ESP32 Board A - Overview & Functions

Main Components & Features:

- ESP-NOW Protocol: The board communicates with a second ESP32 board using the ESP-NOW protocol, ensuring efficient two-way communication without needing a Wi-Fi network. The MAC address of the second ESP32 board (Board B) is used for this communication.
- Sensors & Actuators Integrated:
  - VL53L0X TOF Sensor: Measures water levels to control the pump and valve during liquid filling stages.
  - Ultrasonic Sensor: Detects object presence in the liquid container, ensuring proper system functionality.
  - Dallas Temperature Sensor: Monitors the temperature for accurate chemical processes.
  - Water Pump & Solenoid Valve: Controlled to fill liquids to a specific volume based on sensor data.

Key Libraries & Hardware Interfaces:

- esp_now.h & WiFi.h: Handles ESP-NOW communication.
- Wire.h & LiquidCrystal_I2C.h: For communicating with an I2C-based LCD display for user interactions.
- Keypad.h: Manages keypad input, allowing the user to select options or input concentration values.
- DallasTemperature.h: Reads temperature data from the DS18B20 sensor.
- Adafruit_VL53L0X.h: Manages the TOF sensor used for accurate water level measurements.

Input-Output Pins:

- Relay Pins:
  - PUMP_RELAY_PIN: Connected to pin 23, controls the water pump.
  - VALVE_RELAY_PIN: Connected to pin 19, controls the solenoid valve for liquid flow.
- Ultrasonic Sensor:
  - Trig Pin: Pin 16
  - Echo Pin: Pin 17

- Keypad Pin Assignment:
    - Row pins: 12, 13, 14, 27
    - Column pins: 26, 25, 33, 32
- Temperature Sensor (DS18B20):
    - Pin 4 (One-Wire Bus)

Core Functionalities:

A. User Input & Interaction:

1. Keypad Input:
    - The system accepts user input via a 4x4 matrix keypad. It is used to:
        - Set input concentration (Input C) and output concentration (Output C).
        - Toggle between various system states.
        - Reset the circuit when necessary.
    - Special characters:
        - #: Confirms the user's input and advances the system state.
        - *: Clears entered values.
        - D: Resets the entire system.
2. LCD Display:
    - Custom Characters: Degree symbols, arrows, and progress bars are used for displaying information on the I2C LCD screen.
    - Feedback Display: Displays real-time data such as water levels, concentrations, and warnings like "Max Water Level Reached" or "Input Vo Out of Range."

B. Water Level Measurement & Control:

1. VL53L0X TOF Sensor:
    - The Time-of-Flight sensor measures the water level in the container and applies a refractive index correction for accurate readings.
    - The system averages multiple samples to ensure reliable measurements.
    - If the water level falls below the required threshold, it activates the water pump and valve to fill the liquid to the required volume.
2. Water Filling Control:
    - The system calculates the required water volume based on user input and fills the container using the water pump.
    - The pump is controlled by relays, turning on and off depending on the water level.
    - A maximum water level is set to avoid overfilling.

C. Dilution Process:

1. Input C & Output C:
    - Input C: User inputs the concentration of the initial solution.
    - Output C: User inputs the desired concentration after dilution.
    - Based on these inputs, the system calculates the required volume (Vo) for dilution.
2. Volume Calculation:
    - The system calculates the necessary volume based on the formula: Output V = (Input C * 4) / Output C.
    - The calculated volume is displayed, and any errors, such as volumes outside a safe range, trigger warnings.

D. Temperature Measurement:

1. DS18B20 Temperature Sensor:
    - The system monitors temperature during the chemical process, ensuring safety and accuracy.
    - The temperature is displayed on the LCD screen and recorded for system logs.

E. Ultrasonic Sensor for Object Detection:

1. Ultrasonic Object Detection:
    - Used to detect if the output beaker is correctly placed.
    - Once the object (beaker) is detected, the system proceeds with the water filling or chemical mixing stages.

F. Safety Mechanisms:

1. Warning System:
    - The system triggers visual and audible warnings when certain safety thresholds are breached (e.g., low water level, high concentration errors).
    - The warning messages are displayed on the LCD, and specific LEDs (Red, Yellow, Green) are activated based on the severity.
2. Reset Function:
    - The system includes a reset mechanism where pressing the D key or encountering a critical error trigger a system reset using esp_restart().

Data Handling & Communication:

- The system maintains a struct_message data structure that holds important state flags and variables such as gyroscope data, warning indicators, motor status, etc.
- Data Transmission:
    - Data is sent to the second ESP32 board every 1 second using the ESP-NOW protocol. The data includes vital information like gyroscope readings, water status, and error flags.

Example Communication Flow:

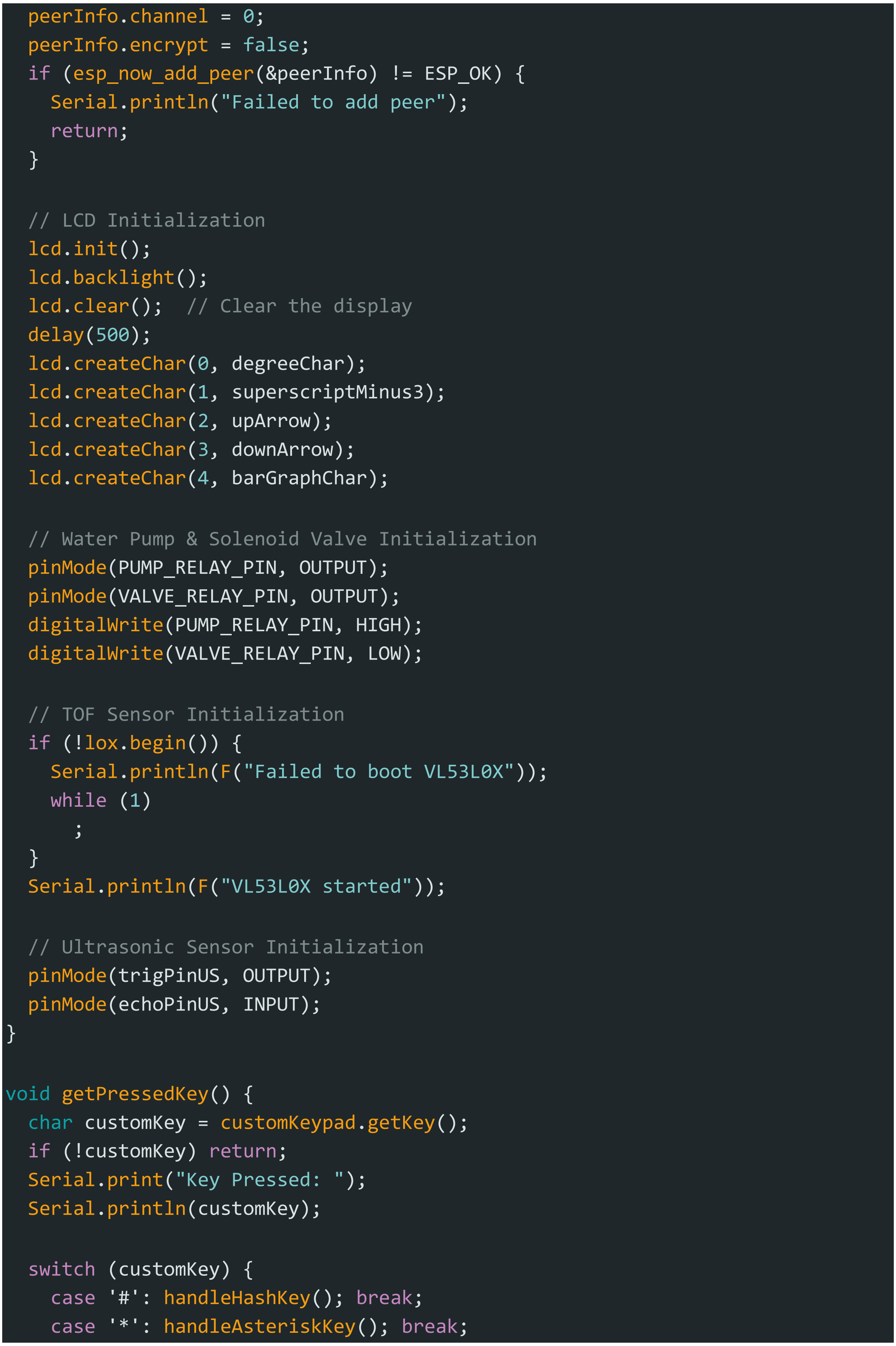
1. The first ESP32 sends data like gyroscope state, water level status, and warnings to the second ESP32.
2. It receives acknowledgment from the second ESP32, which confirms actions like completing a process or triggering an alarm.

ESP 32 Board A – Code

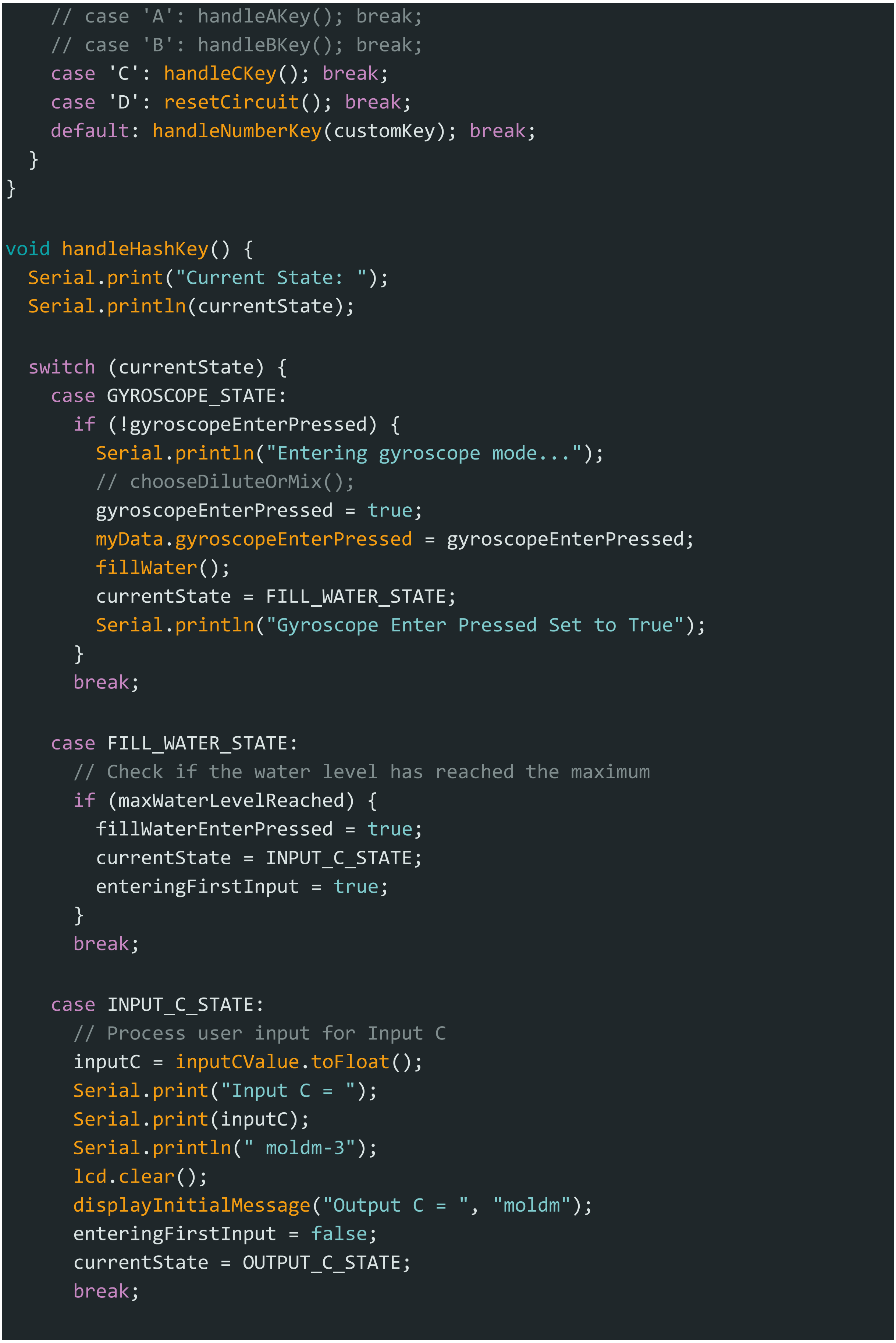
```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
#include <DallasTemperature.h>
#include <Adafruit_VL53L0X.h>
#include <esp_system.h>

// ESP32---------------------------------------------------------
// MAC address of the reciever ESP32 - board b
uint8_t espBoardB_address[] = { 0xB0, 0xB2, 0x1C, 0x97, 0x6D, 0xB4 };

typedef struct struct_message {
  int angleX, angleY, angleZ;  // Gyroscope data
  bool gyroscopeDataSent;      // Flag to indicate if gyroscope data was sent
  bool gyroscopeEnterPressed;
  bool chooseOptionsLoaded;
  bool outputCheckCalled;
  bool ultrasonicObjectDetected;
  bool waterFilled;  // water part completion condition
  bool motorPartCompleted;
  bool warningCalled, buzzerCalled, redLEDCalled, yellowLEDCalled,
greenLEDCalled;
  String warningMessage;
  bool resetCalled;
  bool callRED;
  bool liquidUltrasonicCalled;
} struct_message;

struct_message myData;

// LCD and Keypad Initialization
LiquidCrystal_I2C lcd(0x27, 16, 2);

const byte ROWS = 4;
const byte COLS = 4;
char hexaKeys[ROWS][COLS] = {
  { 'D', '#', '0', '*' },
  { 'C', '9', '8', '7' },
  { 'B', '6', '5', '4' },
  { 'A', '3', '2', '1' }
};
byte rowPins[ROWS] = { 12, 13, 14, 27 };
byte colPins[COLS] = { 26, 25, 33, 32 };
```

```
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);

// Custom Characters
byte degreeChar[8] = { 0b00111, 0b00101, 0b00111, 0b00000, 0b00000, 0b00000,
0b00000, 0b00000 };
byte superscriptMinus3[8] = { 0b00111, 0b00001, 0b11011, 0b00001, 0b00111,
0b00000, 0b00000, 0b00000 };
byte upArrow[8] = { 0b00100, 0b01110, 0b11111, 0b11111, 0b01110, 0b01110,
0b01110, 0b01110 };
byte downArrow[8] = { 0b01110, 0b01110, 0b01110, 0b01110, 0b11111, 0b11111,
0b01110, 0b00100 };
byte barGraphChar[8] = { 0b11111, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111,
0b11111, 0b11111 };

// TEMPERATURE SENSOR------------------------------------------
#define ONE_WIRE_BUS 4
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

// Input Output Variables--------------------------------------
bool completed = false, warningIndication = false;
String inputCValue = "", outputCValue = "";
bool enteringFirstInput = true, inputCHasDecimal = false, outputCHasDecimal =
false, finalOutputIsDisplayed = false;
float inputC = 0.0, outputC = 0.0, temp = 0.0;
double outputV = 0.0;
char outputVStr[10];

bool gyroscopeEnterPressed = false, tofThresholdDetected = false,
maxWaterLevelReached = true, fillWaterEnterPressed = false, resetCalled =
false, mixSecondStage = false, tempRunning = false;

// ULTRASONIC Variables------------------------------------
const int trigPinUS = 16, echoPinUS = 17;
long duration;
float distance;

// Water Variables-------------------------------------------
const float minWaterLevel = 29.5, maxWaterLevel = 9.0;
#define PUMP_RELAY_PIN 23
#define VALVE_RELAY_PIN 19
Adafruit_VL53L0X lox = Adafruit_VL53L0X();
// Refractive index of water
const float refractiveIndex = 1.333;
bool waterFilled = false;
```

