Autonomous Warehouse Management Robot

By: Samarjeet Singh
(17042208145)
B.Tech E.C.E 3rd Year

Autonomous Warehouse Management Robot

By: Vasu Jindal
(17042208108)
B.Tech E.C.E 3rd Year

Acknowledgement

Date: _____

We would like to express our deepest gratitude to all those who supported us throughout the completion of our final year project, "**Autonomous Warehouse Management Robot**."

We are sincerely thankful to our internal guide, faculty members, and project coordinator for their valuable guidance, encouragement, and support at every stage of this project. Their suggestions and feedback have helped us shape this project into its present form.

We also extend our special thanks to the external guide and industry mentors for their expert advice, valuable time, and motivation.

We would like to thank the Head of the Department (HOD) and all the faculty members of our department for providing us with the necessary facilities and environment to complete our project successfully.

Finally, we would like to thank our families and friends for their constant support and encouragement during this journey.

Student Details:

Name	**Roll No.**	**Class Year Field/Section**
Samarjeet Singh	17042208145	B.Tech E.C.E 3rd Year
Vasu Jindal	17042208108	B.Tech E.C.E 3rd Year

Signatures:

Internal Guide _____

External Examiner _____

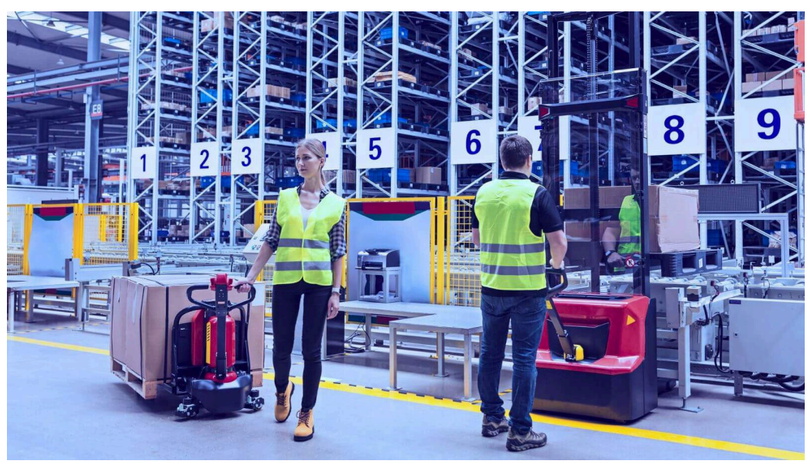
Head of Department (HOD) _____

Index

Content	Page
Traditional Warehouse Management	4-5
Autonomous Warehouse Management	6-7
About Our Project	8-9
Components Used	10-11
Functioning	12-13
Images	14
Flow Diagram	15
Schematic	16-17
Code	18-34
Future Scope	35-36

Traditional Warehouse Management

Before the introduction of autonomous robots, warehouse box management was largely manual or semi-automated, relying on human labor and simple machinery. Here's a brief breakdown of the traditional process:



1. **<u>Manual Labor</u>**
   - Staff were responsible for picking, sorting, and placing boxes.
   - Involved walking long distances, reading labels manually, and physically handling goods.
   - Mistakes in inventory count, misplacement, and damage were common.
2. **<u>Forklifts and Trolleys</u>**
   - For heavy loads, workers used manual or powered forklifts, pallet jacks, or trolleys.
   - Still required a human driver/operator.
   - Risk of injury or collision was present.
3. **<u>Paper-Based Inventory Systems</u>**
   - Inventory tracking was mostly done on paper or simple spreadsheets.

Prone to data errors, time-consuming, and hard to update in real-time.

4. **Barcode Scanning**
    - Some modern warehouses used barcode or QR code scanning, but still relied on manual scanning by workers using handheld devices.
5. **Fixed Conveyor Belts**
    - In semi-automated settings, conveyor belts moved boxes from one point to another.
    - These were stationary systems, not flexible or mobile like today's robots.
    - Layout changes were difficult and expensive.

Limitations of Traditional Warehouse Box Management

- **Inefficiency**: Time-consuming and slow operations.
- **Labor Intensive**: Required a large number of workers.
- **Human Error**: Prone to misplaced items and inventory mistakes.
- **Safety Hazards**: Risk of injury from lifting heavy loads or operating forklifts.
- **Limited Scalability**: Hard to scale with increasing product demand.

Autonomous Warehouse Management

Autonomous warehouse robots (like those used by Amazon, Walmart, etc.) have completely revolutionized warehouse operations by handling tasks that were previously manual, slow, and error-prone. Here's how they make life easier:

1. **<u>Increased Efficiency and Speed</u>**
   - Robots can move continuously without breaks or fatigue.
   - They can optimize travel paths and reduce the distance traveled inside the warehouse.
   - **Result**: Faster picking, packing, sorting, and storing compared to human workers.

2. **<u>Minimized Human Errors</u>**
   - Robots use advanced sensors (like LiDAR, cameras, RFID readers) to precisely identify boxes and navigate the warehouse.
   - Inventory management is automated — real-time updating of item locations and counts.
   - **Result**: Fewer misplaced boxes, better order accuracy, and reduced customer complaints.

3. **Enhanced Safety**
   - Robots handle heavy lifting and movement, reducing workplace injuries caused by lifting or forklift accidents.
   - They have built-in obstacle detection, collision avoidance, and emergency stop systems.
   - **Result**: Safer working environments for human workers.

6. **Real-Time Monitoring and Smart Inventory Management**
   - Robots integrate with Warehouse Management Systems (WMS).
   - They provide live inventory tracking, dynamic shelf management, and automated restocking alerts.
   - **Result**: Better inventory control and faster decision-making.

7. **Adaptability and Scalability**
   - Need more capacity? Just add more robots — no need to redesign the whole warehouse.
   - Robots can adapt to seasonal peaks (like holidays) without requiring mass hiring.
   - **Result**: Flexible and scalable warehouse operations.

About Our Project

Project Idea (Overview)

The goal of this project is to develop an intelligent mobile robot capable of automating warehouse operations such as transporting boxes or items from one location to another within a warehouse environment autonomously with minimum human involment.

With the increasing demand for faster and more efficient inventory handling, this robot is designed to reduce human labor, minimize errors, and enhance operational speed by navigating autonomously through the warehouse, identifying boxes, and delivering them to the required locations.

Functionality

Autonomous Navigation

- The robot can move freely inside the warehouse by detecting its environment using onboard sensors and programmed logic.
- It avoids obstacles, maps its path, and reaches its target location without external guidance.

