Arduino Uno with L298n

Deep Explanation of the Embedded C Code

This code is designed to control a robot using multiple inputs and sensors, including Bluetooth, IR remote, ultrasonic sensors, and infrared sensors for line following. The robot can perform various tasks like manual control, line following, and obstacle avoidance.

Libraries and Pin Definitions

```
c
Copy code
#include<SoftwareSerial.h>
SoftwareSerial BT_Serial(2, 3); // RX, TX

#include <IRremote.h>
const int RECV_PIN = A5;
IRrecv irrecv(RECV_PIN);
decode_results results;

#define enA 10 // Enable1 L298 Pin enA
#define in1 9  // Motor1  L298 Pin in1
#define in2 8  // Motor1  L298 Pin in1
#define in3 7  // Motor2  L298 Pin in1
#define in4 6  // Motor2  L298 Pin in1
#define enB 5  // Enable2 L298 Pin enB

#define servo A4

#define R_S A0 // IR sensor Right
#define L_S A1 // IR sensor Left

#define echo A2    // Echo pin
#define trigger A3 // Trigger pin
```

- **SoftwareSerial.h**: Allows serial communication on other digital pins of the Arduino.
- **IRremote.h**: Used for receiving signals from an IR remote.
- **Motor Control Pins (enA, in1, in2, in3, in4, enB)**: Connected to the L298 motor driver to control two motors.
- **Sensor Pins (servo, R_S, L_S, echo, trigger)**: Connected to various sensors (IR, ultrasonic, and servo).

Variables

```
c
Copy code
int distance_L, distance_F = 30, distance_R;
long distance;
int set = 20;

int bt_ir_data; // Variable to receive data from the serial port and IRremote
int Speed = 130;
int mode = 0;
int IR_data;
```

- **distance_L, distance_F, distance_R**: Stores distance measurements.
- **bt_ir_data**: Stores data received from Bluetooth or IR remote.
- **Speed**: Controls the speed of the motors.
- **mode**: Controls the operational mode (manual, line following, obstacle avoiding).
- **IR_data**: Stores decoded IR remote data.

Setup Function

```
c
Copy code
void setup() {
    pinMode(R_S, INPUT);
    pinMode(L_S, INPUT);
    pinMode(echo, INPUT);
    pinMode(trigger, OUTPUT);
    pinMode(enA, OUTPUT);
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);
    pinMode(enB, OUTPUT);
    irrecv.enableIRIn();
    irrecv.blink13(true);

    Serial.begin(9600);
    BT_Serial.begin(9600);
    pinMode(servo, OUTPUT);

    for (int angle = 70; angle <= 140; angle += 5) {
        servoPulse(servo, angle);
    }
    for (int angle = 140; angle >= 0; angle -= 5) {
        servoPulse(servo, angle);
    }
    for (int angle = 0; angle <= 70; angle += 5) {
        servoPulse(servo, angle);
    }
    delay(500);
}
```

- **pinMode**: Configures the specified pin to behave as an input or output.
- **irrecv.enableIRIn()**: Initializes the IR receiver.
- **Serial.begin(9600)**: Starts serial communication at 9600 bps.
- **BT_Serial.begin(9600)**: Starts Bluetooth serial communication.
- **Servo initialization**: Moves the servo to initial positions for calibration.