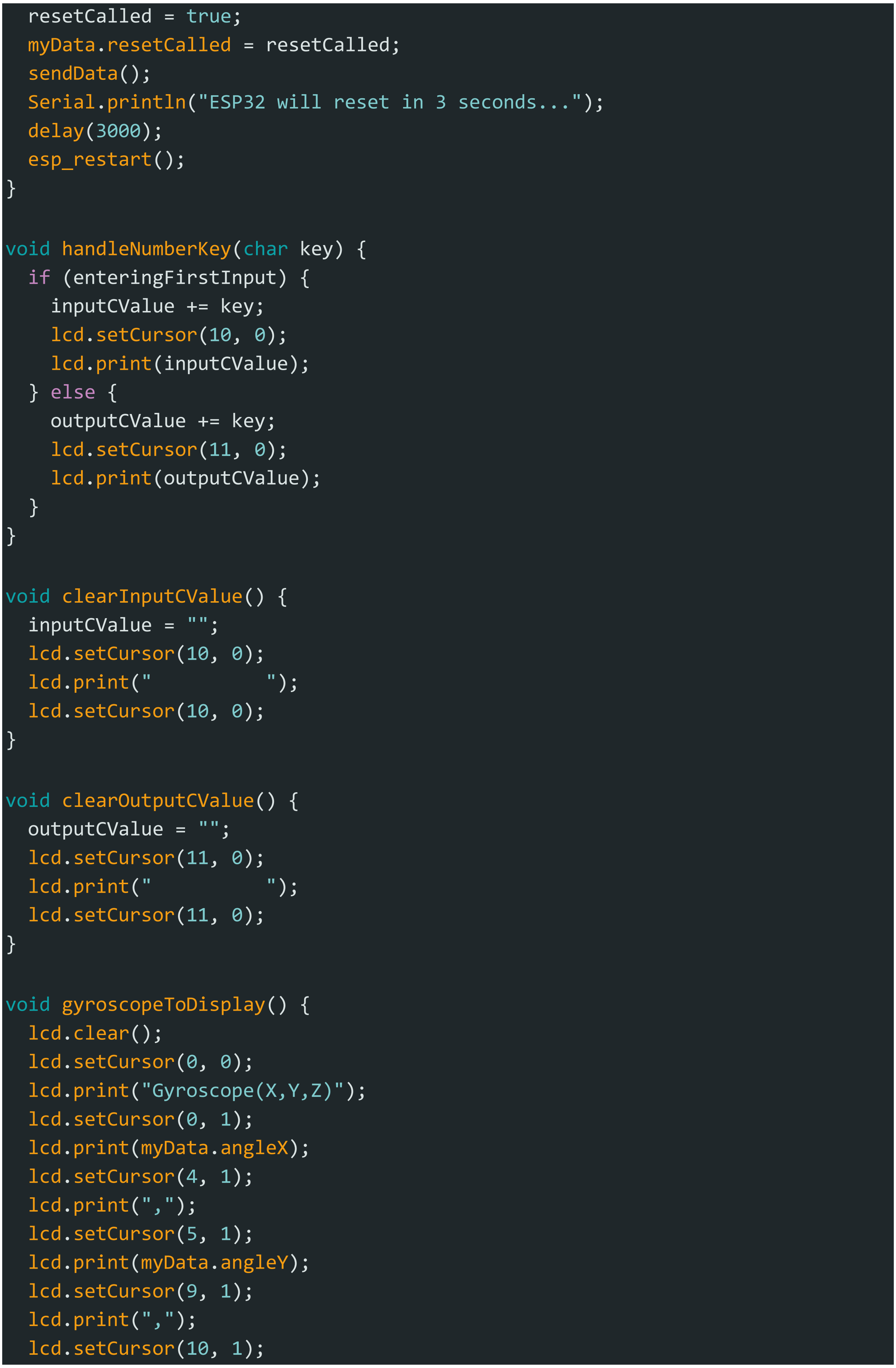
```
// Enumeration for states
enum State {
  GYROSCOPE_STATE,
  INPUT_C_STATE,
  INPUT_V_STATE,
  OUTPUT_C_STATE,
  CHOOSE_DILUTE_OR_MIX_STATE,
  FILL_WATER_STATE,
  HANDLE_DILUTE_OPTION,
  HANDLE_MIX_OPTION,
  CHECK_ULTRASONIC_SENSOR,
  MIX_PLACE_BEAKERS_STATE
  // Add more states as needed (for further development)
};
State currentState = GYROSCOPE_STATE;

// Function Declarations
void initializeComponents();
void getPressedKey();
void handleHashKey();
void handleAsteriskKey();
void handleAKey();
void handleBKey();
void handleCKey();
void resetCircuit();
void handleNumberKey(char key);
void clearInputCValue();
void clearOutputCValue();
void gyroscopeToDisplay();
void chooseDiluteOrMix();
void mixToDisplay();
void ultrasonicToDisplay();
float tofMeasureWaterLevel();
void runMeasureSendWaterVolume();
void fillWater();
void displayInitialMessage(String topRowMessage, String bottomRowMessage);
void displayOutputV();
void runTemp();
void runUltrasonic();
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status);
void OnDataRecv(const esp_now_recv_info *recv_info, const uint8_t
*incomingData, int len);
void sendData();
void warning(const char *message);

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

void loop() {
  getPressedKey();

  if (finalOutputIsDisplayed && !myData.motorPartCompleted) {
    // Serial.println("Final output was displayed");
    myData.outputCheckCalled = true;
    if (!waterFilled) {
      ultrasonicToDisplay();
    } else {
      runTemp();
      tempRunning = true;  // Set flag when temperature function starts
    }
  }
  if (myData.ultrasonicObjectDetected && !waterFilled) {
    Serial.println("Output beaker detected");
    runMeasureSendWaterVolume();
  }
  if (myData.waterFilled && !myData.motorPartCompleted &&
myData.liquidUltrasonicCalled) {
    Serial.println("TOF threshold detected");
    runUltrasonic();
  }

  static unsigned long lastSendTime = 0;
  unsigned long currentTime = millis();
  if (currentTime - lastSendTime > 1000) {
    sendData();
    lastSendTime = currentTime;
  }
}

void initializeComponents() {
  // ESP32 Initialization
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_register_send_cb(OnDataSent);
  esp_now_register_recv_cb(OnDataRecv);
  esp_now_peer_info_t peerInfo;
  memset(&peerInfo, 0, sizeof(peerInfo));
  memcpy(peerInfo.peer_addr, espBoardB_address, 6);
```



```
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  // LCD Initialization
  lcd.init();
  lcd.backlight();
  lcd.clear();  // Clear the display
  delay(500);
  lcd.createChar(0, degreeChar);
  lcd.createChar(1, superscriptMinus3);
  lcd.createChar(2, upArrow);
  lcd.createChar(3, downArrow);
  lcd.createChar(4, barGraphChar);

  // Water Pump & Solenoid Valve Initialization
  pinMode(PUMP_RELAY_PIN, OUTPUT);
  pinMode(VALVE_RELAY_PIN, OUTPUT);
  digitalWrite(PUMP_RELAY_PIN, HIGH);
  digitalWrite(VALVE_RELAY_PIN, LOW);

  // TOF Sensor Initialization
  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1)
      ;
  }
  Serial.println(F("VL53L0X started"));

  // Ultrasonic Sensor Initialization
  pinMode(trigPinUS, OUTPUT);
  pinMode(echoPinUS, INPUT);
}

void getPressedKey() {
  char customKey = customKeypad.getKey();
  if (!customKey) return;
  Serial.print("Key Pressed: ");
  Serial.println(customKey);

  switch (customKey) {
    case '#': handleHashKey(); break;
    case '*': handleAsteriskKey(); break;
```



```
    // case 'A': handleAKey(); break;
    // case 'B': handleBKey(); break;
    case 'C': handleCKey(); break;
    case 'D': resetCircuit(); break;
    default: handleNumberKey(customKey); break;
  }
}

void handleHashKey() {
  Serial.print("Current State: ");
  Serial.println(currentState);

  switch (currentState) {
    case GYROSCOPE_STATE:
      if (!gyroscopeEnterPressed) {
        Serial.println("Entering gyroscope mode...");
        // chooseDiluteOrMix();
        gyroscopeEnterPressed = true;
        myData.gyroscopeEnterPressed = gyroscopeEnterPressed;
        fillWater();
        currentState = FILL_WATER_STATE;
        Serial.println("Gyroscope Enter Pressed Set to True");
      }
      break;

    case FILL_WATER_STATE:
      // Check if the water level has reached the maximum
      if (maxWaterLevelReached) {
        fillWaterEnterPressed = true;
        currentState = INPUT_C_STATE;
        enteringFirstInput = true;
      }
      break;

    case INPUT_C_STATE:
      // Process user input for Input C
      inputC = inputCValue.toFloat();
      Serial.print("Input C = ");
      Serial.print(inputC);
      Serial.println(" moldm-3");
      lcd.clear();
      displayInitialMessage("Output C = ", "moldm");
      enteringFirstInput = false;
      currentState = OUTPUT_C_STATE;
      break;
    case OUTPUT_C_STATE:
      // Process user input for Output C
      if (!enteringFirstInput) {
        outputC = outputCValue.toFloat();
        Serial.print("Output C = ");
        Serial.print(outputC);
        Serial.println(" moldm-3");
        lcd.clear();
        displayOutputV();
        currentState = CHECK_ULTRASONIC_SENSOR;
      }
      break;

    // Add more cases for additional states if needed (for further development)
    default:
      break;
  }
}

void handleAsteriskKey() {
  if (inputCValue.length() > 0 && outputCValue.length() == 0) {
    clearInputCValue();
  } else if (inputCValue.length() > 0 && outputCValue.length() > 0) {
    clearOutputCValue();
  }
}

void handleCKey() {
  if (enteringFirstInput) {
    if (!inputCHasDecimal) {
      inputCValue += ".";
      inputCHasDecimal = true;
      lcd.setCursor(10 + inputCValue.length() - 1, 0);
      lcd.print(".");
    }
  } else {
    if (!outputCHasDecimal) {
      outputCValue += ".";
      outputCHasDecimal = true;
      lcd.setCursor(11 + outputCValue.length() - 1, 0);
      lcd.print(".");
    }
  }
}

void resetCircuit() {
```