Inventory Movement

The robot transports the picked item from the source to the destination (like a packing station, storage shelf, or delivery bay).

Task Execution Based on Commands

- The robot can receive instructions wirelessly or work in a predefined pattern based on the warehouse layout.

Error Handling and Alerts

- In case of a blocked path or incorrect placement, the robot can raise alerts or wait for user input, making it reliable in real-world use.

Components Used

1. **Lithium Cells (x7 Cells)**
   - Act as the primary power source for the robot.
   - They provide the required voltage and current to operate motors, controllers, and sensors reliably for longer periods.
2. **150 RPM DC Metal Geared Motors (x2 Units)**
   - High-torque motors are used for the movement of the robot.
   - 150 RPM speed ensures a balance between speed and torque, which is suitable for carrying loads inside a warehouse.
3. **Servo Motor (for Steering)**
   - A servo motor is used to steer the front wheels or control a steering mechanism.
   - Allows precise control over the direction of the robot during navigation.
4. **ESP32-CAM Module**
   - The ESP32-CAM acts as the main brain aBehind the color detection and steering.
   - It captures real-time images and can perform simple processing tasks like object recognition or path following if needed.
5. **ESP8266 Module**
   - The ESP8266 is connected to the ESP32-CAM using ESP-NOW wireless protocol.

It handles cthe functioning of dc motors via L298n and also the RFID Module.

Components Used

6. **Color Sensor (TCS34725)**
   - The TCS color sensor is used to detect surface colors (e.g., floor markings or colored boxes).
   - It helps the robot make decisions based on color signals, like stopping, changing direction, etc.
7. **Voltage Regulator**
   - Regulates and steps down voltage to levels suitable for sensitive components like the ESP32, ESP8266, and sensors.
   - Ensures stable and safe voltage supply to avoid damage.In this case, it provides output of 5v.
8. **Buzzer**
   - The buzzer is used to provide audio alerts for different events like successful RFID scan, error in path, or completion of task.
   - It enhances user feedback during operation.
9. **L298N Motor Driver Module**
   - Controls the speed and direction of the two DC motors.
   - Allows forward, reverse, movement commands based on ESP32 inputs.
10. **Battery Management System (BMS)**
    - Protects the lithium cells by managing charging, discharging, and balancing.
    - Prevents overcharge, deep discharge, and ensures long battery life and safety.
11. **RFID Module**
    - Used for identifying boxes inside the warehouse.

Functioning

Startup Phase

- Upon powering up, the robot first initializes the RFID module.
- Workers scan the RFID tags attached to each cargo or box.
- ESP8266 processes the scanned data to plot a delivery map showing where each piece of cargo needs to be dropped off inside the warehouse.

Cargo Management and Data Handling

- The RFID system also creates and maintains an Excel file that logs critical information about each cargo:
  - Source location (where it was picked up from)
  - Destination location (where it should be dropped)
  - Cargo type (e.g., fast delivery, standard delivery, fragile items)
- This structured cargo data ensures organized transportation and proper prioritization during delivery.

Movement and Navigation

- After completing the scanning and mapping process, the robot waits for a "start" command to begin its journey.
- Once started, the robot moves autonomously, navigating based on color codes marked on the warehouse floor:
  - Green Color → Turn Right
  - Yellow Color → Turn Left
  - Red Color → Stop immediately

    - Blue Color → Wait for user command (pause and await further instruction)

- The ESP32-CAM acts as the main controller for movement decisions, interpreting color inputs while also providing a live video feed of the robot's surroundings for monitoring purposes.

Cargo Drop-off

- When the robot reaches its assigned destination, it stops automatically.
- Warehouse personnel can pick up the designated cargo from the robot at the stop point.
- After successful unloading, the robot resumes its journey towards the next drop-off point or returns to the main station if all deliveries are completed.

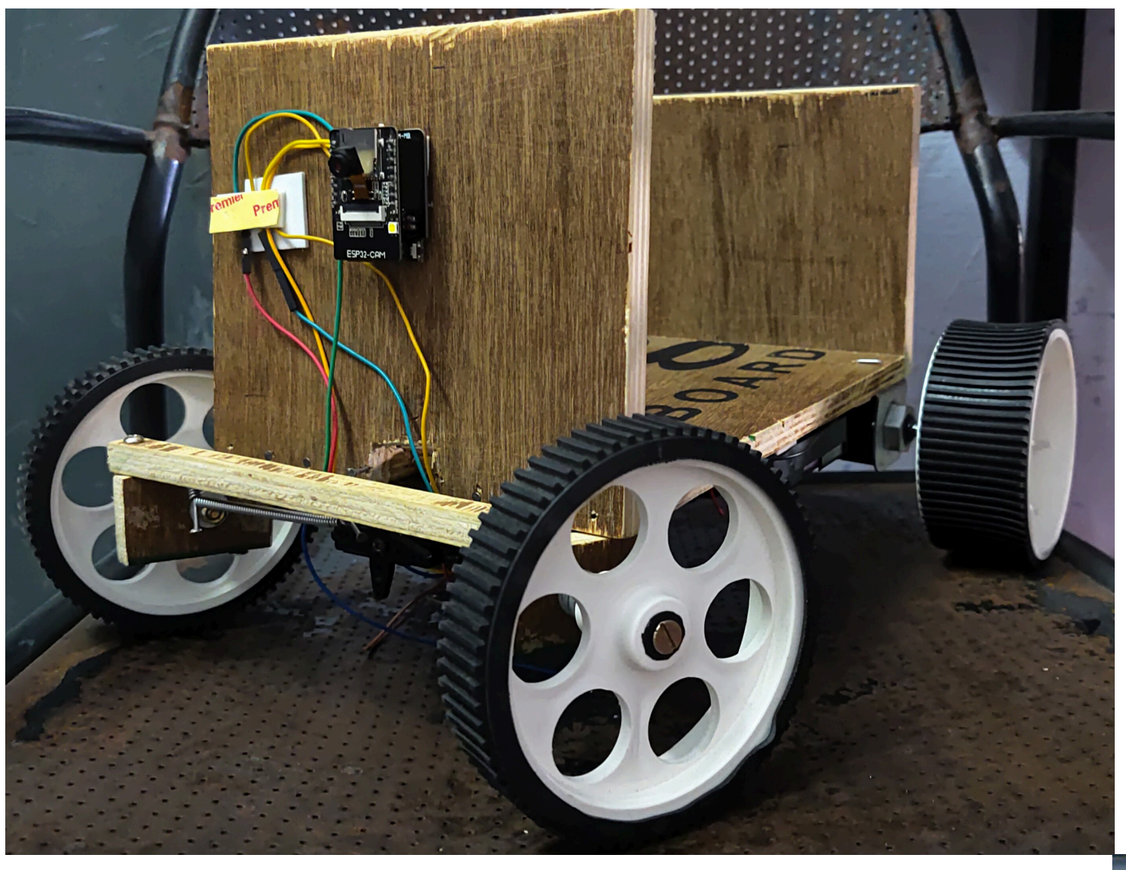
Return to Main Station

- After completing all assigned tasks, the robot navigates back to its main station.
- It prepares itself for the next operational cycle or shift.