Main Loop

```
c
Copy code
void loop() {
    if (BT_Serial.available() > 0) {
        bt_ir_data = BT_Serial.read();
        Serial.println(bt_ir_data);
        if (bt_ir_data > 20) {
```

```
        Speed = bt_ir_data;
    }
}

if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    bt_ir_data = IRremote_data();
    Serial.println(bt_ir_data);
    irrecv.resume();
    delay(100);
}

if (bt_ir_data == 8) {
    mode = 0;
    Stop();
} else if (bt_ir_data == 9) {
    mode = 1;
    Speed = 130;
} else if (bt_ir_data == 10) {
    mode = 2;
    Speed = 255;
}

analogWrite(enA, Speed);
analogWrite(enB, Speed);

if (mode == 0) {
    if (bt_ir_data == 1) {
        forword();
    } else if (bt_ir_data == 2) {
        backword();
    } else if (bt_ir_data == 3) {
        turnLeft();
    } else if (bt_ir_data == 4) {
        turnRight();
    } else if (bt_ir_data == 5) {
        Stop();
    } else if (bt_ir_data == 6) {
        turnLeft();
        delay(400);
        bt_ir_data = 5;
    } else if (bt_ir_data == 7) {
        turnRight();
        delay(400);
        bt_ir_data = 5;
    }
}

if (mode == 1) {
    if ((digitalRead(R_S) == 0) && (digitalRead(L_S) == 0)) {
        forword();
    }
    if ((digitalRead(R_S) == 1) && (digitalRead(L_S) == 0)) {
        turnRight();
    }
    if ((digitalRead(R_S) == 0) && (digitalRead(L_S) == 1)) {
        turnLeft();
    }
```

```
        if ((digitalRead(R_S) == 1) && (digitalRead(L_S) == 1)) {
            Stop();
        }
    }

    if (mode == 2) {
        distance_F = Ultrasonic_read();
        Serial.print("S=");
        Serial.println(distance_F);
        if (distance_F > set) {
            forword();
        } else {
            Check_side();
        }
    }

    delay(10);
}
```

- **BT_Serial.available()**: Checks if data is available to read from the Bluetooth serial.
- **BT_Serial.read()**: Reads the incoming data from Bluetooth.
- **irrecv.decode(&results)**: Decodes the incoming IR signal.
- **Mode Control**: Sets the robot's operational mode based on the received data.
- **Motor Control**: Adjusts motor speed and direction based on the mode.

Helper Functions

- **IRremote_data**: Decodes the IR signal into commands.

```
c
Copy code
long IRremote_data() {
    if (results.value == 0xFF02FD) {
        IR_data = 1;
    } else if (results.value == 0xFF9867) {
        IR_data = 2;
    } else if (results.value == 0xFFE01F) {
        IR_data = 3;
    } else if (results.value == 0xFF906F) {
        IR_data = 4;
    } else if (results.value == 0xFF629D || results.value == 0xFFA857) {
        IR_data = 5;
    } else if (results.value == 0xFF30CF) {
        IR_data = 8;
    } else if (results.value == 0xFF18E7) {
        IR_data = 9;
    } else if (results.value == 0xFF7A85) {
        IR_data = 10;
    }
    return IR_data;
}
```

- **servoPulse**: Generates a servo pulse to control the servo motor.

```
c
```

```
Copy code
void servoPulse(int pin, int angle) {
    int pwm = (angle * 11) + 500;
    digitalWrite(pin, HIGH);
    delayMicroseconds(pwm);
    digitalWrite(pin, LOW);
    delay(50);
}
```

- **Ultrasonic_read**: Reads the distance using the ultrasonic sensor.

```
c
Copy code
long Ultrasonic_read() {
    digitalWrite(trigger, LOW);
    delayMicroseconds(2);
    digitalWrite(trigger, HIGH);
    delayMicroseconds(10);
    distance = pulseIn(echo, HIGH);
    return distance / 29 / 2;
}
```

- **compareDistance and Check_side**: Determine the direction to turn based on ultrasonic sensor readings.

```
c
Copy code
void compareDistance() {
    if (distance_L > distance_R) {
        turnLeft();
        delay(350);
    } else if (distance_R > distance_L) {
        turnRight();
        delay(350);
    } else {
        backword();
        delay(300);
        turnRight();
        delay(600);
    }
}

void Check_side() {
    Stop();
    delay(100);
    for (int angle = 70; angle <= 140; angle += 5) {
        servoPulse(servo, angle);
    }
    delay(300);
    distance_L = Ultrasonic_read();
    delay(100);
    for (int angle = 140; angle >= 0; angle -= 5) {
        servoPulse(servo, angle);
    }
    delay(500);
    distance_R = Ultrasonic_read();
    delay(100);
```

```
    for (int angle = 0; angle <= 70; angle += 5) {
        servoPulse(servo, angle);
    }
    delay(300);
    compareDistance();
}
```