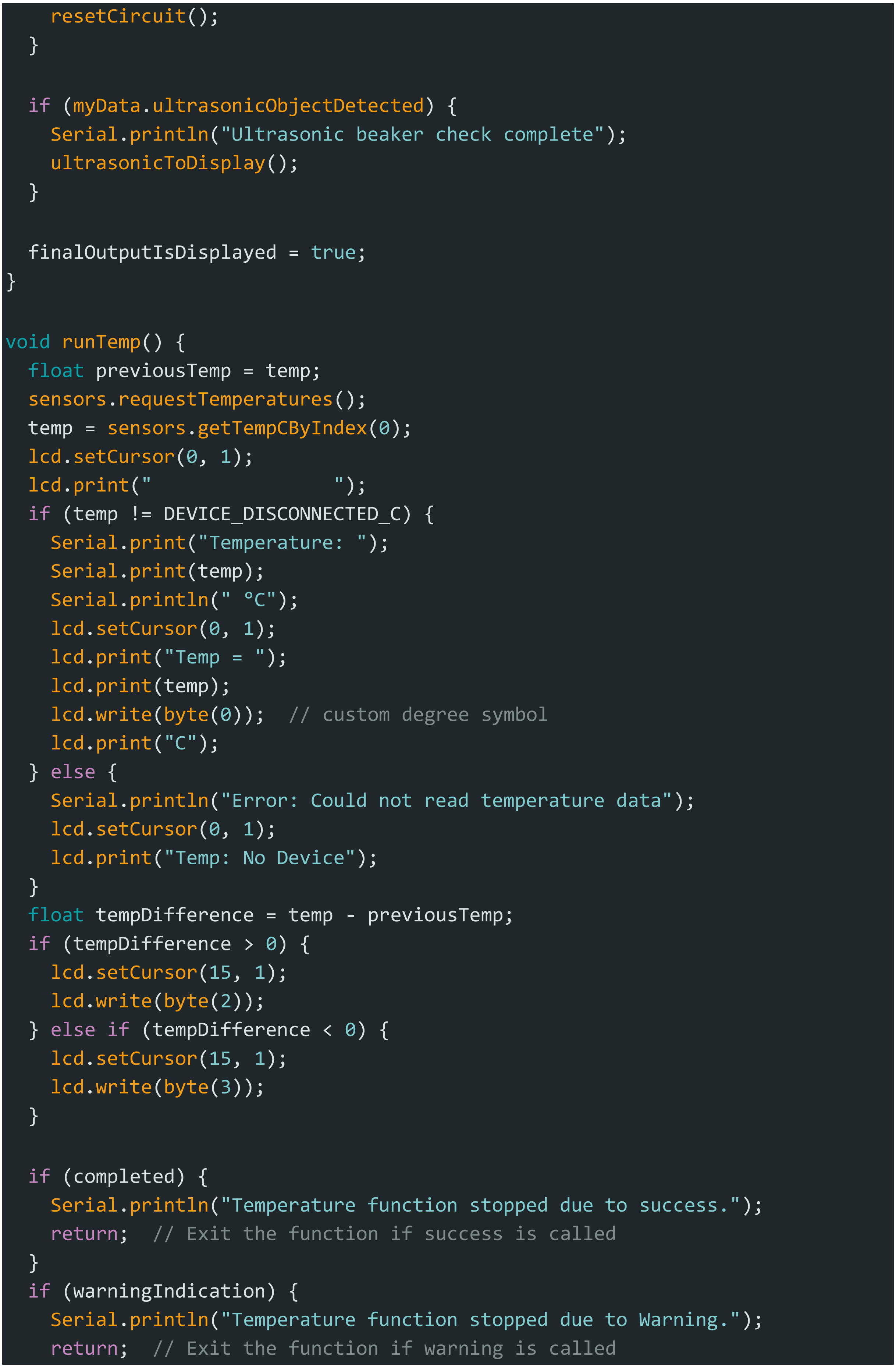


```
  resetCalled = true;
  myData.resetCalled = resetCalled;
  sendData();
  Serial.println("ESP32 will reset in 3 seconds...");
  delay(3000);
  esp_restart();
}

void handleNumberKey(char key) {
  if (enteringFirstInput) {
    inputCValue += key;
    lcd.setCursor(10, 0);
    lcd.print(inputCValue);
  } else {
    outputCValue += key;
    lcd.setCursor(11, 0);
    lcd.print(outputCValue);
  }
}

void clearInputCValue() {
  inputCValue = "";
  lcd.setCursor(10, 0);
  lcd.print("          ");
  lcd.setCursor(10, 0);
}

void clearOutputCValue() {
  outputCValue = "";
  lcd.setCursor(11, 0);
  lcd.print("          ");
  lcd.setCursor(11, 0);
}

void gyroscopeToDisplay() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Gyroscope(X,Y,Z)");
  lcd.setCursor(0, 1);
  lcd.print(myData.angleX);
  lcd.setCursor(4, 1);
  lcd.print(",");
  lcd.setCursor(5, 1);
  lcd.print(myData.angleY);
  lcd.setCursor(9, 1);
  lcd.print(",");
  lcd.setCursor(10, 1);
  lcd.print(myData.angleZ);
}

void ultrasonicToDisplay() {
  lcd.setCursor(0, 1);
  // lcd.print("                ");
  lcd.print("Beaker detected");
}

float tofMeasureWaterLevel() {
  const int numSamples = 30;
  float totalDistance = 0;
  float averageDistance = 0;
  int validSamples = 0;

  for (int i = 0; i < numSamples; i++) {
    VL53L0X_RangingMeasurementData_t measure;

    // Perform the measurement
    lox.rangingTest(&measure, false);

    if (measure.RangeStatus != 4) {                        // phase failures
have incorrect data
      float distance = measure.RangeMilliMeter / 10.0;       // Convert to cm
      float correctedDistance = distance * refractiveIndex;  // Apply
refraction correction
      totalDistance += correctedDistance;
      validSamples++;
    }

    delay(10);  // Delay to achieve approximately 100 samples per second
  }

  if (validSamples > 0) {
    averageDistance = totalDistance / validSamples;
    Serial.print("Average corrected distance: ");
    Serial.print(averageDistance);
    Serial.println(" cm");
  } else {
    Serial.println("No valid samples");
  }

  return averageDistance;

  delay(1000 - (numSamples * 10));  // Adjust delay to complete one second
cycle
```