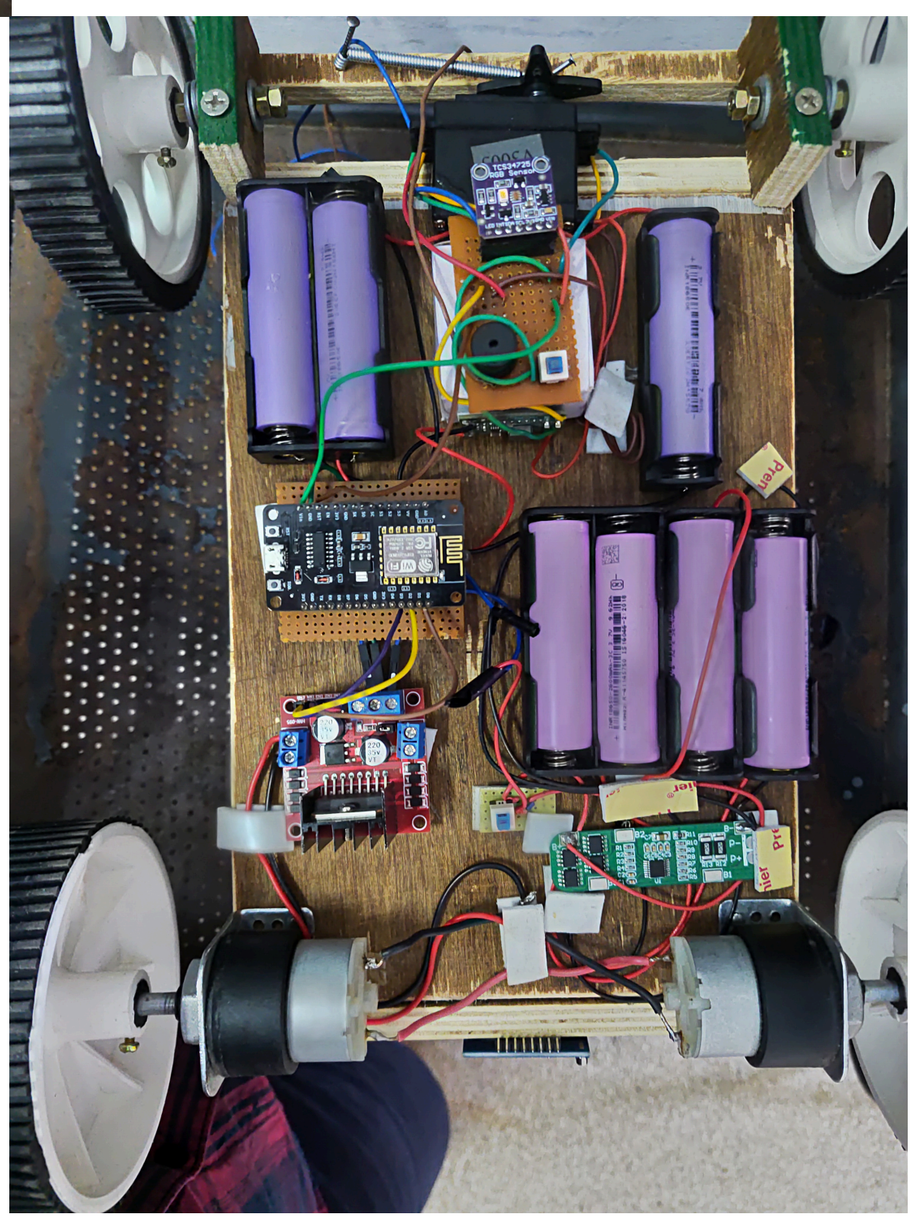
Manual Mode (Optional)

- In case of special situations or if manual intervention is needed, the robot can be switched to manual control mode.
- This provides flexibility to override autonomous functions and allows a user to drive or control the robot manually.