- **Motor Control Functions**: Control the movement of the robot.

```
c
Copy code
void forword() {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
}

void backword() {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
}

void turnRight() {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
}

void turnLeft() {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
}

void Stop() {
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
}
```

Arduino with GPS

Deep Explanation of the Embedded C Code

This code is designed to interface with a GPS module and display the latitude and longitude on an LCD screen. It uses the `TinyGPS++` library for decoding GPS data and the `LiquidCrystal` library for displaying information on an LCD. Let's go through the code step-by-step.

Libraries and Objects

```
c
Copy code
#include <LiquidCrystal.h>
#include <SoftwareSerial.h>
#include <TinyGPS++.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
SoftwareSerial gpsSerial(8, 9); // rx, tx
TinyGPSPlus gps; // create gps object
```

- **LiquidCrystal.h**: Library for controlling an LCD.
- **SoftwareSerial.h**: Library for enabling serial communication on other digital pins.
- **TinyGPS++.h**: Library for parsing and interpreting GPS data.

Variables

```
c
Copy code
float lattitude, longitude; // create variable for latitude and longitude object
```

- **latitude, longitude**: Floating point variables to store the latitude and longitude values obtained from the GPS.

Setup Function

```
c
Copy code
void setup() {
    Serial.begin(9600); // connect serial
    lcd.begin(16, 2); // initialize the LCD with 16 columns and 2 rows
    gpsSerial.begin(9600); // connect GPS sensor at 9600 baud rate

    lcd.print("GPS TRACKER"); // display "GPS TRACKER" on the LCD
    delay(2000); // wait for 2 seconds
    lcd.clear(); // clear the LCD
}
```

- **Serial.begin(9600)**: Initializes serial communication at 9600 bps.
- **lcd.begin(16, 2)**: Initializes the LCD with 16 columns and 2 rows.
- **gpsSerial.begin(9600)**: Initializes the software serial communication for the GPS module.

- **lcd.print("GPS TRACKER")**: Displays a welcome message on the LCD.
- **delay(2000)**: Waits for 2 seconds before clearing the screen.

Loop Function

```
c
Copy code
void loop() {
    while (1) {
        while (gpsSerial.available() > 0) {
            gps.encode(gpsSerial.read());
        }

        if (gps.location.isUpdated()) {
            Serial.print("LAT=");
            Serial.println(gps.location.lat(), 6);
            Serial.print("LONG=");
            Serial.println(gps.location.lng(), 6);

            latitude = gps.location.lat();
            longitude = gps.location.lng();
            break;
        }
    }

    Serial.print("LATTITUDE=");
    Serial.println(latitude, 6);
    Serial.print("LONGITUDE=");
    Serial.println(longitude, 6);

    lcd.print("LAT ");
    lcd.print(latitude, 6);
    lcd.setCursor(0, 1);
    lcd.print("LONG ");
    lcd.print(longitude, 6);

    delay(1000);
    lcd.clear();
}
```

Inner Loop

```
c
Copy code
while (1) {
    while (gpsSerial.available() > 0) {
        gps.encode(gpsSerial.read());
    }

    if (gps.location.isUpdated()) {
        Serial.print("LAT=");
        Serial.println(gps.location.lat(), 6);
        Serial.print("LONG=");
        Serial.println(gps.location.lng(), 6);

        latitude = gps.location.lat();
        longitude = gps.location.lng();
```

```
        break;
    }
}
```

- **while (1)**: An infinite loop to continuously check for GPS data.
- **gpsSerial.available() > 0**: Checks if there is data available to read from the GPS module.
- **gps.encode(gpsSerial.read())**: Reads the data from the GPS module and encodes it using the TinyGPS++ library.
- **gps.location.isUpdated()**: Checks if the location data has been updated.
- **latitude = gps.location.lat()**: Retrieves the latitude from the GPS data.
- **longitude = gps.location.lng()**: Retrieves the longitude from the GPS data.
- **break**: Exits the inner loop once valid GPS data is obtained.