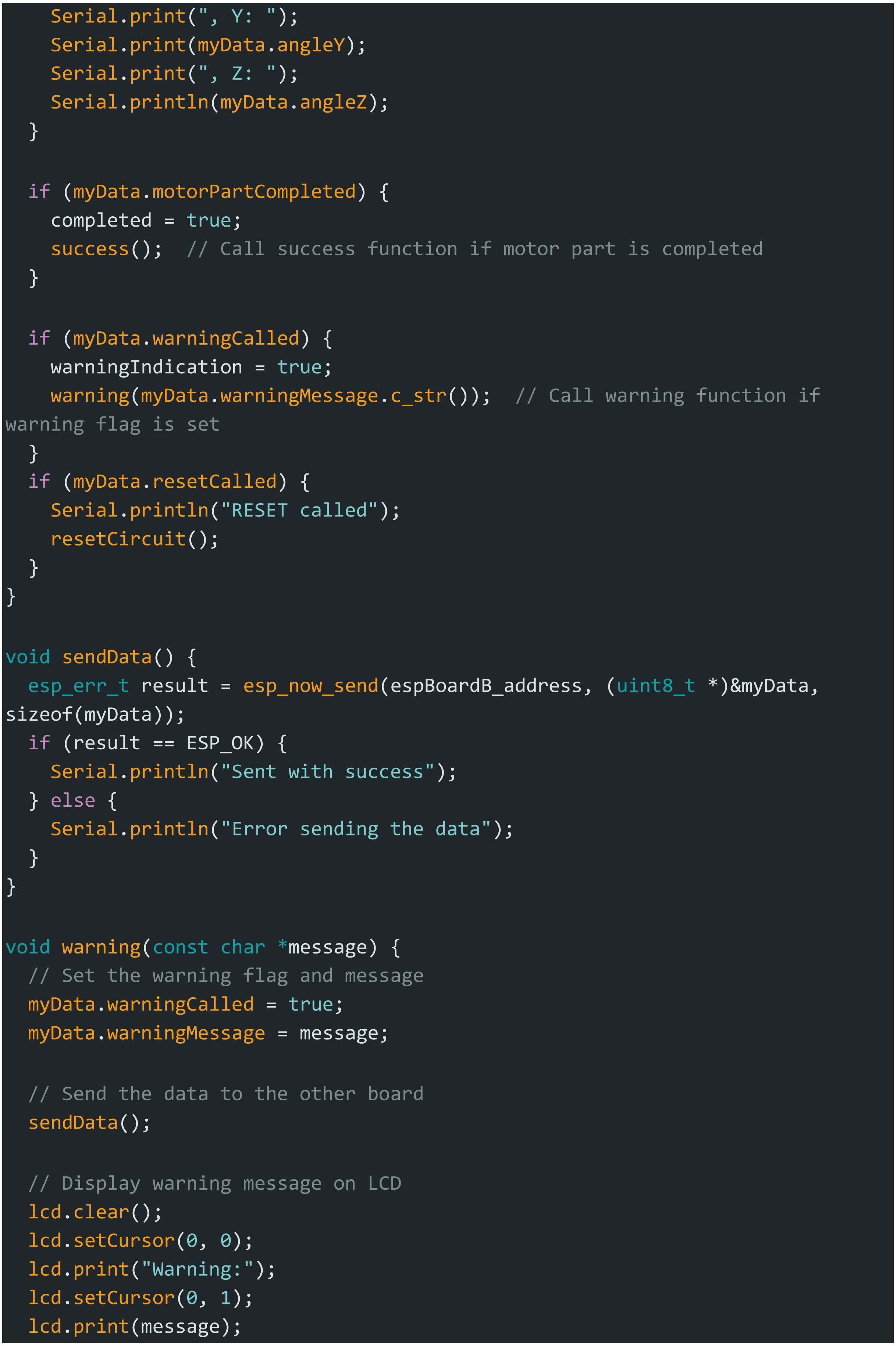
```
}

void runMeasureSendWaterVolume() {
  float currentWaterLevel = tofMeasureWaterLevel();
  float currentVolume = (currentWaterLevel - maxWaterLevel) * 1.694915;
  float pumpRate = 13;  // Pump rate in ml/s

  Serial.print("Current Water Level (mm): ");
  Serial.println(currentWaterLevel);
  Serial.print("Current Water Volume (ml): ");
  Serial.println(currentVolume);

  if (currentWaterLevel >= minWaterLevel) {
    Serial.println("Water volume is not enough");
    warningIndication = true;
    warning("Low Water Level");
    digitalWrite(PUMP_RELAY_PIN, HIGH);
    digitalWrite(VALVE_RELAY_PIN, HIGH);
    while (1)
      ;
  }

  if (outputV > 0 && waterFilled == false) {
    int pumpTime = (outputV - 4) / pumpRate;  // Calculate the time in seconds to pump the desired volume

    // Activate water pump and solenoid valve
    digitalWrite(PUMP_RELAY_PIN, LOW);
    digitalWrite(VALVE_RELAY_PIN, HIGH);

    // Wait for the calculated time
    delay(pumpTime * 1000);  // Convert seconds to milliseconds

    // Deactivate water pump and close solenoid valve
    digitalWrite(PUMP_RELAY_PIN, HIGH);
    digitalWrite(VALVE_RELAY_PIN, LOW);

    waterFilled = true;
    myData.waterFilled = waterFilled;
    Serial.println("Threshold reached from pump, stopping pump and closing valve");
    sendData();
  }
  delay(100);
}
```

```
void fillWater() {
  lcd.clear();
  lcd.print("Fill all Liquids");

  unsigned long startTime = millis();  // Start time for timeout logic
  float currentWaterLevel = tofMeasureWaterLevel();
  int currentVolume = 0;

  while (true) {
    currentWaterLevel = tofMeasureWaterLevel();
    currentVolume = (minWaterLevel - currentWaterLevel) * 1.694915;

    if (currentWaterLevel == -1) {
      lcd.clear();
      lcd.print("Error: Out of range");
      Serial.println("Error: ToF sensor out of range");
      warningIndication = true;
      warning("Error: ToF sensor");
      break;
    }

    int fillAmount = ((minWaterLevel - currentWaterLevel) / (minWaterLevel - 
maxWaterLevel)) * 16;

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Water Level:");
    for (int i = 0; i <= fillAmount; i++) {
      lcd.setCursor(i, 1);
      lcd.write(byte(4));
    }
    delay(2000);

    if (currentWaterLevel <= maxWaterLevel) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Max Water Level");
      lcd.setCursor(0, 1);
      lcd.print("Reached!");
      Serial.println("Max water level reached.");
      maxWaterLevelReached = true;
      delay(5000);

      if (!waterFilled) {
        // Check for timeout (e.g., 2 minutes)
        if (millis() - startTime > 300000) {  // 300000 ms = 5 minutes
```

```
          Serial.println("Timeout: Max water level not reached");
          warningIndication = true;
          warning("Low Water Level");
          fillWater();
          break;
        }
      }

      // Transition to the INPUT_C_STATE
      displayInitialMessage("Input C = ", "moldm");
      break;
    }
  }
}

void displayInitialMessage(String topRowMessage, String bottomRowMessage) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(topRowMessage);
  if (bottomRowMessage.length() > 0) {
    lcd.setCursor(0, 1);
    lcd.print("(");
    lcd.print(bottomRowMessage);
    lcd.write(byte(1));
    lcd.print(")");
  }
}

void displayOutputV() {
  lcd.clear();
  outputV = (inputC * 4) / outputC;
  dtostrf(outputV, 9, 5, outputVStr);
  lcd.setCursor(0, 0);
  lcd.print("Vo = ");
  lcd.print(outputVStr);
  lcd.print("ml");
  Serial.print("Output V = ");
  Serial.print(outputV);
  Serial.println("ml");

  if (outputV > 225 || outputV < 50) {
    warningIndication = true;
    warning("Vo Out of Range");
    myData.callRED = true;
    sendData();
    delay(5000);
    myData.resetCalled = true;
```



```
    resetCircuit();
  }

  if (myData.ultrasonicObjectDetected) {
    Serial.println("Ultrasonic beaker check complete");
    ultrasonicToDisplay();
  }

  finalOutputIsDisplayed = true;
}

void runTemp() {
  float previousTemp = temp;
  sensors.requestTemperatures();
  temp = sensors.getTempCByIndex(0);
  lcd.setCursor(0, 1);
  lcd.print("                ");
  if (temp != DEVICE_DISCONNECTED_C) {
    Serial.print("Temperature: ");
    Serial.print(temp);
    Serial.println(" °C");
    lcd.setCursor(0, 1);
    lcd.print("Temp = ");
    lcd.print(temp);
    lcd.write(byte(0));  // custom degree symbol
    lcd.print("C");
  } else {
    Serial.println("Error: Could not read temperature data");
    lcd.setCursor(0, 1);
    lcd.print("Temp: No Device");
  }
  float tempDifference = temp - previousTemp;
  if (tempDifference > 0) {
    lcd.setCursor(15, 1);
    lcd.write(byte(2));
  } else if (tempDifference < 0) {
    lcd.setCursor(15, 1);
    lcd.write(byte(3));
  }

  if (completed) {
    Serial.println("Temperature function stopped due to success.");
    return;  // Exit the function if success is called
  }
  if (warningIndication) {
    Serial.println("Temperature function stopped due to Warning.");
    return;  // Exit the function if warning is called
  }

  delay(100);
}

void runUltrasonic() {
  digitalWrite(trigPinUS, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinUS, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinUS, LOW);
  duration = pulseIn(echoPinUS, HIGH);
  distance = duration * 0.034 / 2;
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  if (distance > 6.5) {
    warningIndication = true;
    myData.redLEDCalled = true;
    warning("No Liquid Detected");
  }
  delay(1000);
}

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\nLast Packet Send Status: ");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

void OnDataRecv(const esp_now_recv_info *recv_info, const uint8_t
*incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Bytes received: ");
  Serial.println(len);

  if (gyroscopeEnterPressed) {
    Serial.println("Gyroscope Finished");
  } else {
    gyroscopeToDisplay();
  }

  if (myData.gyroscopeDataSent) {
    // Handle gyroscope data
    Serial.print("Received Gyroscope Data - X: ");
    Serial.print(myData.angleX);
```



```
    Serial.print(", Y: ");
    Serial.print(myData.angleY);
    Serial.print(", Z: ");
    Serial.println(myData.angleZ);
  }

  if (myData.motorPartCompleted) {
    completed = true;
    success();  // Call success function if motor part is completed
  }

  if (myData.warningCalled) {
    warningIndication = true;
    warning(myData.warningMessage.c_str());  // Call warning function if
warning flag is set
  }
  if (myData.resetCalled) {
    Serial.println("RESET called");
    resetCircuit();
  }
}

void sendData() {
  esp_err_t result = esp_now_send(espBoardB_address, (uint8_t *)&myData,
sizeof(myData));
  if (result == ESP_OK) {
    Serial.println("Sent with success");
  } else {
    Serial.println("Error sending the data");
  }
}

void warning(const char *message) {
  // Set the warning flag and message
  myData.warningCalled = true;
  myData.warningMessage = message;

  // Send the data to the other board
  sendData();

  // Display warning message on LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Warning:");
  lcd.setCursor(0, 1);
  lcd.print(message);
  // Send warning message to the other ESP32 board
  struct_message warningData = myData;
  esp_now_send(espBoardB_address, (uint8_t *)&warningData,
sizeof(warningData));

  // Keep the warning state for 5 seconds
  delay(5000);

  // turn off the warning flag
  myData.warningCalled = false;

  // Reset the tempRunning flag
  tempRunning = false;
  // warningIndication = false;

  delay(5000);
  myData.resetCalled = true;
  resetCircuit();
}

void success() {
  // Display warning message on LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Success! Press");
  lcd.setCursor(0, 1);
  lcd.print("Reset to redo");

  // Keep the success state for 5 seconds
  delay(5000);

  // Reset the tempRunning flag
  tempRunning = false;
}
```

ESP32 Board B - Overview & Functions

Main Components & Features:

- ESP-NOW Protocol: This board communicates wirelessly with the first ESP32 board using the ESP-NOW protocol. It receives data from the first board and processes specific actions like monitoring gyroscope movements, triggering alarms, and controlling auxiliary safety systems.
- Gyroscope Integration:
  - MPU6050 Gyroscope: Monitors the angular position and orientation of the system to ensure safety and stability during operations, particularly in liquid handling processes.
- Safety Indicators:
  - LED Warnings: The board controls Red, Yellow, and Green LEDs to provide visual feedback on system status and warnings based on the data received from the first board.

Key Libraries & Hardware Interfaces:

- esp_now.h & WiFi.h: Handles ESP-NOW communication with the first ESP32 board.
- Wire.h: Interfaces with the gyroscope sensor (MPU6050) over I2C communication.
- Adafruit_MPU6050.h: Used for accessing the gyroscope readings for real-time monitoring of the system's orientation and angular velocity.
- LedControl.h: Manages the LEDs that act as visual indicators for system warnings and statuses.

Input-Output Pins:

- LED Control:
  - RED_LED_PIN: Pin 15
  - YELLOW_LED_PIN: Pin 2
  - GREEN_LED_PIN: Pin 4
- Buzzer: Pin 21
- MPU6050 Gyroscope:
  - SCL Pin: Pin 22
  - SDA Pin: Pin 21 (I2C Communication)

Core Functionalities:

A. Data Reception & Processing:

1. ESP-NOW Data Reception:
   - The second ESP32 board receives periodic updates (every 1 second) from the first ESP32 board. This data includes critical information like water levels, sensor warnings, gyroscope data, and system status flags.
   - The data structure (struct_message) contains fields like:
     - GyroX, GyroY, GyroZ: Gyroscope readings from the MPU6050.
     - Water level status: Indicates if the water is below the required level.
     - Error/Warning flags: Warnings from the first board, such as max water levels, concentration errors, or system malfunctions.
2. Error Handling:
   - The board evaluates the received error or warning flags and determines the system's response.
   - Based on this data, it activates the appropriate safety indicators (LEDs or buzzer) to alert the user about potential issues.

B. Safety Monitoring & Gyroscope Feedback:

1. MPU6050 Gyroscope Monitoring:
   - The MPU6050 sensor continuously monitors the angular velocity and position of the system.
   - If the system tilts beyond safe thresholds, the board triggers safety mechanisms, including warnings or halting liquid processes to avoid spills or system damage.
2. Tilt Detection & Action:
   - The board processes real-time data from the gyroscope to detect any abnormal tilting or motion. If an unsafe tilt is detected (above predefined safe angles for the system), the board:
     - Sends a warning signal back to the first ESP32.
     - Activates an emergency stop mechanism for pumps and valves.
     - Lights the Red LED to indicate critical instability.

C. Visual Indicators & Alarm System:

1. LED Status Indications:
   - Red LED (Pin 15):
     - Lit when a critical error or instability is detected, such as:
       - Excessive tilt or system instability from the gyroscope.

            - Major system warnings received from the first ESP32 (e.g., maximum water level exceeded).
    - Yellow LED (Pin 2):
        - Lit during system warnings such as:
            - High concentration errors or out-of-range input/output values.
            - Non-critical warnings that require attention but do not stop the system.
    - Green LED (Pin 4):
        - Lit when the system is operating normally without errors or warnings.
2. Buzzer Alarms:
    - Buzzer (Pin 21):
        - Activated when the system encounters critical issues, such as dangerous tilt angles, maximum water level, or faulty sensor readings.
        - Sounds in conjunction with the Red LED to provide an audible alarm to notify operators of immediate attention.

D. Communication Flow with ESP32 Board A:

1. Data Transmission:
    - The second ESP32 board sends acknowledgment messages back to the first board when actions are completed, or warnings have been triggered.
    - Gyroscope data, error states, and LED/buzzer status are shared between the two boards to ensure synchronized operation and error handling.
2. Real-Time Updates:
    - The two ESP32 boards exchange information every second, ensuring that any error detected by one board is immediately relayed to the other.
    - Board B informs Board A of any critical errors like tilt detection, which causes Board A to halt the liquid processing functions if needed.

