```
#include <WiFiS3.h>
#include <WiFiUdp.h>

// ===================== PINS =====================
// Motors
int enA = 9, in1 = 8, in2 = 7;
int enB = 10, in3 = 12, in4 = 13;

// Front Ultrasonic (Trig=5, Echo=4)
const int FRONT_TRIG_PIN = 5;
const int FRONT_ECHO_PIN = 4;

// Left Ultrasonic (Trig = 11, Echo = 6)
const int LEFT_TRIG_PIN = 11;
const int LEFT_ECHO_PIN = 6;

// Photocell (for telemetry)
const int PHOTO_PIN = A0;

// ===================== SETTINGS =====================
const char* ssid = "COGS300_BOT";
const char* pass = "robot1234";

int speed = 150;            // 0-255 (Try 255 if it won't move)
const int WALL_THRESHOLD = 50; // cm
const int FRONT_THRESHOLD = 15;     // cm
const int ERROR_MARGIN = 5;

// ===================== STATE =====================
WiFiUDP udp;
unsigned long lastCmdMs = 0;
char driveMode = 'S';       // F, B, L, R, S
bool selfDrive = false;
int frontDistance = 0;
int wallDistance = 0;
unsigned long lastTelemMs = 0;
enum Driving { FORWARD, TURNING};
Driving currentlyDriving = FORWARD;

// *** FIXED: ADDED THIS MISSING LINE ***
```

```
IPAddress telemetryIP(192, 168, 4, 255);

void setup() {
  Serial.begin(115200);

  pinMode(enA, OUTPUT); pinMode(in1, OUTPUT); pinMode(in2, OUTPUT);
  pinMode(enB, OUTPUT); pinMode(in3, OUTPUT); pinMode(in4, OUTPUT);

  pinMode(FRONT_TRIG_PIN, OUTPUT);
  pinMode(FRONT_ECHO_PIN, INPUT);
  pinMode(LEFT_TRIG_PIN, OUTPUT);
  pinMode(LEFT_ECHO_PIN, INPUT);

  // Connect to WiFi
  if (WiFi.beginAP(ssid, pass) != WL_AP_LISTENING) {
    Serial.println("AP Failed");
  }
  udp.begin(4210);
  Serial.println("Robot Ready. Listening on Port 4210");
}

void loop() {
  // 1. Read Sensor
  readFrontUltrasonic();
  int photo = analogRead(PHOTO_PIN);
  readLeftUltrasonic();
  // 2. Handle WiFi Commands
  int packetSize = udp.parsePacket();
  if (packetSize > 0) {
    char cmd = udp.read();
    lastCmdMs = millis();

    // Command Logic
    if (cmd == 'M') {
      selfDrive = true;
      Serial.println("Mode: Follow Me");
    }
    else if (cmd == 'S') { // Stop key (Space)
      selfDrive = false;
      driveMode = 'S';
```

```
      Serial.println("Mode: Stop");
    }
    else {
      // Manual controls (F, B, L, R) disable Follow Me automatically
      selfDrive = false;
      handleManualCmd(cmd);
    }
  }


  // 3. Drive Logic
if (selfDrive) {
  if (currentlyDriving == FORWARD) {
    if (frontDistance > FRONT_THRESHOLD) {
      driveMode = 'F';  // Path clear, keep going forward
    }
    else {
      currentlyDriving = TURNING;
      if (wallDistance >= WALL_THRESHOLD + ERROR_MARGIN) {
        driveMode = 'L';
      }
      else {
        driveMode = 'R';
      }
    }
  }
  else if (currentlyDriving == TURNING) {
    // Keep turning until path is clear
    if (frontDistance > FRONT_THRESHOLD) {
      currentlyDriving = FORWARD;  // Exit turning state
      driveMode = 'F';
    }
    // else: maintain current driveMode (keep turning)
  }
}


  // 4. Apply Motors
  applyDriveMode();
```

```
  // 5. Send Data to Processing
  if (millis() - lastTelemMs > 100) {
    sendTelemetry(photo);
    lastTelemMs = millis();
  }
}

// ================ HELPERS ================

void readFrontUltrasonic() {
  digitalWrite(FRONT_TRIG_PIN, LOW); delayMicroseconds(2);
  digitalWrite(FRONT_TRIG_PIN, HIGH); delayMicroseconds(10);
  digitalWrite(FRONT_TRIG_PIN, LOW);

  long duration = pulseIn(FRONT_ECHO_PIN, HIGH, 30000);
  if (duration > 0) frontDistance = duration * 0.034 / 2;
  else frontDistance = 999;
}

void readLeftUltrasonic() {
  digitalWrite(LEFT_TRIG_PIN, LOW); delayMicroseconds(2);
  digitalWrite(LEFT_TRIG_PIN, HIGH); delayMicroseconds(10);
  digitalWrite(LEFT_TRIG_PIN, LOW);

  long duration = pulseIn(LEFT_ECHO_PIN, HIGH, 30000);
  if (duration > 0) wallDistance = duration * 0.034 / 2;
  else wallDistance = 999;
}

void handleManualCmd(char cmd) {
  if (cmd == 'F' || cmd == 'B' || cmd == 'L' || cmd == 'R' || cmd == 'S')
{
    driveMode = cmd;
  }
  // Allow speed changes 0-9
  if (cmd >= '0' && cmd <= '9') {
    speed = map(cmd - '0', 0, 9, 0, 255);
  }
}
```

```
void applyDriveMode() {
  // If robot moves BACKWARD when you press FORWARD, swap HIGH/LOW below
  if (driveMode == 'S') {
    analogWrite(enA, 0); analogWrite(enB, 0);
  } else if (driveMode == 'F') {
    digitalWrite(in1, LOW); digitalWrite(in2, HIGH);
    digitalWrite(in3, LOW); digitalWrite(in4, HIGH);
    analogWrite(enA, speed); analogWrite(enB, speed+80);
  } else if (driveMode == 'B') {
    digitalWrite(in1, HIGH); digitalWrite(in2, LOW);
    digitalWrite(in3, HIGH); digitalWrite(in4, LOW);
    analogWrite(enA, speed); analogWrite(enB, speed);
  } else if (driveMode == 'L') {
    digitalWrite(in1, LOW); digitalWrite(in2, HIGH);
    digitalWrite(in3, HIGH); digitalWrite(in4, LOW);
    analogWrite(enA, speed); analogWrite(enB, speed +80);
  } else if (driveMode == 'R') {
    digitalWrite(in1, HIGH); digitalWrite(in2, LOW);
    digitalWrite(in3, LOW); digitalWrite(in4, HIGH);
    analogWrite(enA, speed); analogWrite(enB, speed+80);
  }
}

void sendTelemetry(int photo) {
  char buf[64];
  // Format: "DATA,photo,distance,activeMode"
  sprintf(buf, "DATA,%d,%d,%d", photo, frontDistance, selfDrive);
  udp.beginPacket(telemetryIP, 4211);
  udp.write(buf);
  udp.endPacket();
}
```