Display and Serial Output

```
c
Copy code
Serial.print("LATTITUDE=");
Serial.println(latitude, 6);
Serial.print("LONGITUDE=");
Serial.println(longitude, 6);

lcd.print("LAT ");
lcd.print(latitude, 6);
lcd.setCursor(0, 1);
lcd.print("LONG ");
lcd.print(longitude, 6);

delay(1000);
lcd.clear();
```

- **Serial.print("LATTITUDE=")** and **Serial.print("LONGITUDE=")**: Print the latitude and longitude values to the serial monitor.
- **lcd.print("LAT ")** and **lcd.print("LONG ")**: Display the latitude and longitude values on the LCD.
- **lcd.setCursor(0, 1)**: Sets the cursor to the second row of the LCD.
- **delay(1000)**: Waits for 1 second before clearing the screen.
- **lcd.clear()**: Clears the LCD to prepare for the next update.

Arduino nano with ultrasonic sensor

Detailed Explanation of the Embedded C Code

This code is designed to implement a radar system using an Arduino, an ultrasonic sensor, an OLED display, and a servo motor. The system measures distances to objects within its range and displays these distances on the OLED screen while rotating the servo motor. The following libraries are used:

- **SPI.h**: Serial Peripheral Interface library, used for communication.
- **Wire.h**: I2C communication library.
- **Adafruit_GFX.h**: Core graphics library.
- **Adafruit_SSD1306.h**: Library for the SSD1306 OLED display.
- **Servo.h**: Library for controlling servo motors.

Libraries and Object Initialization

```
c
Copy code
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Servo.h>

#define Trig 8
#define Echo 9

Adafruit_SSD1306 display(-1);
Servo Servo1;
```

- **SPI.h** and **Wire.h**: Used for communication protocols.
- **Adafruit_GFX.h** and **Adafruit_SSD1306.h**: Used to control the OLED display.
- **Servo.h**: Used to control the servo motor.
- **#define Trig 8** and **#define Echo 9**: Define pins for the ultrasonic sensor.
- **Adafruit_SSD1306 display(-1)**: Create an OLED display object.
- **Servo Servo1**: Create a servo object.

Setup Function

```
c
Copy code
void setup() {
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    pinMode(Trig, OUTPUT);
    pinMode(Echo, INPUT);
    Serial.begin(9600);
    Servo1.attach(11);
    delay(1000);
    Servo1.write(40);
    display.clearDisplay();
    display.setTextSize(1);
```

```
    display.setTextColor(WHITE);
    display.setCursor(20, 10);
    display.println("Arduino RADAR");
    display.display();
    delay(2000);
}
```

- **display.begin(SSD1306_SWITCHCAPVCC, 0x3C)**: Initialize the OLED display with address 0x3C.
- **pinMode(Trig, OUTPUT)** and **pinMode(Echo, INPUT)**: Set ultrasonic sensor pins.
- **Serial.begin(9600)**: Initialize serial communication at 9600 bps.
- **Servo1.attach(11)**: Attach the servo motor to pin 11.
- **delay(1000)**: Wait for 1 second.
- **Servo1.write(40)**: Set servo to initial position.
- **display.clearDisplay()**: Clear the OLED display.
- **display.setTextSize(1)** and **display.setTextColor(WHITE)**: Set text size and color.
- **display.setCursor(20, 10)**: Set cursor position.
- **display.println("Arduino RADAR")**: Print "Arduino RADAR" on the OLED.
- **display.display()**: Update the display.
- **delay(2000)**: Wait for 2 seconds.