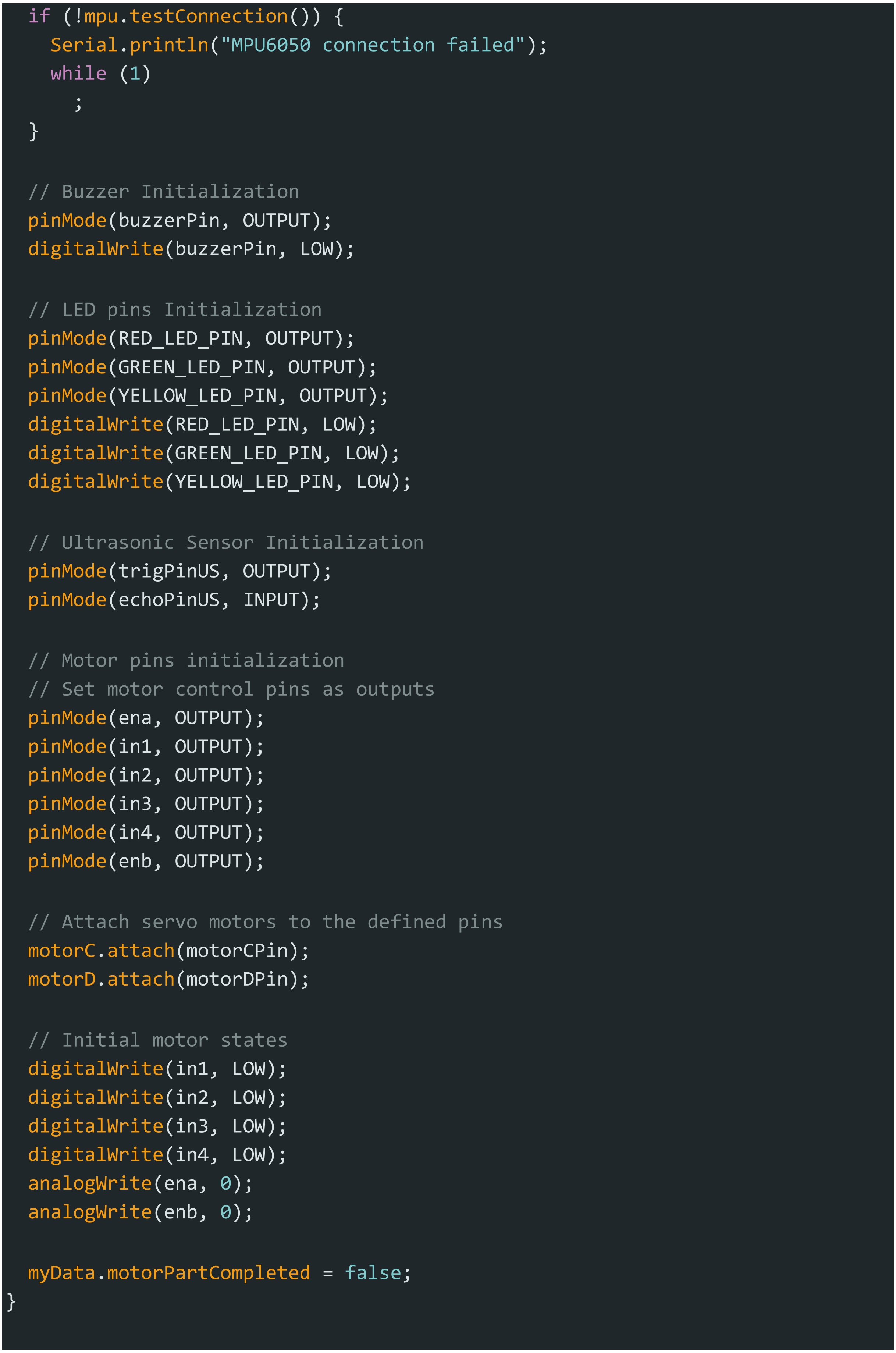
E. Reset & Error Recovery:

1. Reset Mechanism:
    - The board is programmed to reset its operation if a severe fault is detected, ensuring that the system can recover from unexpected failures.
    - Manual reset can also be triggered by Board A when critical errors are flagged.
2. Error Logging:
    - Any major errors like tilt events or sensor malfunctions are logged and displayed through the LED system. The Red LED stays lit until the error is resolved.

ESP 32 Board B – Code

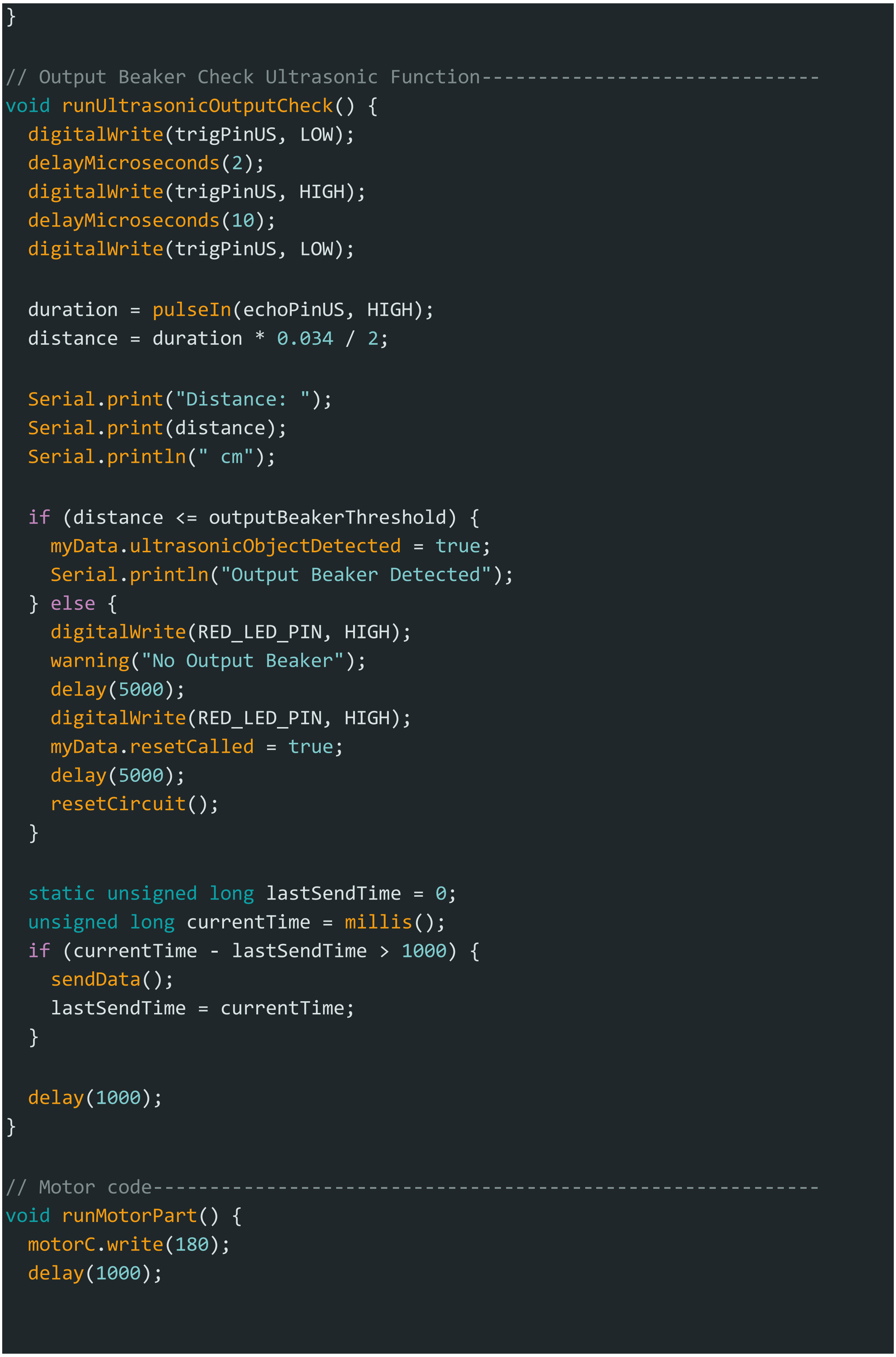
```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <MPU6050.h>
#include <ESP32Servo.h>
#include <Arduino.h>
#include <driver/ledc.h>
#include <esp_system.h>

// ESP32------------------------------------------------------
// MAC address of the receiver ESP32 - board a
uint8_t espBoardA_address[] = { 0xA0, 0xA3, 0xB3, 0x2A, 0xDE, 0x3C };

// Structure to hold data to be sent
typedef struct struct_message {
  int angleX, angleY, angleZ;  // Gyroscope data
  bool gyroscopeDataSent;      // Flag to indicate if gyroscope data was sent
  bool gyroscopeEnterPressed;
  bool chooseOptionsLoaded;
  bool outputCheckCalled;
  bool ultrasonicObjectDetected;
  bool waterFilled;  // water part completion condition
  bool motorPartCompleted;
  bool warningCalled, buzzerCalled, redLEDCalled, yellowLEDCalled,
greenLEDCalled;
  String warningMessage;
  bool resetCalled;
  bool callRED;
  bool liquidUltrasonicCalled;
} struct_message;

struct_message myData;

float outputBeakerThreshold = 10.0;

// Gyroscope-----------------------------------------------
MPU6050 mpu;

// ULTRASONIC Variables------------------------------------
const int trigPinUS = 12;
const int echoPinUS = 13;

// Variables to store the duration of the pulse and the distance
long duration;
```

```
float distance;

// Buzzer and LED Pins---------------------------------------
#define buzzerPin 33
#define RED_LED_PIN 26
#define GREEN_LED_PIN 16
#define YELLOW_LED_PIN 25

// Define sound patterns
const int EMERGENCY_SOUND = 0;
const int AFTER_WORK_SOUND = 1;
const int CLEANING_TIME_SOUND = 2;

// Water Variables------------------------------------------
bool waterFilled = false;

// Motor Part Variables---------------------------------------
// Define motor control pins for Motor A and B (connected to the first motor
driver)
int ena = 19;  // Motor A enable pin
int in1 = 18;  // Motor A input 1
int in2 = 5;   // Motor A input 2
int in3 = 4;   // Motor B input 1
int in4 = 2;   // Motor B input 2
int enb = 15;  // Motor B enable pin

Servo motorC;  // Create a Servo object for motor C
Servo motorD;  // Create a Servo object for motor D

const int motorCPin = 27;  // Define the pin number for motor C
const int motorDPin = 14;  // Define the pin number for motor D

bool completed = false;

// Function Declarations--------------------------------------
void initializeComponents();
void runGyroscope();
void runUltrasonicOutputCheck();
void runMotorPart();
void shortRepeatedBeeps();
void playReversingSound();
void longBeepWithShortPauses();
void tone(int pin, int frequency);
void noTone(int pin);
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status);
void OnDataRecv(const esp_now_recv_info *recv_info, const uint8_t *incomingData, int len);
void sendData();
void warning(const char *message);

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

  motorC.write(180);
  delay(1000);

  // MOTOR_B_COUNTERCLOCKWISE_01
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
  analogWrite(enb, 178);
  delay(240);

  // MOTOR_B_COUNTERCLOCKWISE_01_STOP
  analogWrite(enb, 0);
  delay(1000);
  // MOTOR_A_COUNTERCLOCKWISE_01
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  analogWrite(ena, 245);
  delay(1500);

  // MOTOR_A_COUNTERCLOCKWISE_01_STOP
  analogWrite(ena, 0);
  delay(1500);
}

void loop() {
  bool gyroscopeFinishedPrinted = false;
  bool ultrasonicCheckCalled = false;
  String warningMessage = myData.warningMessage;

  if (myData.gyroscopeEnterPressed) {
    if (!gyroscopeFinishedPrinted) {
      Serial.println("Gyroscope Finished");
      gyroscopeFinishedPrinted = true;
    }
  } else {
    runGyroscope();
    gyroscopeFinishedPrinted = false;  // Reset the flag if gyroscope is not finished
  }

  if (myData.outputCheckCalled && !myData.motorPartCompleted) {
    // run ultrasonic sensor to check output beaker
    Serial.println("Final output was displayed and output beaker check
called");
    runUltrasonicOutputCheck();
  }

  if (waterFilled && !myData.motorPartCompleted && !completed) {
    // run motor part
    Serial.println("TOF detected threshold and motor part called");
    runMotorPart();
  }
  if (myData.warningCalled) {
    Serial.println(warningMessage);
    warning(warningMessage.c_str());
  }
}

void initializeComponents() {
  // ESP32 Initialization
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_send_cb(OnDataSent);
  esp_now_register_recv_cb(OnDataRecv);

  esp_now_peer_info_t peerInfo;
  memset(&peerInfo, 0, sizeof(peerInfo));
  memcpy(peerInfo.peer_addr, espBoardA_address, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  // Gyroscope Initialization
  Wire.begin();
  mpu.initialize();
```