Images



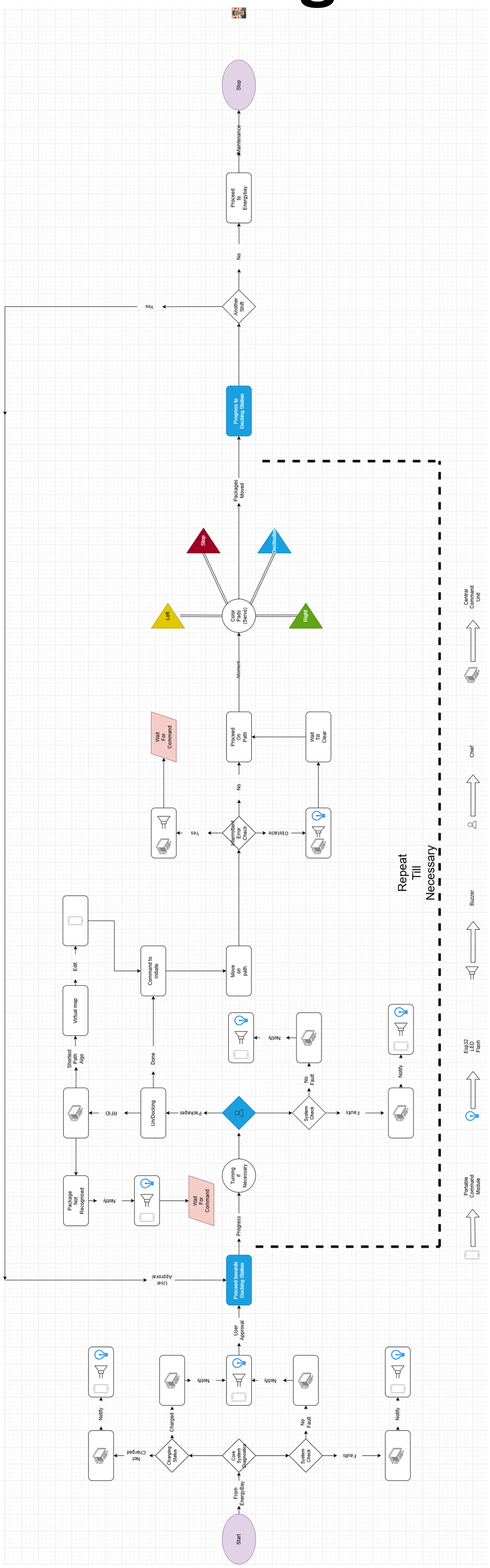
Front



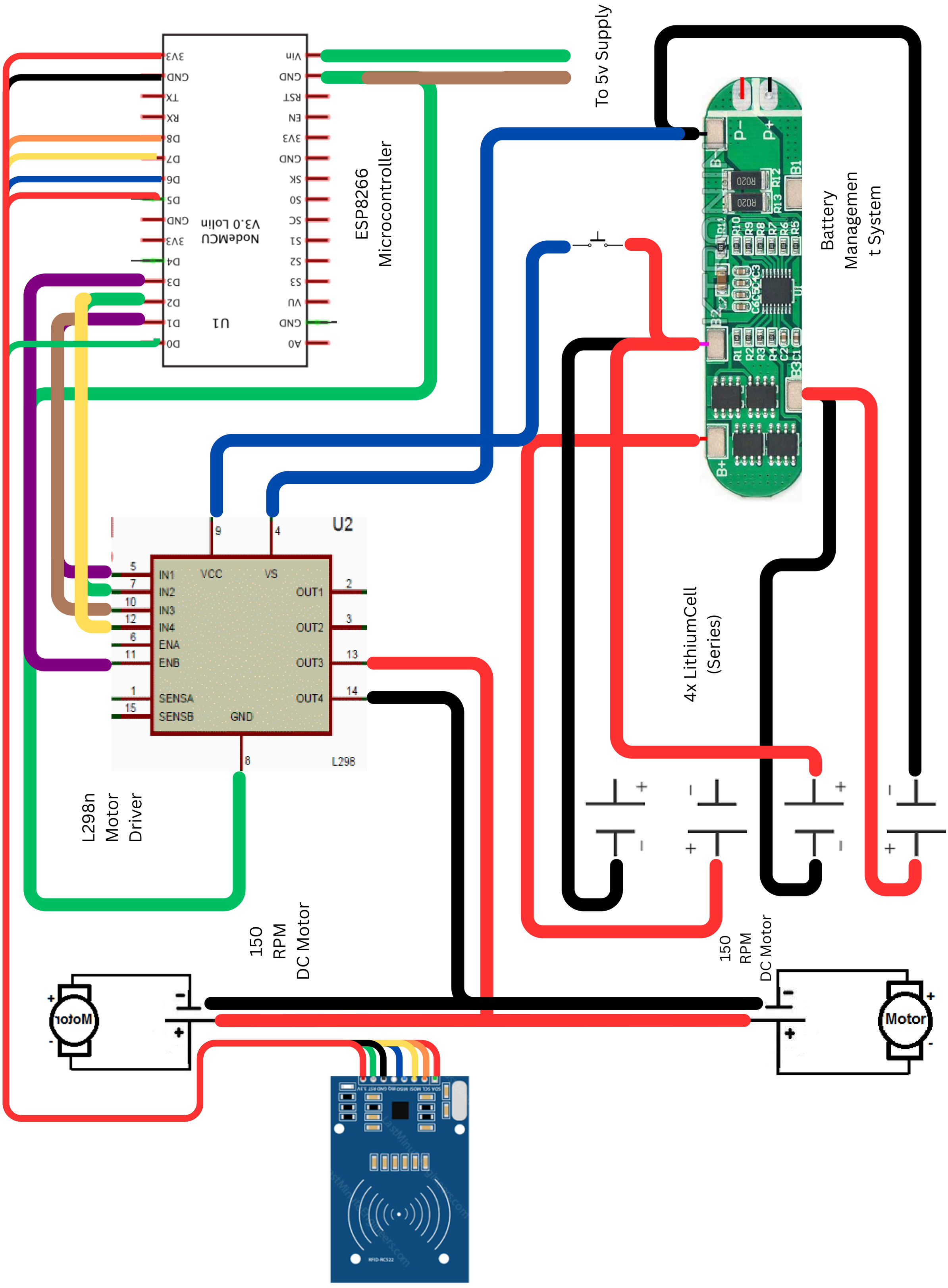
Bottom

Back

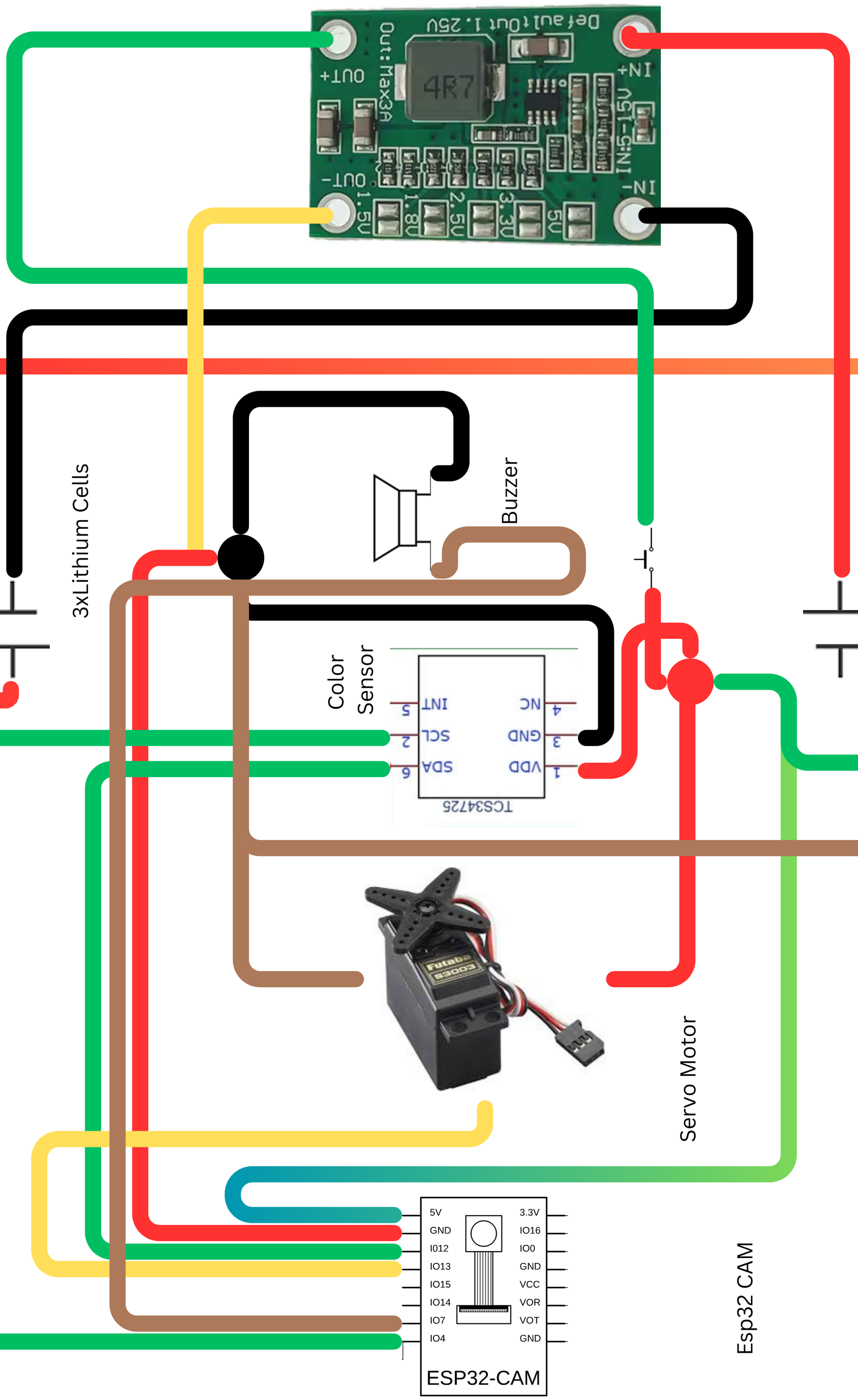
Flow Diagram



Schematic-1


To 5v Supply
ESP8266 Microcontroller
NodeMCU V3.0 Lolin
U1
Vin
GND
RST
EN
3V3
GND
SK
SO
SC
S1
S2
S3
VU
GND
A0
3V3
GND
TX
RX
D8
D7
D6
D5
GND
3V3
D4
D3
D2
D1
D0
Battery Managemen t System
P-
P+
B-
B1
B2
B3
B+
R020
R020
R13
R12
R10
R9
R8
R7
R6
R5
C6
C5
C4
C3
R1
R2
R3
R4
C2
C1
4x LithiumCell (Series)
U2
IN1
IN2
IN3
IN4
ENA
ENB
SENSA
SENSB
VCC
VS
OUT1
OUT2
OUT3
OUT4
GND
L298
L298n Motor Driver
150 RPM DC Motor
Motor
Motor
RFID-RC522

Schematic-2


Default Out 1.25V
Out:Max3A
OUT+
OUT-
4R7
IN+
IN-
5V-15V
1.5V
1.8V
2.5V
3.3V
5V
Voltage Regulator
3xLithium Cells
Buzzer
Color Sensor
INT
SCL
SDA
NC
GND
VDD
TCS34725
Towards ESP8266
Servo Motor
Futaba
S3003
5V
GND
I012
IO13
IO15
IO14
IO7
IO4
3.3V
IO16
IO0
GND
VCC
VOR
VOT
GND
ESP32-CAM
Esp32 CAM

