<https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json>

/\* 8-motor skid-steer rover for ESP32 (38-pin).

Controls 4 L298N modules (8 motors), 1 servo (boot), HC-SR04 front sensor.

Uses PWM on IN pins (no separate EN PWM assumed).

Web server runs in SoftAP mode at 192.168.4.1

IMPORTANT: Calibrate TURN\_MS\_45 for your robot - see comments below.

\*/

#include <WiFi.h>

#include <WebServer.h>

#include <ESPmDNS.h>

#include <ESPAsyncWebServer.h> // not required; using WebServer below

#include <Arduino.h>

#include <Servo.h>

// ---------- CONFIG ----------

const char\* ssid\_ap = "RoverAP";

const char\* ap\_password = "rover1234"; // 8+ chars recommended

// Motor pins: each motor needs two IN pins (INa, INb)

// Motor order: L1,R1,L2,R2,L3,R3,L4,R4 (indices 0..7)

const uint8\_t motorPins[8][2] = {

{2, 4}, // L1

{5, 12}, // R1

{13, 14}, // L2

{15, 16}, // R2

{17, 18}, // L3

{19, 21}, // R3

{22, 23}, // L4

{25, 26} // R4

};

// We'll create one PWM channel per IN pin (16 channels total)

const int pwmFreq = 2000;

const int pwmResolution = 8; // 0-255

int pwmChannel[8][2]; // map motors->channels

// Servo

const int servoPin = 27;

Servo bootServo;

const int SERVO\_CLOSED = 0;

const int SERVO\_OPEN = 90;

// HC-SR04 pins

const int trigPin = 32;

const int echoPin = 33; // must be level-shifted externally

// distances

const float STOP\_DISTANCE\_CM = 25.0; // threshold to stop and scan

const unsigned long ECHO\_TIMEOUT = 30000; // microseconds

// timed-turn constants (must calibrate)

unsigned long TURN\_MS\_45 = 500; // initial guess: time to rotate ~45 degrees (ms) - calibrate

int defaultSpeed = 200; // 0-255

// Autonomous mode flag

volatile bool autoMode = false;

WebServer server(80);

// ---------- Helpers ----------

void pwmSetupChannels() {

int ch = 0;

for (int m = 0; m < 8; ++m) {

for (int b = 0; b < 2; ++b) {

pwmChannel[m][b] = ch;

ledcSetup(ch, pwmFreq, pwmResolution);

ledcAttachPin(motorPins[m][b], ch);

++ch;

}

}

}

void setMotor(int mIndex, int dir, int speed) {

// dir: 1 = forward (INa pwm, INb LOW)

// dir: -1 = backward (INa LOW, INb pwm)

// dir: 0 = stop (both LOW)

if (mIndex < 0 || mIndex >= 8) return;

uint8\_t pinA = motorPins[mIndex][0];

uint8\_t pinB = motorPins[mIndex][1];

int chA = pwmChannel[mIndex][0];

int chB = pwmChannel[mIndex][1];

if (dir == 1) {

ledcWrite(chA, speed);

ledcWrite(chB, 0);

} else if (dir == -1) {

ledcWrite(chA, 0);

ledcWrite(chB, speed);

} else {

ledcWrite(chA, 0);

ledcWrite(chB, 0);

}

}

void setAllMotors(int dir, int speed) {

for (int i = 0; i < 8; ++i) setMotor(i, dir, speed);

}

void stopAll() {

setAllMotors(0,0);

}

void forwardCmd() {

// forward: all motors forward

setAllMotors(1, defaultSpeed);

}

void reverseCmd() {

setAllMotors(-1, defaultSpeed);

}

void skidLeftCmd() {

// skid left: left side reverse, right side forward (in-place turn left)

// left motors: indices 0,2,4,6 ; right: 1,3,5,7

for (int i = 0; i < 8; ++i) {

if (i % 2 == 0) setMotor(i, -1, defaultSpeed); // left motors

else setMotor(i, 1, defaultSpeed); // right motors

}

}

void skidRightCmd() {

for (int i = 0; i < 8; ++i) {

if (i % 2 == 0) setMotor(i, 1, defaultSpeed); // left motors forward

else setMotor(i, -1, defaultSpeed); // right motors backward

}

}

float getDistanceCM() {

// returns -1 on error

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

unsigned long duration = pulseIn(echoPin, HIGH, ECHO\_TIMEOUT);

if (duration == 0) return -1.0;

float dist = (duration / 2.0) \* 0.0343; // cm (speed of sound ~343 m/s)

return dist;

}

// Rotate in place right for ms ms (approx degrees depend on TURN\_MS\_45)

void timedRotateRight(unsigned long ms) {

skidRightCmd();

delay(ms);

stopAll();

}

void timedRotateLeft(unsigned long ms) {

skidLeftCmd();

delay(ms);

stopAll();

}

void bootOpen() {

bootServo.write(SERVO\_OPEN);

}

void bootClose() {

bootServo.write(SERVO\_CLOSED);

