#include <WiFi.h>

#include <ESPAsyncWebServer.h>

#include <AsyncTCP.h>

const char\* ssid = "V30+\_6936";

const char\* password = "doraboots";

// motor 1 pins

int motor1Pin1 = 27;

int motor1Pin2 = 26;

int enable1Pin = 14;

// motor 2 pins

int motor2Pin1 = 25;

int motor2Pin2 = 33;

int enable2Pin = 32;

const int freq = 30000;

const int pwmChannel1 = 0;

const int pwmChannel2 = 1;

const int resolution = 8;

int dutyCycle = 0;

const int maxDutyCycle = 255;

const int accelerationStep = 5;

const int decelerationStep = 5;

int currentDutyCycle = 0;

AsyncWebServer server(80);

AsyncWebSocket ws("/ws");

unsigned long lastCommandTime = 0; // Store last command timestamp

const unsigned long commandTimeout = 1000; // 1-second timeout

void setup() {

pinMode(motor1Pin1, OUTPUT);

pinMode(motor1Pin2, OUTPUT);

pinMode(motor2Pin1, OUTPUT);

pinMode(motor2Pin2, OUTPUT);

ledcSetup(pwmChannel1, freq, resolution);

ledcAttachPin(enable1Pin, pwmChannel1);

ledcSetup(pwmChannel2, freq, resolution);

ledcAttachPin(enable2Pin, pwmChannel2);

Serial.begin(115200);

connectToWiFi();

ws.onEvent(onWebSocketEvent);

server.addHandler(&ws);

server.begin();

Serial.println("Started Server");

}

void connectToWiFi() {

WiFi.mode(WIFI\_AP);

WiFi.softAP(ssid, password);

Serial.print("Access Point Started. IP address: ");

Serial.println(WiFi.softAPIP());

// while (WiFi.status() != WL\_CONNECTED) {

// delay(1000);

// Serial.println("Connecting to WiFi...");

// }

// Serial.println("Connected to WiFi");

}

void onWebSocketEvent(AsyncWebSocket \*server, AsyncWebSocketClient \*client,

AwsEventType type, void \*arg, uint8\_t \*data, size\_t len) {

if (type == WS\_EVT\_CONNECT) {

// check if client connected

Serial.println("Client connected");

}

else if (type == WS\_EVT\_DISCONNECT) {

Serial.println("Client disconnected");

// check if client disconnected

}

else if (type == WS\_EVT\_DATA) {

// if data is received

String message = (const char \*)data;

Serial.println("Received message: " + message);

lastCommandTime = millis(); // Update last command timestamp

if (message == "UP") {

accelerateForward();

} else if (message == "DOWN") {

accelerateBackward();

} else if (message == "LEFT") {

turnLeft();

} else if (message == "RIGHT") {

turnRight();

} else if (message == "STOP") {

decelerate();

}

}

}

void accelerateForward() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, HIGH);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, HIGH);

if (currentDutyCycle < maxDutyCycle) {

currentDutyCycle += accelerationStep;

}

ledcWrite(pwmChannel1, currentDutyCycle);

ledcWrite(pwmChannel2, currentDutyCycle);

}

void accelerateBackward() {

digitalWrite(motor1Pin1, HIGH);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, HIGH);

digitalWrite(motor2Pin2, LOW);

if (currentDutyCycle < maxDutyCycle) {

currentDutyCycle += accelerationStep;

}

ledcWrite(pwmChannel1, currentDutyCycle);

ledcWrite(pwmChannel2, currentDutyCycle);

}

void turnLeft() {

ledcWrite(pwmChannel1, currentDutyCycle / 2);

ledcWrite(pwmChannel2, currentDutyCycle);

}

void turnRight() {

ledcWrite(pwmChannel1, currentDutyCycle);

ledcWrite(pwmChannel2, currentDutyCycle / 2);

}

void decelerate() {

if (currentDutyCycle > 0) {

currentDutyCycle -= decelerationStep;

ledcWrite(pwmChannel1, currentDutyCycle);

ledcWrite(pwmChannel2, currentDutyCycle);

}

}

void failsafeCheck() {

if (millis() - lastCommandTime > commandTimeout) {

// Stop the motors if no command is received within the timeout

ledcWrite(pwmChannel1, 0);

ledcWrite(pwmChannel2, 0);

currentDutyCycle = 0; // Reset duty cycle to 0

}

}

void loop() {

ws.cleanupClients();

failsafeCheck();

// Reconnect to WiFi if disconnected

if (WiFi.status() != WL\_CONNECTED) {

connectToWiFi();

}

}

**WEBSITE CODE**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<title>Battle Bot Control</title>**

**</head>**

**<body>**

**<h2>Control the Battle Bot</h2>**

**<script>**

**// Define the WebSocket variable globally to access it in multiple functions**

**let ws;**

**// Function to connect to the WebSocket**

**function connectWebSocket() {**

**// Replace <ESP32\_IP> with the actual IP address of the ESP32 AP, like 192.168.4.1**

**ws = new WebSocket('ws://192.168.4.1/ws');**

**ws.onopen = () => {**

**console.log('Connected to WebSocket');**

**};**

**ws.onmessage = (event) => {**

**console.log("Received message from ESP32: " + event.data);**

**};**

**ws.onerror = (error) => {**

**console.error("WebSocket error:", error);**

**};**

**ws.onclose = (event) => {**

**console.log("WebSocket closed, attempting to reconnect...");**

**setTimeout(connectWebSocket, 1000); // Retry connection after 1 second**

**};**

**}**

**// Call the function to initially connect to the WebSocket**

**connectWebSocket();**

**// Check if WebSocket is open before sending**

**function sendCommand(command) {**

**if (ws.readyState === WebSocket.OPEN) {**

**ws.send(command);**

**} else {**

**console.log("WebSocket not connected yet. Command not sent:", command);**

**}**

**}**

**// Event listeners for keyboard controls**

**document.addEventListener('keydown', (event) => {**

**if (event.key === 'ArrowUp') {**

**sendCommand('UP');**

**} else if (event.key === 'ArrowDown') {**

**sendCommand('DOWN');**

**} else if (event.key === 'ArrowLeft') {**

**sendCommand('LEFT');**

**} else if (event.key === 'ArrowRight') {**

**sendCommand('RIGHT');**

**}**

**});**

**document.addEventListener('keyup', () => {**

**sendCommand('STOP');**

**});**

**</script>**

**</body>**

**</html>**