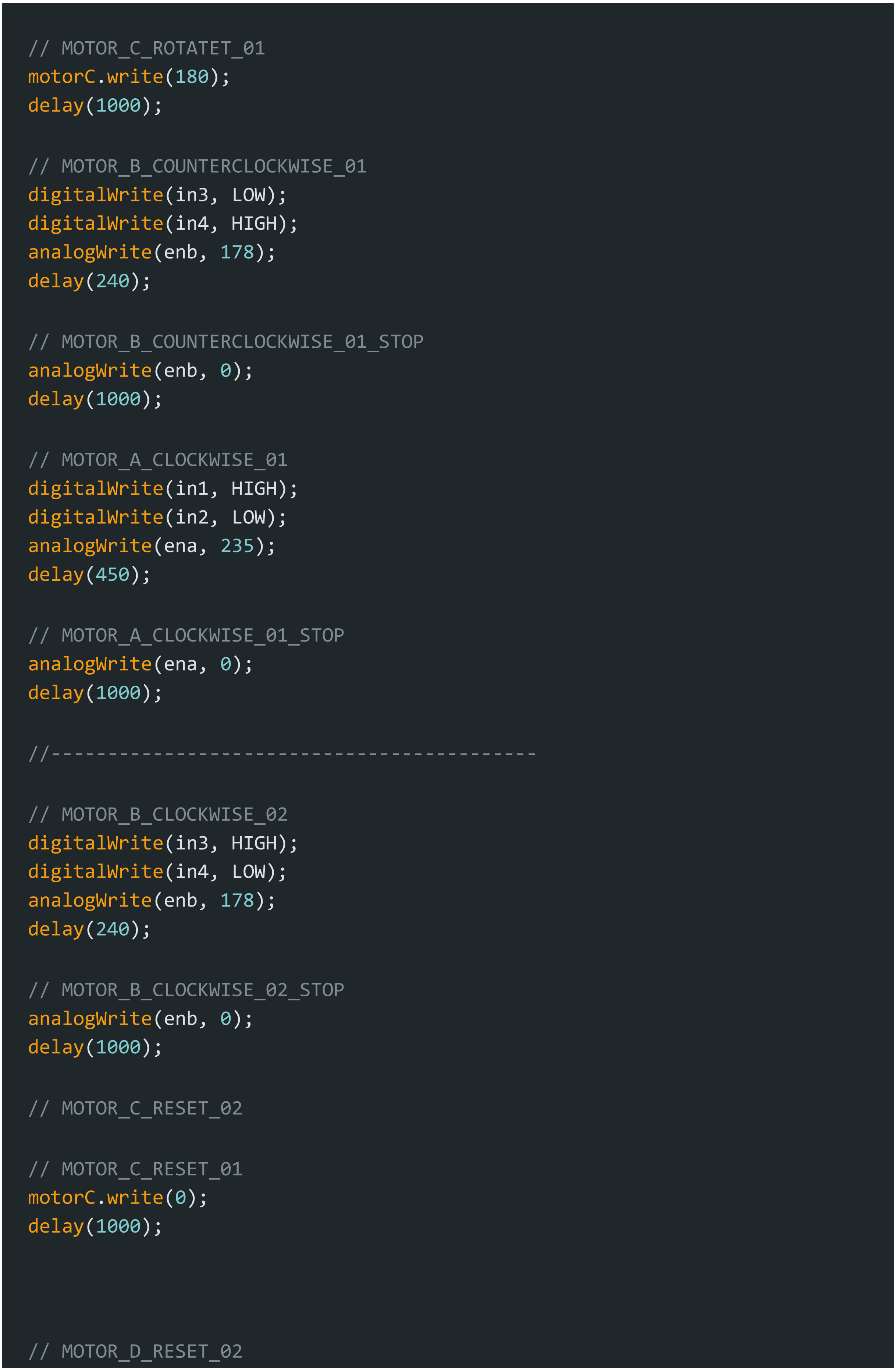
```
  if (!mpu.testConnection()) {
    Serial.println("MPU6050 connection failed");
    while (1)
      ;
  }

  // Buzzer Initialization
  pinMode(buzzerPin, OUTPUT);
  digitalWrite(buzzerPin, LOW);

  // LED pins Initialization
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(GREEN_LED_PIN, OUTPUT);
  pinMode(YELLOW_LED_PIN, OUTPUT);
  digitalWrite(RED_LED_PIN, LOW);
  digitalWrite(GREEN_LED_PIN, LOW);
  digitalWrite(YELLOW_LED_PIN, LOW);

  // Ultrasonic Sensor Initialization
  pinMode(trigPinUS, OUTPUT);
  pinMode(echoPinUS, INPUT);

  // Motor pins initialization
  // Set motor control pins as outputs
  pinMode(ena, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(enb, OUTPUT);

  // Attach servo motors to the defined pins
  motorC.attach(motorCPin);
  motorD.attach(motorDPin);

  // Initial motor states
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  analogWrite(ena, 0);
  analogWrite(enb, 0);

  myData.motorPartCompleted = false;
}
```

```
// Gyroscope Function---------------------------------------------------
void runGyroscope() {
  Serial.println("Gyroscope Running");
  int16_t ax, ay, az;
  int16_t gx, gy, gz;
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  float accelX = ax / 16384.0;
  float accelY = ay / 16384.0;
  float accelZ = az / 16384.0;

  int angleX = atan2(accelY, sqrt(accelX * accelX + accelZ * accelZ)) * 180.0
/ PI;
  int angleY = atan2(accelX, sqrt(accelY * accelY + accelZ * accelZ)) * 180.0
/ PI;
  int angleZ = atan2(sqrt(accelX * accelX + accelY * accelY), accelZ) * 180.0
/ PI;

  Serial.print("X: ");
  Serial.print(angleX + 1);
  Serial.print(" degrees\t");

  Serial.print("Y: ");
  Serial.print(angleY - 1);
  Serial.print(" degrees\t");

  Serial.print("Z: ");
  Serial.print(angleZ - 2);
  Serial.println(" degrees");

  myData.angleX = angleX + 1;
  myData.angleY = angleY - 1;
  myData.angleZ = angleZ - 2;
  myData.gyroscopeDataSent = true;

  static unsigned long lastSendTime = 0;
  unsigned long currentTime = millis();
  if (currentTime - lastSendTime > 1000) {
    sendData();
    lastSendTime = currentTime;
  }

  // Reset the gyroscope data flag after sending
  myData.gyroscopeDataSent = false;

  delay(1000);
```

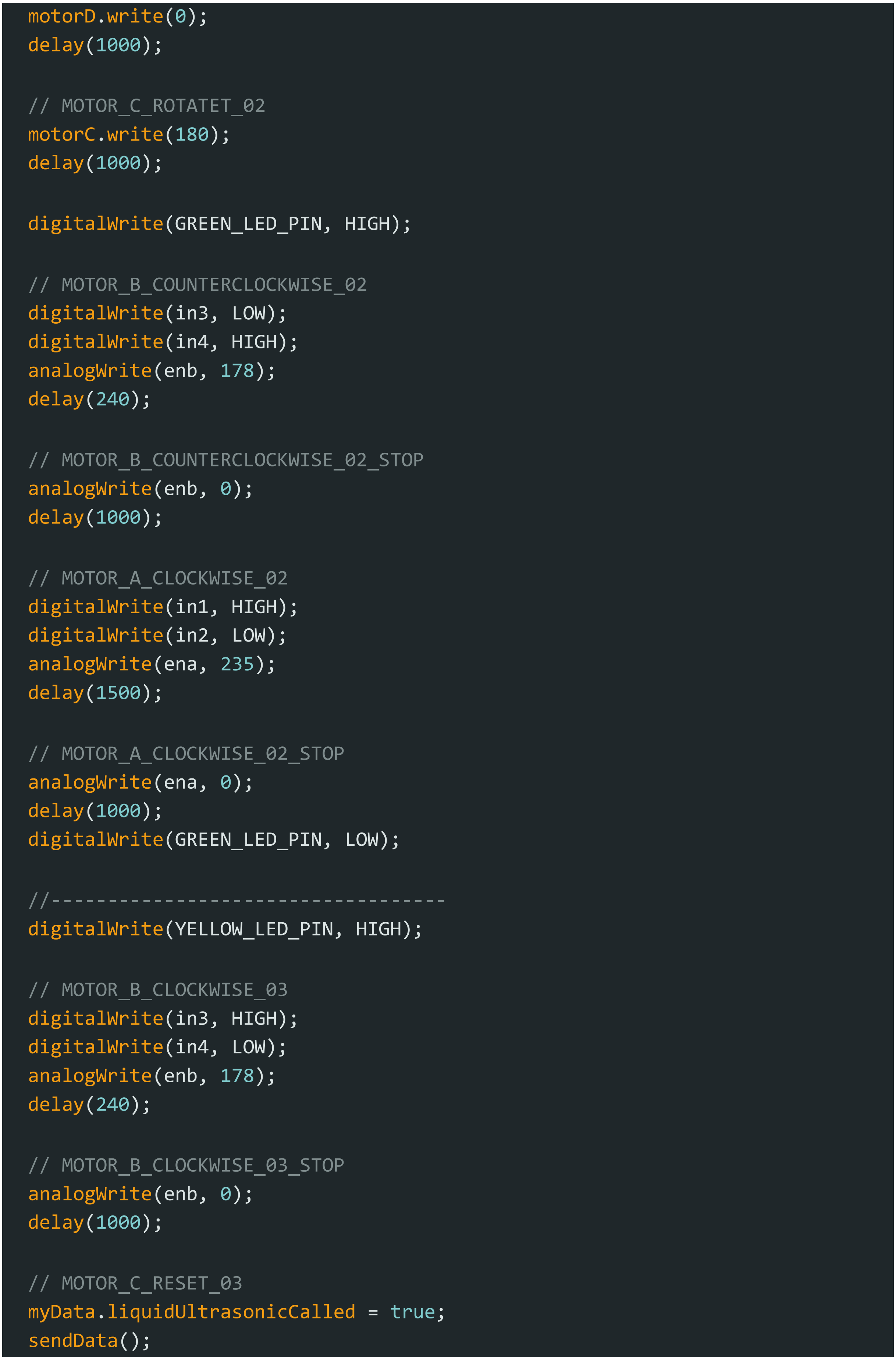


```
}

// Output Beaker Check Ultrasonic Function-----------------------------
void runUltrasonicOutputCheck() {
  digitalWrite(trigPinUS, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinUS, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinUS, LOW);

  duration = pulseIn(echoPinUS, HIGH);
  distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  if (distance <= outputBeakerThreshold) {
    myData.ultrasonicObjectDetected = true;
    Serial.println("Output Beaker Detected");
  } else {
    digitalWrite(RED_LED_PIN, HIGH);
    warning("No Output Beaker");
    delay(5000);
    digitalWrite(RED_LED_PIN, HIGH);
    myData.resetCalled = true;
    delay(5000);
    resetCircuit();
  }

  static unsigned long lastSendTime = 0;
  unsigned long currentTime = millis();
  if (currentTime - lastSendTime > 1000) {
    sendData();
    lastSendTime = currentTime;
  }

  delay(1000);
}

// Motor code----------------------------------------------------------
void runMotorPart() {
  motorC.write(180);
  delay(1000);
```

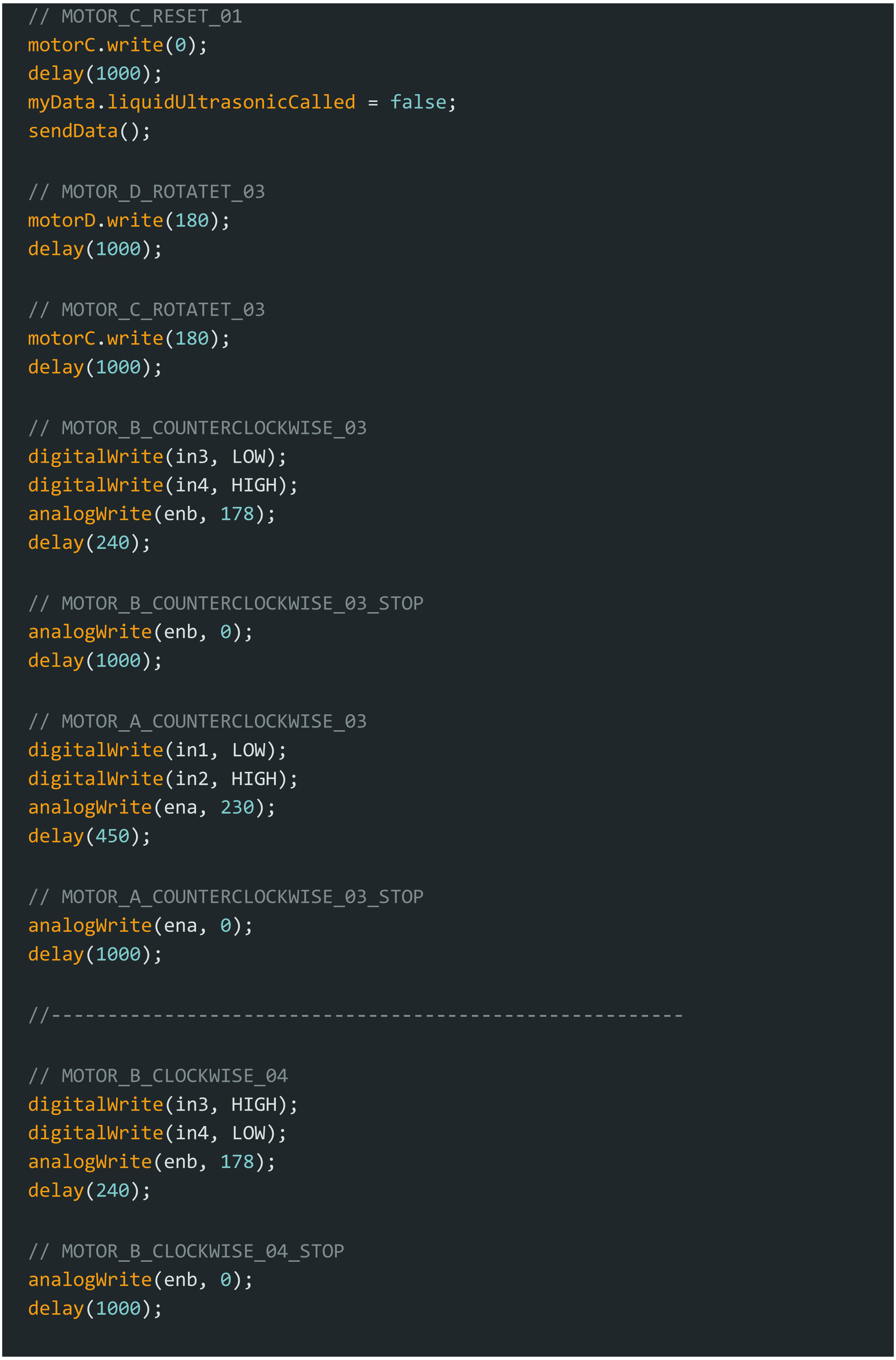
```
// MOTOR_B_COUNTERCLOCKWISE_01
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_COUNTERCLOCKWISE_01_STOP
analogWrite(enb, 0);
delay(1000);
// MOTOR_A_COUNTERCLOCKWISE_01
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
analogWrite(ena, 245);
delay(1500);

// MOTOR_A_COUNTERCLOCKWISE_01_STOP
analogWrite(ena, 0);
delay(1500);
//----------------------------------------

// MOTOR_B_CLOCKWISE_01
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_CLOCKWISE_01_STOP
analogWrite(enb, 0);
delay(1000);


// MOTOR_C_RESET_01
myData.liquidUltrasonicCalled = true;
sendData();
// MOTOR_C_RESET_01
motorC.write(0);
delay(1000);
myData.liquidUltrasonicCalled = false;
sendData();

 motorD.write(0);
delay(1000);

// MOTOR_D_ROTATET_01
motorD.write(180);
delay(1000);
```