}

// ---------- Web server handlers ----------

String pageHtml() {

String s = R"rawliteral(

<!doctype html>

<html>

<head><meta name="viewport" content="width=device-width, initial-scale=1"/>

<title>Rover Control</title>

<style>

body{font-family:Arial;text-align:center;margin:10px}

button{padding:18px 24px;margin:6px;font-size:18px}

#dist{font-weight:bold}

</style>

</head>

<body>

<h2>Rover Control</h2>

<div>

<button onclick="cmd('forward')">Forward</button><br/>

<button onclick="cmd('left')">Left</button>

<button onclick="cmd('stop')">Stop</button>

<button onclick="cmd('right')">Right</button><br/>

<button onclick="cmd('reverse')">Reverse</button>

</div>

<div style="margin-top:12px;">

<button onclick="cmd('bootopen')">Open Boot</button>

<button onclick="cmd('bootclose')">Close Boot</button>

</div>

<div style="margin-top:12px;">

<button onclick="cmd('autotoggle')">Toggle Auto Mode</button>

<p>Auto mode: <span id="automode">OFF</span></p>

</div>

<p>Front distance: <span id="dist">--</span> cm</p>

<script>

function cmd(c){

fetch('/cmd?c='+c).then(r=>r.text()).then(t=>console.log(t));

}

setInterval(()=>{

fetch('/status').then(r=>r.json()).then(j=>{

document.getElementById('dist').innerText = j.distance.toFixed(1);

document.getElementById('automode').innerText = j.auto ? 'ON' : 'OFF';

});

}, 700);

</script>

</body>

</html>

)rawliteral";

return s;

}

void handleRoot() {

server.send(200, "text/html", pageHtml());

}

void handleStatus() {

float d = getDistanceCM();

if (d < 0) d = 9999.0;

String json = String("{\"distance\":") + d + String(",\"auto\":") + (autoMode ? "true" : "false") + String("}");

server.send(200, "application/json", json);

}

void handleCmd() {

if (!server.hasArg("c")) { server.send(400, "text/plain", "Bad"); return; }

String c = server.arg("c");

if (c=="forward") { forwardCmd(); server.send(200,"text/plain","OK"); return; }

if (c=="reverse") { reverseCmd(); server.send(200,"text/plain","OK"); return; }

if (c=="left") { skidLeftCmd(); server.send(200,"text/plain","OK"); return; }

if (c=="right") { skidRightCmd(); server.send(200,"text/plain","OK"); return; }

if (c=="stop") { stopAll(); server.send(200,"text/plain","OK"); return; }

if (c=="bootopen") { bootOpen(); server.send(200,"text/plain","OK"); return; }

if (c=="bootclose") { bootClose(); server.send(200,"text/plain","OK"); return; }

if (c=="autotoggle") { autoMode = !autoMode; server.send(200,"text/plain", autoMode ? "AUTO ON" : "AUTO OFF"); return; }

server.send(400,"text/plain","Unknown");

}

// ---------- Autonomous behavior ----------

void autonomousStep() {

// Called regularly in loop() when autoMode == true.

float d = getDistanceCM();

if (d > 0 && d <= STOP\_DISTANCE\_CM) {

// obstacle: stop and scan

stopAll();

delay(100);

// rotate right ~45

timedRotateRight(TURN\_MS\_45);

delay(100);

float dR = getDistanceCM();

if (dR > 0 && dR > STOP\_DISTANCE\_CM) {

// path found

forwardCmd();

return;

}

// rotate back to center

timedRotateLeft(TURN\_MS\_45);

// rotate left ~45

timedRotateLeft(TURN\_MS\_45);

delay(100);

float dL = getDistanceCM();

if (dL > 0 && dL > STOP\_DISTANCE\_CM) {

forwardCmd();

return;

}

// neither side clear -> rotate 180 and try reverse briefly

timedRotateRight(2 \* TURN\_MS\_45); // approximately opposite

reverseCmd();

delay(500);

stopAll();

delay(200);

} else {

// clear path, go forward

forwardCmd();

}

}

// ---------- Setup ----------

void setup() {

Serial.begin(115200);

delay(100);

// pins

for (int m=0;m<8;++m){

pinMode(motorPins[m][0], OUTPUT);

pinMode(motorPins[m][1], OUTPUT);

}

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

// servo

bootServo.attach(servoPin);

bootClose();

// PWM channels

pwmSetupChannels();

// WiFi AP

WiFi.softAP(ssid\_ap, ap\_password);

IPAddress myIP = WiFi.softAPIP();

Serial.print("AP IP address: ");

Serial.println(myIP);

server.on("/", handleRoot);

server.on("/status", handleStatus);

server.on("/cmd", handleCmd);

server.begin();

// small blink

digitalWrite(trigPin, LOW);

}

// ---------- Loop ----------

unsigned long lastAutoCheck = 0;

void loop() {

server.handleClient();

if (autoMode) {

unsigned long now = millis();

if (now - lastAutoCheck > 200) { // check ~5x/sec

autonomousStep();

lastAutoCheck = now;

}

}

}