Code
ESP32 CAM

```
#include <WiFi.h>
#include <esp_now.h>
#include <Wire.h>
#include "Adafruit_TCS34725.h"
#include <ESP32Servo.h>
#include <HTTPClient.h>
#include "esp_camera.h"
#include "esp_http_server.h"
#include <base64.h>

// ---------- CAMERA PINS ----------
#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

// ---------- CONFIGURATIONS ----------

// Your WiFi credentials
const char* ssid = "Redmi12";
const char* password = "12345678";

// Your laptop IP address running the Flask app
const char* flaskServer = "http:// 192.168.150.199:5000";  // Update this to your IP

// MAC address of ESP8266
uint8_t receiverMAC[] = {0xF4, 0xCF, 0xA2, 0x4D, 0xE5, 0x11};  // Update this

// I2C Pins for ESP32-CAM
#define SDA_PIN 14
#define SCL_PIN 13

// Servo pin
const int servoPin = 15;

// Buzzer pin
const int buzzerPin = 2;  // Change this to your actual buzzer pin
```

```
// Global states
String currentMode = ""; // Start with no mode selected
bool emergencyStop = false;
String previousMode = ""; // To track mode changes
bool cameraStreamActive = true;
unsigned long lastImageSend = 0;
const unsigned long IMAGE_INTERVAL = 1000; // Send image every 1 second

// Objects
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);
Servo myServo;

// ---------- CAMERA SETUP ----------
bool setupCamera() {
 camera_config_t config;
 config.ledc_channel = LEDC_CHANNEL_0;
 config.ledc_timer = LEDC_TIMER_0;
 config.pin_d0 = Y2_GPIO_NUM;
 config.pin_d1 = Y3_GPIO_NUM;
 config.pin_d2 = Y4_GPIO_NUM;
 config.pin_d3 = Y5_GPIO_NUM;
 config.pin_d4 = Y6_GPIO_NUM;
 config.pin_d5 = Y7_GPIO_NUM;
 config.pin_d6 = Y8_GPIO_NUM;
 config.pin_d7 = Y9_GPIO_NUM;
 config.pin_xclk = XCLK_GPIO_NUM;
 config.pin_pclk = PCLK_GPIO_NUM;
 config.pin_vsync = VSYNC_GPIO_NUM;
 config.pin_href = HREF_GPIO_NUM;
 config.pin_sscb_sda = SIOD_GPIO_NUM;
 config.pin_sscb_scl = SIOC_GPIO_NUM;
 config.pin_pwdn = PWDN_GPIO_NUM;
 config.pin_reset = RESET_GPIO_NUM;
 config.xclk_freq_hz = 20000000;
 config.pixel_format = PIXFORMAT_JPEG;

 // Initialize with lower quality but faster
 config.frame_size = FRAMESIZE_VGA;
 config.jpeg_quality = 12; // 0-63, lower means higher quality
 config.fb_count = 2;

 // Camera init
 esp_err_t err = esp_camera_init(&config);
 if (err != ESP_OK) {
 Serial.printf("Camera init failed with error 0x%x", err);
 return false;
 }

 sensor_t * s = esp_camera_sensor_get();
 s->set_brightness(s, 1); // Increase brightness
 s->set_contrast(s, 1); // Increase contrast
```

```
 return true;
}
// Function to send camera frame to Flask server
void sendCameraFrame() {
 if (!cameraStreamActive || millis() - lastImageSend < IMAGE_INTERVAL) {
 return;
 }

 camera_fb_t * fb = esp_camera_fb_get();
 if (!fb) {
 Serial.println("Camera capture failed");
 return;
 }

 HTTPClient http;
 http.begin(String(flaskServer) + "/upload_image");
 http.addHeader("Content-Type", "application/octet-stream");

 // Send the image data
 int httpResponseCode = http.POST(fb->buf, fb->len);

 if (httpResponseCode > 0) {
 Serial.printf("Image uploaded, code: %d\n", httpResponseCode);
 } else {
 Serial.printf("Image upload error: %s\n", http.errorToString(httpResponseCode).c_str());
 }

 http.end();
 esp_camera_fb_return(fb);
 lastImageSend = millis();
}
// ---------- BUZZER FUNCTIONS ----------
void beep(int times) {
 for (int i = 0; i < times; i++) {
 digitalWrite(buzzerPin, HIGH);
 delay(200);
 digitalWrite(buzzerPin, LOW);
 if (i < times - 1) {
 delay(200); // Pause between beeps
 }
 }
}
// ---------- ESP-NOW SEND ----------
void sendCommand(uint8_t cmd) {
 if (emergencyStop) {
 uint8_t emergencyData[] = {0}; // Create a regular array for emergency stop
 esp_now_send(receiverMAC, emergencyData, sizeof(emergencyData));
 Serial.println("EMERGENCY: Forcing STOP");
 return;
 }
 esp_now_send(receiverMAC, &cmd, 1);
 Serial.printf("Sent command %d\n", cmd);
}
```

```
// ---------- MODE + COMMAND FETCH ----------
void fetchModeAndCommand() {
 HTTPClient http;
 http.begin(String(flaskServer) + "/status");
 int code = http.GET();
 if (code == 200) {
 String payload = http.getString();
 previousMode = currentMode; // Save previous mode before updating

 if (payload.indexOf("emergency") >= 0) {
emergencyStop = true;
sendCommand(0);
} else if (payload.indexOf("manual") >= 0) {
currentMode = "MANUAL";
if (previousMode != "MANUAL") {
beep(2); // Beep twice for manual mode
Serial.println("Switched to MANUAL mode");
}
} else if (payload.indexOf("auto") >= 0) {
currentMode = "AUTO";
if (previousMode != "AUTO") {
beep(3); // Beep three times for auto mode
Serial.println("Switched to AUTO mode");
}
}
 }
 http.end();
}
void fetchManualCommand() {
 HTTPClient http;
 http.begin(String(flaskServer) + "/manual_command");
 int code = http.GET();
 if (code == 200) {
 String cmd = http.getString();
 if (cmd == "start") sendCommand(1);
 else if (cmd == "stop") sendCommand(0);
 else if (cmd == "left") {
 myServo.write(0);
 delay(1000);
 myServo.write(90);
 }
 else if (cmd == "right") {
 myServo.write(180);
 delay(1000);
 myServo.write(90);
 }
 }
 http.end();
}
// ---------- COLOR DETECTION ----------
void handleColorDetection() {
 uint16_t clear, red, green, blue;
 tcs.getRawData(&red, &green, &blue, &clear);
 // Improved color detection based on raw values
 // Using the data you provided
```

```
// Check for RED
if (red > 85 && green < 50 && blue < 50) {
Serial.println("Red detected");
sendCommand(0);
}
// Check for BLUE
else if (blue > 75 && red < 75 && green < 85 && blue > red) {
Serial.println("Blue detected");
// No longer waiting for approval
sendCommand(0);
}
// Check for GREEN
else if (green > 50 && red < 60 && blue < 50 && green > red && green > blue) {
Serial.println("Green detected — turning RIGHT");
sendCommand(0);
myServo.write(180);
delay(1500);
myServo.write(90);
sendCommand(1);
}
// Check for YELLOW
else if (red > 120 && green > 90 && blue < 65 && red > blue && green > blue) {
Serial.println("Yellow detected — turning LEFT");
sendCommand(0);
myServo.write(0);
 delay(1500);
 myServo.write(90);
 sendCommand(1);
 }
 else {
 Serial.println("Unknown color - no action");
 // Don't send any command for unknown colors
 }
}
// ---------- SETUP ----------
void setup() {
 Serial.begin(115200);

 // Setup buzzer pin
 pinMode(buzzerPin, OUTPUT);
 digitalWrite(buzzerPin, LOW);

 myServo.attach(servoPin);
 myServo.write(90); // center
 // Initialize camera
 if (!setupCamera()) {
 Serial.println("Camera setup failed!");
 } else {
 Serial.println("Camera initialized successfully");
 }
 Wire.begin(SDA_PIN, SCL_PIN);
 if (!tcs.begin()) {
 Serial.println("Color sensor not found");
 while (1);
 }
```

```
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
Serial.println("Connecting to WiFi...");
// Wait indefinitely for WiFi connection
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}

Serial.println("\nWiFi connected!");
beep(1); // One beep when WiFi is connected
Serial.print("IP Address: ");
Serial.println(WiFi.localIP());
if (esp_now_init() != ESP_OK) {
Serial.println("ESP-NOW init failed");
return;
}
esp_now_peer_info_t peerInfo = {};
memcpy(peerInfo.peer_addr, receiverMAC, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;
if (esp_now_add_peer(&peerInfo) != ESP_OK) {
Serial.println("Failed to add peer");
return;
}
 Serial.println("Setup complete, waiting for mode selection...");

 // Wait for user to select a mode
 while (currentMode == "") {
 fetchModeAndCommand();
 delay(1000);
 }
}
// ---------- LOOP ----------
void loop() {
 bool wifiOk = WiFi.status() == WL_CONNECTED;
 if (wifiOk) {
 fetchModeAndCommand();

 // Send camera frame regardless of mode
 sendCameraFrame();

if (emergencyStop) {
delay(1000);
return;
}
if (currentMode == "AUTO") {
handleColorDetection();
} else if (currentMode == "MANUAL") {
fetchManualCommand();
}
} else {
// Try to reconnect WiFi
Serial.println("WiFi disconnected. Attempting to reconnect...");
WiFi.begin(ssid, password);
```

```
// Wait for reconnection
unsigned long reconnectStart = millis();
while (WiFi.status() != WL_CONNECTED && millis() - reconnectStart < 10000) {
delay(500);
Serial.print(".");
}

if (WiFi.status() == WL_CONNECTED) {
Serial.println("\nWiFi reconnected!");
beep(1);
 }
 }
 delay(2000);
}
```