Loop Function

```
c
Copy code
long distance, duration, Distance;
void loop() {
    int i, j, k, l, m, n;

    // Loop for moving the line from center left to top left
    for (i = 10; i >= 0; i -= 2) {
        int servomap = map(i, 0, 10, 60, 40);
        Distance = Distance_value();

        Serial.print(servomap);
        Serial.print(",");
        Serial.print(Distance);
        Serial.print(".");

        if (Distance > 40) {
            display.clearDisplay();
            Servo1.write(servomap);
            display.drawLine(64, 32, 0, i, WHITE);
            display.display();
            delay(100);
        } else {
            int xmap1 = map(Distance, 0, 40, 64, 0);
            int ymap1 = map(Distance, 0, 40, 32, i);
            display.clearDisplay();
            Servo1.write(servomap);
            display.drawLine(64, 32, 0, i, WHITE);
            display.drawCircle(xmap1, ymap1, 3, WHITE);
            display.fillCircle(10, 30, 1, WHITE);
            display.fillRoundRect(10, 20, 2, 8, 2, WHITE);
            display.display();
            delay(100);
```

```
        }
    }
    ...
    // The rest of the loop follows a similar pattern
}
```

- **for (i = 10; i >= 0; i -= 2)**: Loop to move the servo motor and update the display.
- **int servomap = map(i, 0, 10, 60, 40)**: Map loop variable to servo angles.
- **Distance = Distance_value()**: Get distance from the ultrasonic sensor.
- **Serial.print(...)**: Print servo angle and distance for debugging.
- **if (Distance > 40)**: Check if the distance is out of range.
    - **display.clearDisplay()**, **Servo1.write(servomap)**: Clear display and set servo position.
    - **display.drawLine(64, 32, 0, i, WHITE)**: Draw line from center to the current position.
    - **display.display()**: Update the display.
    - **delay(100)**: Wait for 100 ms.
- **else**: If the distance is within range.
    - **int xmap1 = map(Distance, 0, 40, 64, 0)**, **int ymap1 = map(Distance, 0, 40, 32, i)**: Map distance to coordinates.
    - **display.clearDisplay()**, **Servo1.write(servomap)**: Clear display and set servo position.
    - **display.drawLine(64, 32, 0, i, WHITE)**: Draw line from center to the current position.
    - **display.drawCircle(xmap1, ymap1, 3, WHITE)**: Draw circle at the detected object position.
    - **display.fillCircle(10, 30, 1, WHITE)**, **display.fillRoundRect(10, 20, 2, 8, 2, WHITE)**: Draw detection signs.
    - **display.display()**: Update the display.
    - **delay(100)**: Wait for 100 ms.

Distance Measurement Function

```
c
Copy code
int Distance_value() {
    digitalWrite(Trig, LOW);
    delayMicroseconds(2);
    digitalWrite(Trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(Trig, LOW);
    duration = pulseIn(Echo, HIGH);
    distance = (duration / 2) / 29.1;
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(100, 25);
    display.println(distance);
    delay(15);
    display.display();
    return distance;
}
```

- **digitalWrite(Trig, LOW)**, **delayMicroseconds(2)**: Ensure trigger pin is low.
- **digitalWrite(Trig, HIGH)**, **delayMicroseconds(10)**: Send a 10-microsecond pulse.
- **digitalWrite(Trig, LOW)**: Stop the pulse.
- **duration = pulseIn(Echo, HIGH)**: Measure the duration of the echo pulse.
- **distance = (duration / 2) / 29.1**: Calculate distance in cm.

- **display.setTextSize(1)**, **display.setTextColor(WHITE)**, **display.setCursor(100, 25)**: Prepare to display the distance.
- **display.println(distance)**: Print distance on the OLED.
- **delay(15)**: Short delay.
- **display.display()**: Update the display.
- **return distance**: Return the calculated distance.