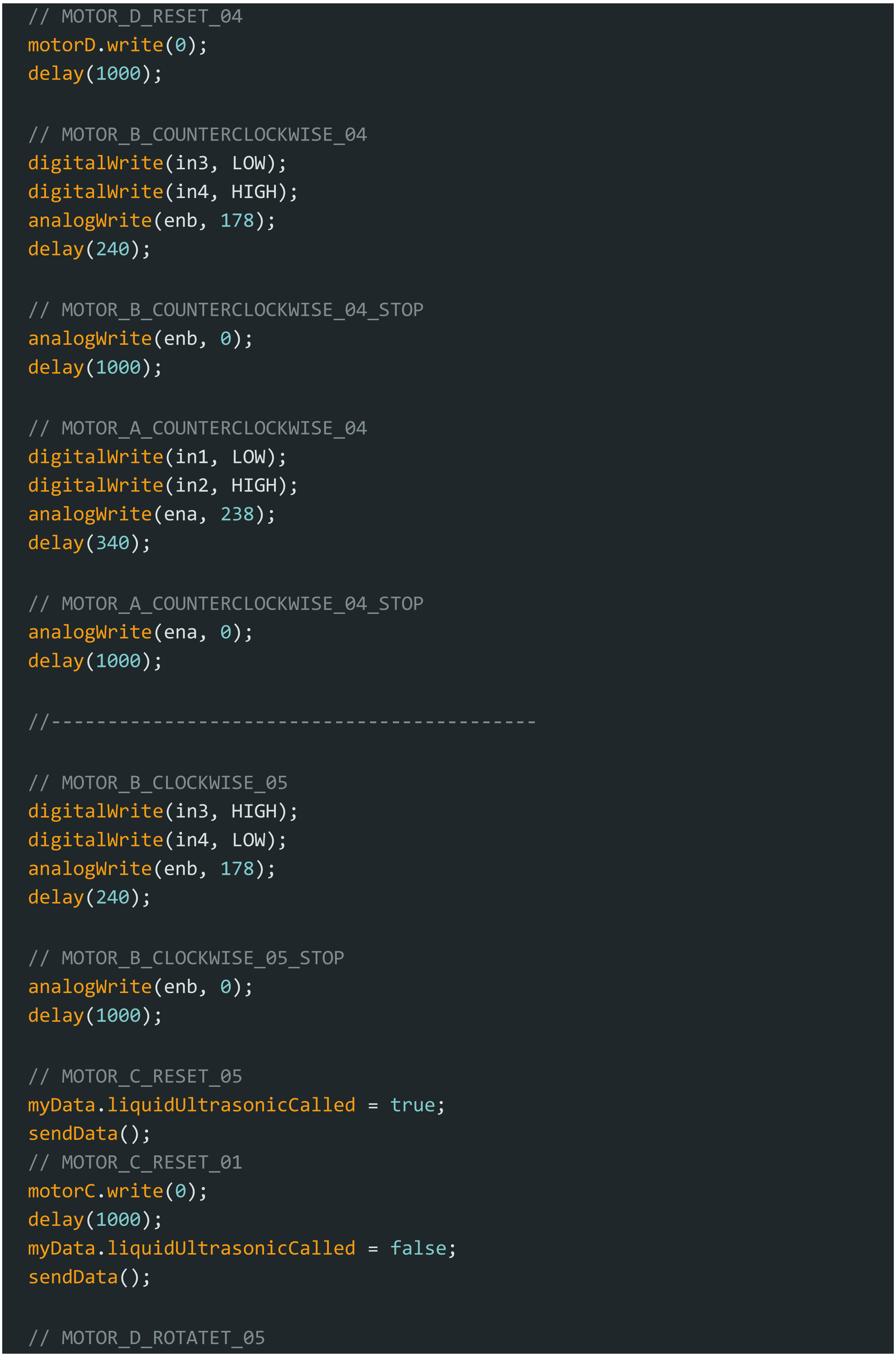
```
// MOTOR_C_ROTATET_01
motorC.write(180);
delay(1000);

// MOTOR_B_COUNTERCLOCKWISE_01
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_COUNTERCLOCKWISE_01_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_A_CLOCKWISE_01
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
analogWrite(ena, 235);
delay(450);

// MOTOR_A_CLOCKWISE_01_STOP
analogWrite(ena, 0);
delay(1000);

//-------------------------------------------

// MOTOR_B_CLOCKWISE_02
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_CLOCKWISE_02_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_C_RESET_02

// MOTOR_C_RESET_01
motorC.write(0);
delay(1000);



// MOTOR_D_RESET_02
```



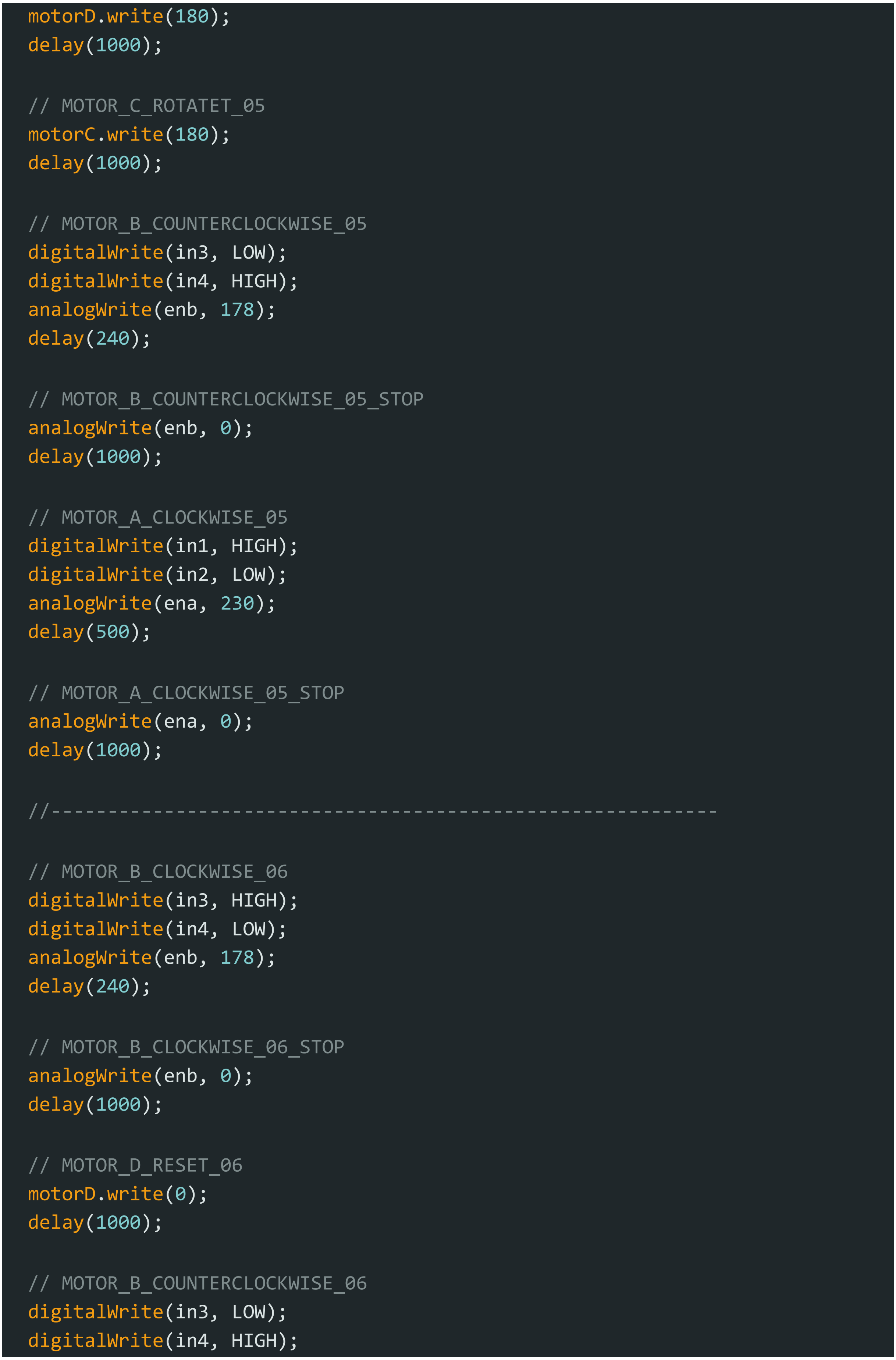
```
motorD.write(0);
delay(1000);

// MOTOR_C_ROTATET_02
motorC.write(180);
delay(1000);

digitalWrite(GREEN_LED_PIN, HIGH);

// MOTOR_B_COUNTERCLOCKWISE_02
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_COUNTERCLOCKWISE_02_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_A_CLOCKWISE_02
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
analogWrite(ena, 235);
delay(1500);

// MOTOR_A_CLOCKWISE_02_STOP
analogWrite(ena, 0);
delay(1000);
digitalWrite(GREEN_LED_PIN, LOW);

//-----------------------------------
digitalWrite(YELLOW_LED_PIN, HIGH);

// MOTOR_B_CLOCKWISE_03
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_CLOCKWISE_03_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_C_RESET_03
myData.liquidUltrasonicCalled = true;
sendData();
```