Code
ESP8266

```
#include <ESP8266WiFi.h>
#include <SPI.h>
#include <MFRC522.h>
extern "C" {
  #include <espnow.h>
}

// L298N motor driver pins (corrected to IN3 and IN4)
#define MOTOR_IN3 D1
#define MOTOR_IN4 D2
#define MOTOR_EN  D3

// RFID connections
#define RFID_POWER_PIN D5   // Controls power to RFID reader (optional)
#define RST_PIN D0          // Reset pin for RC522
#define SS_PIN D8           // SDA pin for RC522

// Create MFRC522 instance
MFRC522 mfrc522(SS_PIN, RST_PIN);

// Flags for RFID scanning
bool rfidActive = false;
bool rfidDataCaptured = false;
String capturedUID = "";

void startMotor() {
  digitalWrite(MOTOR_IN3, HIGH);
  digitalWrite(MOTOR_IN4, LOW);
  analogWrite(MOTOR_EN, 1023);  // Full speed
  digitalWrite(RFID_POWER_PIN, HIGH); // Power ON RFID
  Serial.println("Motor started");
}

void stopMotor() {
  digitalWrite(MOTOR_IN3, LOW);
  digitalWrite(MOTOR_IN4, LOW);
  analogWrite(MOTOR_EN, 0);      // Motor OFF
  digitalWrite(RFID_POWER_PIN, LOW); // Power OFF RFID
  Serial.println("Motor stopped");
}

void receiveCallback(uint8_t *mac, uint8_t *incomingData, uint8_t len) {
  if (len != 1) {
    Serial.println("Invalid data length");
    return;
  }

  uint8_t command = incomingData[0];
  Serial.printf("Received command: %d\n", command);
```

```
switch (command) {
case 0:
stopMotor();
break;
case 1:
startMotor();
break;
case 2:
Serial.println("Unknown color - maintaining current state");
break;
default:
 Serial.printf("Unhandled command: %d\n", command);
 }
}
void setup() {
 Serial.begin(115200);
 WiFi.mode(WIFI_STA);
 WiFi.disconnect();
 if (esp_now_init() != 0) {
 Serial.println("ESP-NOW init failed");
 return;
 }
 esp_now_set_self_role(ESP_NOW_ROLE_SLAVE);
 esp_now_register_recv_cb(receiveCallback);
 // Motor Pins
 pinMode(MOTOR_IN3, OUTPUT);
 pinMode(MOTOR_IN4, OUTPUT);
 pinMode(MOTOR_EN, OUTPUT);
 pinMode(RFID_POWER_PIN, OUTPUT);
 stopMotor(); // Default: motor stopped, RFID off
 // RFID Init
 digitalWrite(RFID_POWER_PIN, HIGH);
 delay(100); // Let RFID power stabilize
 SPI.begin();
 mfrc522.PCD_Init();
 Serial.println("RFID reader initialized");
 rfidActive = true;
}
void loop() {
 if (rfidActive && !rfidDataCaptured) {
 if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
// Capture UID
capturedUID = "";
for (byte i = 0; i < mfrc522.uid.size; i++) {
capturedUID += String(mfrc522.uid.uidByte[i], HEX);
}
capturedUID.toUpperCase();
// Sample Data (you can change dynamically later)
String source = "Warehouse_A";
String destination = "Dock_3";
String parcelType = "Electronics";
String timestamp = "2025-04-26 15:00"; // (Fixed timestamp for now)
```

```
Serial.println("\n--- RFID Tag Scanned ---");
 Serial.println("Captured UID: " + capturedUID);
 Serial.println("Source: " + source);
 Serial.println("Destination: " + destination);
 Serial.println("Parcel Type: " + parcelType);
 Serial.println("Timestamp: " + timestamp);
 Serial.println("-------------------------");
 // After first scan, power off RFID
 rfidDataCaptured = true;
 rfidActive = false;
 digitalWrite(RFID_POWER_PIN, LOW); // Save power
 }
 }
```

Code
WebApp(Backend)

```
# flask_robot_control.py
from flask import Flask, jsonify, request, render_template, Response, send_file
import os
import io
from datetime import datetime
import base64
from PIL import Image
import threading
import time

app = Flask(__name__)

# Global states
mode = "AUTO"       # or "MANUAL"
manual_command = "stop"  # start/stop/left/right
emergency = False
pause_requested = False

# Camera related variables
latest_image = None
latest_image_time = None
image_lock = threading.Lock()

@app.route("/")
def home():
                return render_template("index.html", mode=mode, command=manual_command,
emergency=emergency)

@app.route("/status")
def get_status():
    global mode, emergency, pause_requested
    if emergency:
        return "emergency"
    if pause_requested:
        return "pause"
    return mode.lower()

@app.route("/manual_command")
def get_manual_command():
    global manual_command
    return manual_command

@app.route("/set_mode", methods=["POST"])
def set_mode():
    global mode
    mode = request.form.get("mode", "AUTO")
    return ("", 204)
```

```
@app.route("/set_command", methods=["POST"])
def set_command():
 global manual_command
 manual_command = request.form.get("command", "stop")
 return ("", 204)
@app.route("/emergency", methods=["POST"])
def toggle_emergency():
 global emergency
 emergency = not emergency
 return ("", 204)
@app.route("/pause_notice")
def pause_notice():
 global pause_requested
 pause_requested = True
 return ("Pause acknowledged", 200)
@app.route("/approve_resume", methods=["POST"])
def approve_resume():
 global pause_requested
 pause_requested = False
 return ("", 204)
@app.route("/upload_image", methods=["POST"])
def upload_image():
 global latest_image, latest_image_time

 # Get the raw image data from the request
 image_data = request.data

 # Save the image with timestamp
 with image_lock:
 latest_image = image_data
 latest_image_time = datetime.now()

 return "Image received", 200
@app.route("/camera_feed")
def camera_feed():
 global latest_image

 if latest_image is None:
 # Return a placeholder image or message
 return "No image available", 404

 with image_lock:
 # Return the latest image
 return Response(latest_image, mimetype="image/jpeg")
@app.route("/camera_status")
def camera_status():
 global latest_image_time

 if latest_image_time is None:
 return jsonify({
 "status": "No image received",
 "last_update": None
 })
```

```
 time_diff = (datetime.now() - latest_image_time).total_seconds()

 return jsonify({
 "status": "Active" if time_diff < 5 else "Inactive",
 "last_update": latest_image_time.strftime("%H:%M:%S"),
 "seconds_ago": round(time_diff)
 })
@app.route("/video_feed")
def video_feed():
 def generate():
 global latest_image
 while True:
 with image_lock:
 if latest_image is not None:
 yield (b'--frame\r\n'
 b'Content-Type: image/jpeg\r\n\r\n' + latest_image + b'\r\n')
 time.sleep(0.1) # Adjust based on desired frame rate

 return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')
if __name__ == "__main__":
 app.run(host="0.0.0.0", port=5000, debug=False, threaded=True)
```

