detected!");
 // Your actual system stop logic would go here
}
```

7. Cloud Applications Used

The Smart Sorting Conveyor System leverages multiple cloud services to enhance functionality, enable remote access, and maintain digital records. The following are the cloud applications integrated into the project:

7.1 Arduino IoT Cloud

- **Purpose**: Enables real-time remote monitoring and control of the system.

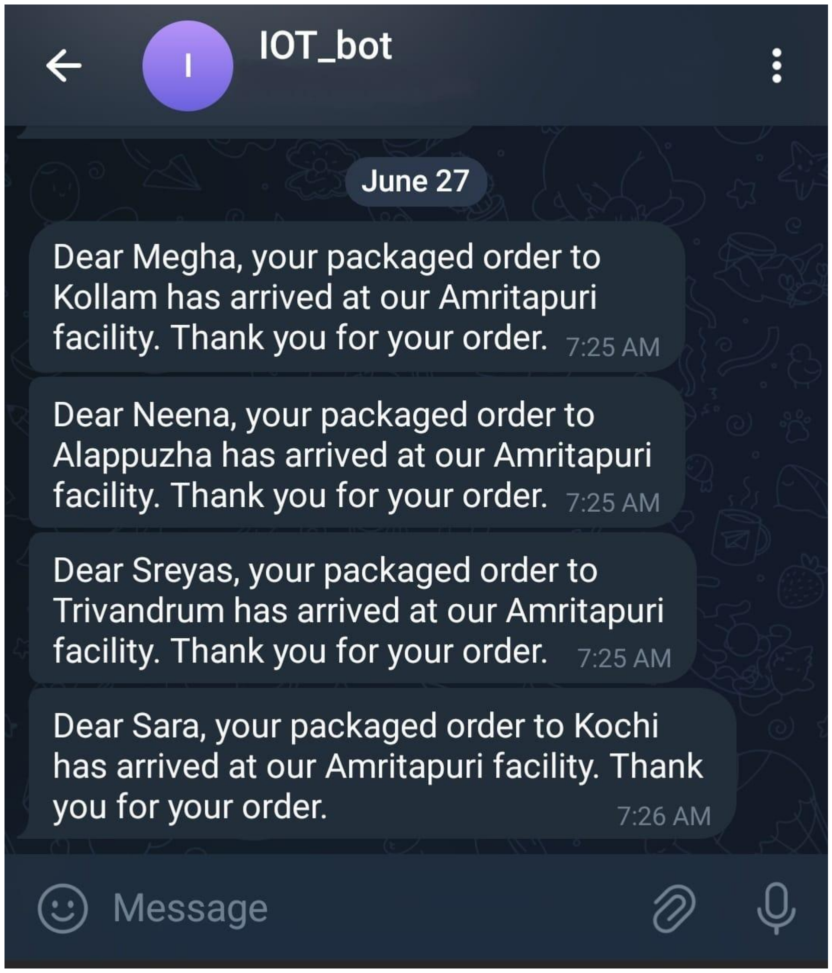
- **Features Used**:
    - **Boolean Variables** like `stopSystem` and `resetServos` for remote commands.
    - **Cloud Dashboard**: Visual interface to monitor system status and issue stop/reset actions from anywhere.
- **Benefits**:
    - Reduces the need for physical intervention.
    - Enables remote pausing of operations in case of a fault or inspection.
    - Provides a user-friendly dashboard for interaction.

7.2 Google Sheets (via Google Apps Script)

- **Purpose**: Cloud-based logging of scanned parcel data.
- **Integration Method**: An HTTP GET request is sent to a Google Apps Script Web App which logs:
    - Timestamp
    - Customer name
    - Phone number
    - Location
- **Benefits**:
    - Real-time, structured storage of parcel logs.
    - Easy access, filtering, and analysis from any device.
    - Can be extended to create dashboards using Google Data Studio.

7.3 Telegram Bot API

- **Purpose**: Sends automated notifications to customers.
- **Message Format**: `Dear <Customer Name>, your packaged order to <Location> has arrived at our Amritapuri facility. Thank you for your order.`
- **Integration Method**:
  - ESP32-CAM sends a GET request to the Telegram Bot API with the chat ID and message.
- **Benefits**:
  - Real-time alerts to customers.
  - Cost-free and reliable messaging.
  - Can be extended to alert admins as well.



Dear Megha, your packaged order to Kollam has arrived at our Amritapuri facility. Thank you for your order. 7:25 AM

Dear Neena, your packaged order to Alappuzha has arrived at our Amritapuri facility. Thank you for your order. 7:25 AM

Dear Sreyas, your packaged order to Trivandrum has arrived at our Amritapuri facility. Thank you for your order. 7:25 AM

Dear Sara, your packaged order to Kochi has arrived at our Amritapuri facility. Thank you for your order. 7:26 AM

Fig 10: Telegram BOT response

CHAPTER 7

ADVANTAGES & APPLICATIONS

7.1 ADVANTAGES

- Real-time automation of package sorting minimizes manual effort and human error.
- Cost-effective hardware components make the system affordable and scalable.
- Cloud-based control (via Arduino Cloud) allows remote system management.
- Instant customer notification through Telegram improves communication and trust.
- Flexible logic allows handling of unknown or invalid QR codes.
- Seamless integration of software and hardware ensures efficient workflow.

7.2 APPLICATIONS

- Logistics and courier centers for automatic package routing.
- E-commerce warehouses for quick and accurate parcel sorting.
- Smart post offices to enhance delivery pipeline.
- Airport baggage handling systems with QR-based destination mapping.
- University campuses or hostel package collection centers.

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

The Smart Sorting Conveyor System successfully integrates mechanical hardware and IoT technologies to deliver an efficient parcel sorting solution. Through the use of ESP32-CAM for QR scanning, servo-driven gate control, and cloud-based communication, the system addresses common challenges in manual logistics handling. The addition of Telegram alerts and Google Sheets logging further enhances system transparency and reliability.

This system showcases the potential of affordable, scalable IoT devices in transforming traditional industrial operations. It automates complex workflows, reduces manual labor, and improves traceability of packages from arrival to routing.

In the future, the system can be enhanced by incorporating:

- Live camera streaming for remote visual monitoring
- AI-based image processing to detect damaged or improperly labeled packages
- Integration with warehouse management systems (WMS) or ERP tools
- Power-efficient components to improve sustainability
- Auto-calibration and machine learning for smarter gate selection

With such enhancements, this project can be expanded for deployment in real-world logistics hubs, demonstrating how embedded systems and IoT can bring smart automation to everyday operations.

REFERENCES

[1] Robosort: Vision-Based Sorting System Using ESP32-CAM and Servo Motors. (2021).

[2] QR Code Detection and Servo Control Using ESP32-CAM. (2023).

[3] Design and Implementation of a QR Code Based Intelligent Sorting System Using Raspberry Pi. Proceedings of the 5th International Conference on Artificial Intelligence and Computer Engineering (ICAICE), IEEE. doi:10.1109/ICAICE55312.2022.00052

[4] ESP32 Camera Module Based QR Code Verification for Output On/Off. (2025).

[5] ESP32 PWM/LEDC Servo Multiplexing Guide, using Adafruit library.