```
/*
                                                       จัดทำโดย
                                                สุรเทพ ทนทาน 1610901629

                                                จักริน ศรีหาญ  1660901420


ESP32 Car bot control by MicroBlue it’s BlueTooth app

The MicroBlue app allows users to control microcontrollers
wirelessly via Bluetooth/BLE. It works by establishing a
connection between the app and a microcontroller device,
enabling remote control and monitoring. The app provides a
customizable interface, allowing users to create layouts with
various components like buttons, sliders, and text inputs,
tailored to their specific project needs.

*/

#include <BleSerial.h>
BleSerial ble;

// PWM configuration
#define PWM_PIN 25          // GPIO 25 for L298N ENA (speed
control)
#define IN1_PIN 26          // GPIO 26 for L298N IN1 (direction)
#define IN2_PIN 27          // GPIO 27 for L298N IN2 (direction)

// LED configuration
#define BLE_LED_PIN 2       // GPIO 2 for BLE connection status
LED
#define TIMER_LED_PIN 4     // GPIO 4 for timer interrupt
indication

// Ultrasonic sensor configuration
#define TRIG_PIN 5          // GPIO 5 for HC-SR04 Trigger
#define ECHO_PIN 18         // GPIO 18 for HC-SR04 Echo
#define DISTANCE_THRESHOLD 15 // Distance threshold in cm

// Parsing variables
String receivedID = "";
String receivedValue = "";
bool parsingID = false;
bool parsingValue = false;
volatile int lastPwmValue = 0; // Store last valid PWM value
from slider
volatile bool isForward = true; // Store direction (true =
forward, false = reverse)

// Timer configuration
```

```
hw_timer_t *timer = NULL;
volatile bool checkDistanceFlag = false; // Flag to trigger distance check
volatile bool toggleTimerLed = false;    // Control timer LED toggling
volatile byte timerLedState = LOW;       // Timer LED state

// Timer interrupt handler
void IRAM_ATTR onTimer() {
  checkDistanceFlag = true; // Signal to check distance in loop
  if (toggleTimerLed) {
    timerLedState = !timerLedState;
    digitalWrite(TIMER_LED_PIN, timerLedState); // Toggle LED if enabled
  } else {
    timerLedState = LOW;
    digitalWrite(TIMER_LED_PIN, LOW); // Keep LED off
  }
}

float measureDistance() {
  // Send 10us pulse to Trigger
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Measure Echo pulse duration
  long duration = pulseIn(ECHO_PIN, HIGH, 15000); // Timeout after 15ms (~2.5m)

  // Calculate distance (speed of sound = 343 m/s, or 0.0343 cm/us)
  return (duration == 0) ? 400.0 : (duration * 0.0343 / 2);
}

void setMotorDirection(bool forward) {
  if (forward) {
    digitalWrite(IN1_PIN, HIGH);
    digitalWrite(IN2_PIN, LOW); // Forward
  } else {
    digitalWrite(IN1_PIN, LOW);
    digitalWrite(IN2_PIN, HIGH); // Reverse
  }
}

void stopMotor() {
  analogWrite(PWM_PIN, 0); // Disable PWM
  digitalWrite(IN1_PIN, LOW);
  digitalWrite(IN2_PIN, LOW); // Stop motor
}

void setup() {
  Serial.begin(115200);
  ble.begin("ESP32_MicroBlue"); // BLE device name for
MicroBlue

  // Initialize pins
  pinMode(BLE_LED_PIN, OUTPUT);
  pinMode(TIMER_LED_PIN, OUTPUT);
  pinMode(PWM_PIN, OUTPUT);
  pinMode(IN1_PIN, OUTPUT);
  pinMode(IN2_PIN, OUTPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Set initial states
  digitalWrite(BLE_LED_PIN, HIGH);   // BLE LED off until
connected
  digitalWrite(TIMER_LED_PIN, LOW);  // Timer LED off
  stopMotor();                       // Start with motor
stopped
  setMotorDirection(isForward);      // Set initial direction

  // Initialize timer (1 MHz, 1 tick = 1 µs)
  timer = timerBegin(1000000); // Frequency in Hz
  timerAttachInterrupt(timer, &onTimer);
  // Set alarm every 100 ms (100,000 ticks), auto-reload
  timerAlarm(timer, 100000, true, 0);

  Serial.println("Timer started");
}

void loop() {
  // Handle distance check from timer interrupt
  if (checkDistanceFlag) {
    float distance = measureDistance();
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
    if (distance < DISTANCE_THRESHOLD) {
      stopMotor(); // Stop motor
      toggleTimerLed = true; // Enable timer LED toggling
      Serial.println("Motor stopped: Object detected within 15
cm");
    } else {
      analogWrite(PWM_PIN, lastPwmValue); // Restore PWM
      setMotorDirection(isForward); // Restore direction
      toggleTimerLed = false; // Disable timer LED
    }
    checkDistanceFlag = false; // Reset flag
  }

  // Handle BLE communication
  if (ble.connected()) {
    digitalWrite(BLE_LED_PIN, LOW); // BLE LED on when
connected
    // Read and parse MicroBlue data
    if (ble.available()) {
      while (ble.available()) {
        char c = ble.read();
        Serial.print(c); // Echo to Serial for debugging
        if (c == 1) { // SOH: Start of ID
          parsingID = true;
          parsingValue = false;
          receivedID = "";
        } else if (c == 2) { // STX: Start of Value
          parsingID = false;
          parsingValue = true;
          receivedValue = "";
        } else if (c == 3) { // ETX: End of transmission
          parsingID = false;
          parsingValue = false;
          processData();
        } else if (parsingID) {
          receivedID += c;
        } else if (parsingValue) {
          receivedValue += c;
        }
      }
      Serial.println(); // Newline after data
    }
  } else {
    digitalWrite(BLE_LED_PIN, HIGH); // BLE LED off when
disconnected
    stopMotor(); // Stop motor
    toggleTimerLed = false; // Disable timer LED
    digitalWrite(TIMER_LED_PIN, LOW);
  }
}

void processData() {
  Serial.print("ID: ");
  Serial.print(receivedID);
  Serial.print(", Value: ");
  Serial.println(receivedValue);
  if (receivedID == "s1") { // Speed slider
    int pwmValue = receivedValue.toInt();
  if (pwmValue >= 0 && pwmValue <= 255) {
   lastPwmValue = pwmValue; // Store PWM value
   // Apply PWM only if distance > threshold
   float distance = measureDistance();
   if (distance > DISTANCE_THRESHOLD) {
    analogWrite(PWM_PIN, pwmValue); // Set PWM
    setMotorDirection(isForward); // Set direction
    Serial.print("Set PWM to: ");
    Serial.println(pwmValue);
   }
  }
 } else if (receivedID == "b1") { // Direction button
  int directionValue = receivedValue.toInt();
  isForward = (directionValue == 0); // 0 = forward, 1 =
reverse
  // Apply direction only if distance > threshold
  float distance = measureDistance();
  if (distance > DISTANCE_THRESHOLD) {
   setMotorDirection(isForward);
   analogWrite(PWM_PIN, lastPwmValue); // Restore PWM
   Serial.print("Set direction: ");
   Serial.println(isForward ? "Forward" : "Reverse");
  }
 }
}
```