Code
WebApp(Frontend)

html code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Robot Control Panel</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    body {
        font-family: Arial, sans-serif;
        max-width: 1200px;
        margin: 0 auto;
        padding: 20px;
    }
    .control-panel {
        display: flex;
        flex-wrap: wrap;
        gap: 20px;
    }
    .section {
        flex: 1;
        min-width: 300px;
        background-color: #f5f5f5;
        padding: 15px;
        border-radius: 8px;
        box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}
.button-group {
    display: flex;
    gap: 10px;
    margin: 10px 0;
}
button {
    padding: 10px 15px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-weight: bold;
}
.primary {
    background-color: #4CAF50;
    color: white;
}
.secondary {
    background-color: #2196F3;
    color: white;
}
.
```

```
danger {
 background-color: #f44336;
 color: white;
 }
 .warning {
 background-color: #ff9800;
 color: white;
 }
 .neutral {
 background-color: #9e9e9e;
 color: white;
 }
 h1, h2 {
 color: #333;
 }
 .active {
 border: 3px solid #4CAF50;
 }
 .camera-feed {
 width: 100%;
 max-height: 360px;
 object-fit: contain;
 background-color: #000;
 border-radius: 4px;
 }
 .status {
 margin-top: 10px;
 font-size: 14px;
 color: #666;
 }
 .controls {
 display: grid;
 grid-template-columns: 1fr 1fr 1fr;
 gap: 10px;
 margin-top: 15px;
 }
 .controls button {
 padding: 15px;
 font-size: 16px;
 }
 .center-col {
 grid-column: 2;
 }
 </style>
</head>
<body>
 <h1>Robot Control Panel</h1>

 <div class="control-panel">
 <div class="section">
 <h2>Camera Feed</h2>
 <img src="/video_feed" class="camera-feed" alt="Camera Feed">
 <div class="status" id="camera-status">Camera status: checking...</div>
 </div>
```

```
<div class="section">
<h2>Operation Mode</h2>
<div class="button-group">
  <button id="auto-btn" class="secondary {% if mode=='AUTO' %}active{% endif %}" onclick="setMode('AUTO')">AUTO</button>
  <button id="manual-btn" class="secondary {% if mode=='MANUAL' %}active{% endif %}" onclick="setMode('MANUAL')">MANUAL</button>
</div>

<h2>Manual Controls</h2>
<div class="controls">
<div></div>
<button class="primary center-col" onclick="sendCommand('start')">▲ FORWARD</button>
<div></div>
<button class="warning" onclick="sendCommand('left')">◀ LEFT</button>
<button class="danger" onclick="sendCommand('stop')">◻ STOP</button>
<button class="warning" onclick="sendCommand('right')">RIGHT ▶</button>
</div>

<h2>System Control</h2>
<div class="button-group">
<button id="emergency-btn" class="danger" onclick="toggleEmergency()">
{% if emergency %}RESET EMERGENCY{% else %}EMERGENCY STOP{% endif %}
</button>
{% if pause_requested %}
    <button id="approve-btn" class="primary" onclick="approveResume()">APPROVE RESUME</button>
 {% endif %}
 </div>
 </div>
 </div>

 <script>
 function setMode(newMode) {
 fetch('/set_mode', {
 method: 'POST',
 headers: {'Content-Type': 'application/x-www-form-urlencoded'},
 body: mode=${newMode}
 }).then(() => {
 document.getElementById('auto-btn').classList.toggle('active', newMode === 'AUTO');
 document.getElementById('manual-btn').classList.toggle('active', newMode === 'MANUAL');
 });
 }

 function sendCommand(cmd) {
 fetch('/set_command', {
 method: 'POST',
 headers: {'Content-Type': 'application/x-www-form-urlencoded'},
 body: command=${cmd}
 });
 }
```

```
function toggleEmergency() {
fetch('/emergency', {
method: 'POST'
}).then(() => {
window.location.reload();
});
}

function approveResume() {
fetch('/approve_resume', {
method: 'POST'
}).then(() => {
window.location.reload();
});
}

// Check camera status periodically
setInterval(() => {
fetch('/camera_status')
.then(response => response.json())
.then(data => {
let statusText = Camera status: ${data.status};
if (data.last_update) {
statusText += ` (last update: ${data.last_update}, ${data.seconds_ago}s ago)`;
}
document.getElementById('camera-status').innerText = statusText;
});
}, 2000);
</script>
</body>
</html>
```

Future scope

The "**Autonomous Warehouse Management Robot**" project lays the foundation for smart warehouse logistics. However, there is significant potential for further enhancement and development. Some possible future improvements are but are not limited to:

Full Automation through Robot Networking

- A network of autonomous robots can be created where multiple robots communicate with each other using the ESP-NOW protocol.
- This will allow dynamic task sharing, route optimization, collision avoidance, and coordinated movement inside the warehouse, leading to a fully automated warehouse environment.

Voice-Controlled and Chatbot-Enabled Robots

- The robot can be upgraded with voice recognition technology to accept commands directly from users.
- Integration with a chatbot system and speaker will allow interactive communication, making it easier for warehouse staff to give instructions, receive updates, and monitor tasks verbally.

Hydraulic Arm Integration

- A hydraulic-powered robotic arm can be added to the system.
- This will enable the robot to lift heavier packages and drop or pick up cargo from varying shelf heights, greatly expanding its capabilities beyond just surface-level transportation.

Infrared (IR) Module for Object Detection and Avoidance

- Adding IR sensors will allow the robot to perform better obstacle detection and avoidance.
- It will increase the safety and reliability of autonomous navigation, especially in congested or dynamic warehouse environments.

Fire Detection and Safety Monitoring

- Integration of fire detection modules (like flame sensors or temperature sensors) will allow the robot to monitor warehouse environments for fire hazards.
- In case of detection, the robot can raise immediate alarms and assist in evacuation or damage control operations.