#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;

// motor 1 pins

int motor1Pin1 = 18;

int motor1Pin2 = 16;

int enable1Pin = 17;

const int freq = 30000;

//const int pwmChannel1 = 0;

//const int pwmChannel2 = 1;

const int resolution = 8;

int dutyCycle = 0;

const int maxDutyCycle = 240; // capping voltage to motors at 6V

const int accelerationStep = 5;

const int decelerationStep = 5;

int currentDutyCycle = 0;

int turnDutyCycle = 280;

AsyncWebServer server(80);

AsyncWebSocket ws("/ws");

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

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

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

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

void setup() {

pinMode(motor1Pin1, OUTPUT);

pinMode(motor1Pin2, OUTPUT);

pinMode(motor2Pin1, OUTPUT);

pinMode(motor2Pin2, OUTPUT);

pinMode(enable1Pin, OUTPUT);

pinMode(enable2Pin, OUTPUT);

Serial.begin(115200);

connectToWiFi();

ws.onEvent(onWebSocketEvent);

server.addHandler(&ws);

server.begin();

Serial.println("Started Server");

}

void connectToWiFi() {

WiFi.mode(WIFI\_STA); // Set ESP32 to Station mode

WiFi.begin(ssid, password); // Connect to hotspot

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

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

delay(1000);

Serial.print(".");

}

Serial.println("\nConnected to WiFi!");

Serial.println("ESP32 IP Address: " + WiFi.localIP().toString());

}

void accelerateForward() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, HIGH);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, HIGH);

if (currentDutyCycle < maxDutyCycle) {

currentDutyCycle += accelerationStep;

}

digitalWrite(enable1Pin, currentDutyCycle);

digitalWrite(enable2Pin, currentDutyCycle);

}

void accelerateBackward() {

digitalWrite(motor1Pin1, HIGH);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, HIGH);

digitalWrite(motor2Pin2, LOW);

if (currentDutyCycle < maxDutyCycle) {

currentDutyCycle += accelerationStep;

}

digitalWrite(enable1Pin, currentDutyCycle);

digitalWrite(enable2Pin, currentDutyCycle);

}

void turnLeft() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, HIGH);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, LOW);

digitalWrite(enable1Pin, turnDutyCycle / 2);

digitalWrite(enable2Pin, turnDutyCycle);

}

void turnRight() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, HIGH);

digitalWrite(enable1Pin, turnDutyCycle);

digitalWrite(enable2Pin, turnDutyCycle / 2);

}

void decelerate() {

if (currentDutyCycle > 0) {

currentDutyCycle -= accelerationStep;

}

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, LOW);

digitalWrite(enable1Pin, currentDutyCycle);

digitalWrite(enable2Pin, currentDutyCycle);

}

void immediateStop() {

Serial.println("Immediate stop triggered");

currentDutyCycle = 0; // Reset duty cycle

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, LOW);

// turn off motors

digitalWrite(enable1Pin, 0);

digitalWrite(enable2Pin, 0);

}

char currentCommand = 'S'; // Default to 'S' for stop

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

if (len > 0) {

char command = (char)data[0]; // Read only the first character --> for buffer management

Serial.println("Received command: " + String(command));

executeCommand();

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

currentCommand = command;

}

}

}

void executeCommand() {

static char lastCommand = 'S'; // Store the last executed command

if (currentCommand != lastCommand) {

lastCommand = currentCommand; // Update the last executed command

// Execute the function corresponding to the current command

if (currentCommand == 'U') {

accelerateForward();

Serial.println(currentDutyCycle);

} else if (currentCommand == 'D') {

accelerateBackward();

Serial.println(currentDutyCycle);

} else if (currentCommand == 'L') {

turnLeft();

} else if (currentCommand == 'R') {

turnRight();

} else if (currentCommand == 'X') {

decelerate();

} else if (currentCommand == 'S') {

immediateStop();

}

}

}

void failsafeCheck() {

if (millis() - lastCommandTime > commandTimeout && currentCommand != 'S') {

Serial.println("Failsafe triggered: Stopping motors");

currentCommand = 'S'; // Set to stop

currentDutyCycle = 0;

// stop motors

digitalWrite(enable1Pin, 0);

digitalWrite(enable2Pin, 0);

}

}

void loop() {

ws.cleanupClients();

failsafeCheck();

// Non-blocking WiFi reconnection

static unsigned long lastWiFiCheck = 0;

if (millis() - lastWiFiCheck > 10000) { // Check every 10 seconds

lastWiFiCheck = millis();

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

connectToWiFi();

}

}

}