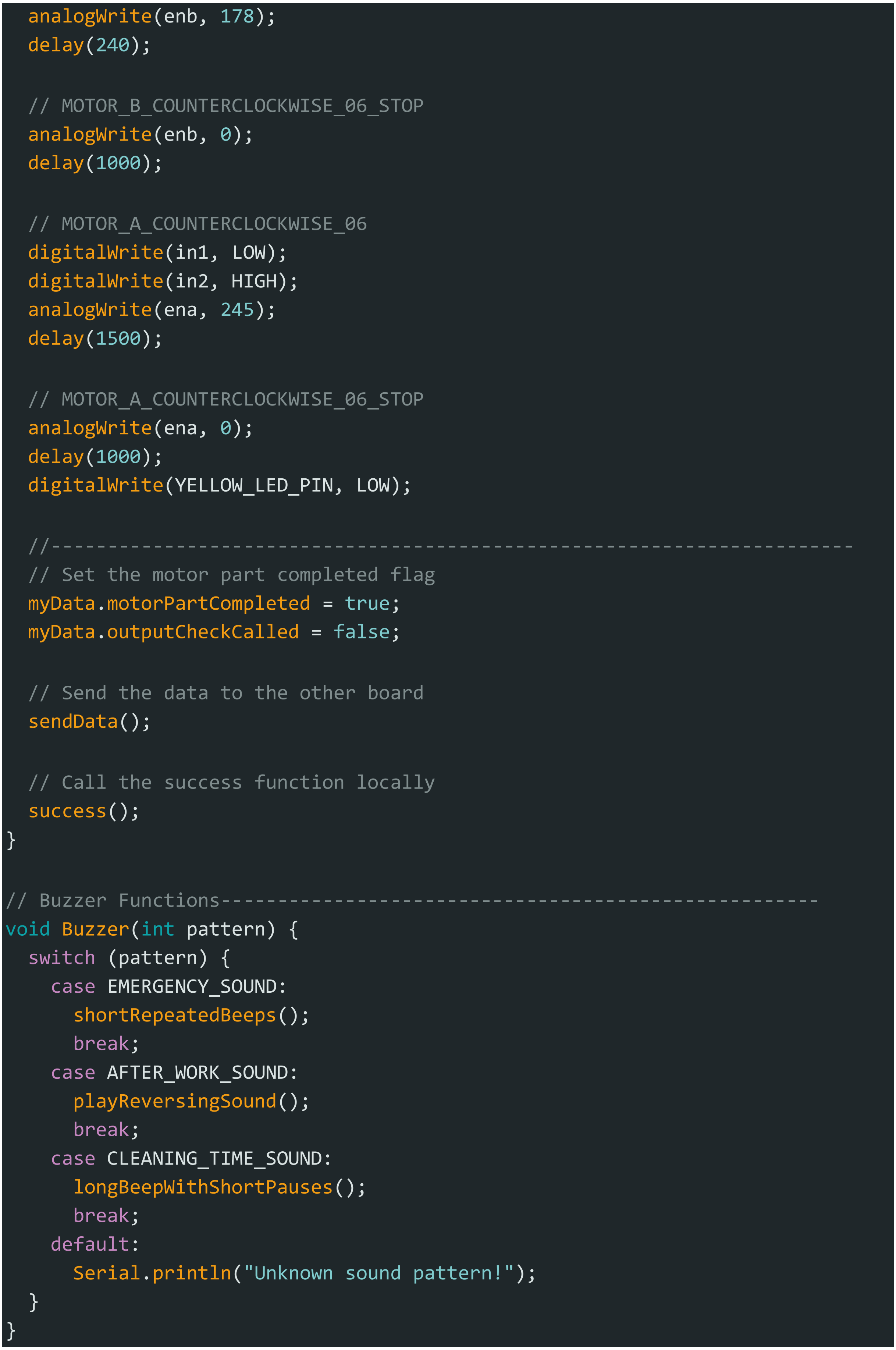
```
// MOTOR_C_RESET_01
motorC.write(0);
delay(1000);
myData.liquidUltrasonicCalled = false;
sendData();

// MOTOR_D_ROTATET_03
motorD.write(180);
delay(1000);

// MOTOR_C_ROTATET_03
motorC.write(180);
delay(1000);

// MOTOR_B_COUNTERCLOCKWISE_03
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_COUNTERCLOCKWISE_03_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_A_COUNTERCLOCKWISE_03
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
analogWrite(ena, 230);
delay(450);

// MOTOR_A_COUNTERCLOCKWISE_03_STOP
analogWrite(ena, 0);
delay(1000);

//-------------------------------------------------------

// MOTOR_B_CLOCKWISE_04
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_CLOCKWISE_04_STOP
analogWrite(enb, 0);
delay(1000);
```



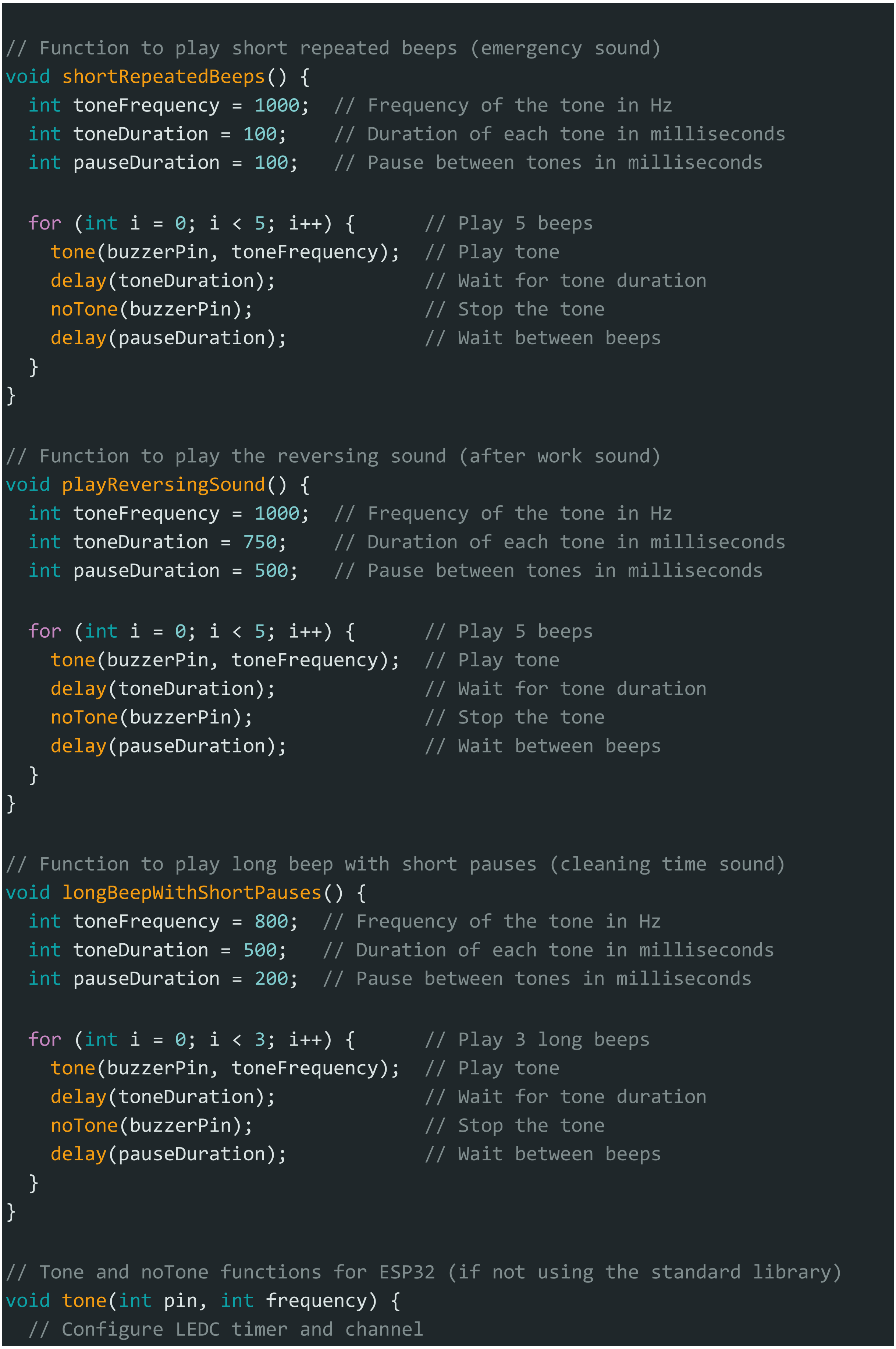
```
// MOTOR_D_RESET_04
motorD.write(0);
delay(1000);

// MOTOR_B_COUNTERCLOCKWISE_04
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_COUNTERCLOCKWISE_04_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_A_COUNTERCLOCKWISE_04
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
analogWrite(ena, 238);
delay(340);

// MOTOR_A_COUNTERCLOCKWISE_04_STOP
analogWrite(ena, 0);
delay(1000);

//-------------------------------------------

// MOTOR_B_CLOCKWISE_05
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_CLOCKWISE_05_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_C_RESET_05
myData.liquidUltrasonicCalled = true;
sendData();
// MOTOR_C_RESET_01
motorC.write(0);
delay(1000);
myData.liquidUltrasonicCalled = false;
sendData();

// MOTOR_D_ROTATET_05
```



```
motorD.write(180);
delay(1000);

// MOTOR_C_ROTATET_05
motorC.write(180);
delay(1000);

// MOTOR_B_COUNTERCLOCKWISE_05
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_COUNTERCLOCKWISE_05_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_A_CLOCKWISE_05
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
analogWrite(ena, 230);
delay(500);

// MOTOR_A_CLOCKWISE_05_STOP
analogWrite(ena, 0);
delay(1000);

//-----------------------------------------------------------

// MOTOR_B_CLOCKWISE_06
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
analogWrite(enb, 178);
delay(240);

// MOTOR_B_CLOCKWISE_06_STOP
analogWrite(enb, 0);
delay(1000);

// MOTOR_D_RESET_06
motorD.write(0);
delay(1000);

// MOTOR_B_COUNTERCLOCKWISE_06
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
```



```
  analogWrite(enb, 178);
  delay(240);

  // MOTOR_B_COUNTERCLOCKWISE_06_STOP
  analogWrite(enb, 0);
  delay(1000);

  // MOTOR_A_COUNTERCLOCKWISE_06
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  analogWrite(ena, 245);
  delay(1500);

  // MOTOR_A_COUNTERCLOCKWISE_06_STOP
  analogWrite(ena, 0);
  delay(1000);
  digitalWrite(YELLOW_LED_PIN, LOW);

  //-----------------------------------------------------------------------
  // Set the motor part completed flag
  myData.motorPartCompleted = true;
  myData.outputCheckCalled = false;

  // Send the data to the other board
  sendData();

  // Call the success function locally
  success();
}

// Buzzer Functions-----------------------------------------------------
void Buzzer(int pattern) {
  switch (pattern) {
    case EMERGENCY_SOUND:
      shortRepeatedBeeps();
      break;
    case AFTER_WORK_SOUND:
      playReversingSound();
      break;
    case CLEANING_TIME_SOUND:
      longBeepWithShortPauses();
      break;
    default:
      Serial.println("Unknown sound pattern!");
  }
}
```



```
// Function to play short repeated beeps (emergency sound)
void shortRepeatedBeeps() {
  int toneFrequency = 1000;  // Frequency of the tone in Hz
  int toneDuration = 100;    // Duration of each tone in milliseconds
  int pauseDuration = 100;   // Pause between tones in milliseconds

  for (int i = 0; i < 5; i++) {      // Play 5 beeps
    tone(buzzerPin, toneFrequency);  // Play tone
    delay(toneDuration);             // Wait for tone duration
    noTone(buzzerPin);               // Stop the tone
    delay(pauseDuration);            // Wait between beeps
  }
}

// Function to play the reversing sound (after work sound)
void playReversingSound() {
  int toneFrequency = 1000;  // Frequency of the tone in Hz
  int toneDuration = 750;    // Duration of each tone in milliseconds
  int pauseDuration = 500;   // Pause between tones in milliseconds

  for (int i = 0; i < 5; i++) {      // Play 5 beeps
    tone(buzzerPin, toneFrequency);  // Play tone
    delay(toneDuration);             // Wait for tone duration
    noTone(buzzerPin);               // Stop the tone
    delay(pauseDuration);            // Wait between beeps
  }
}

// Function to play long beep with short pauses (cleaning time sound)
void longBeepWithShortPauses() {
  int toneFrequency = 800;  // Frequency of the tone in Hz
  int toneDuration = 500;   // Duration of each tone in milliseconds
  int pauseDuration = 200;  // Pause between tones in milliseconds

  for (int i = 0; i < 3; i++) {      // Play 3 long beeps
    tone(buzzerPin, toneFrequency);  // Play tone
    delay(toneDuration);             // Wait for tone duration
    noTone(buzzerPin);               // Stop the tone
    delay(pauseDuration);            // Wait between beeps
  }
}

// Tone and noTone functions for ESP32 (if not using the standard library)
void tone(int pin, int frequency) {
  // Configure LEDC timer and channel
  ledc_timer_config_t ledc_timer = {
    .speed_mode = LEDC_HIGH_SPEED_MODE,
    .duty_resolution = LEDC_TIMER_8_BIT,
    .timer_num = LEDC_TIMER_0,
    .freq_hz = (uint32_t)frequency,
    .clk_cfg = LEDC_AUTO_CLK
  };
  ledc_timer_config(&ledc_timer);

  ledc_channel_config_t ledc_channel = {
    .gpio_num = pin,
    .speed_mode = LEDC_HIGH_SPEED_MODE,
    .channel = LEDC_CHANNEL_0,
    .intr_type = LEDC_INTR_DISABLE,
    .timer_sel = LEDC_TIMER_0,
    .duty = 127,  // 50% duty cycle
    .hpoint = 0
  };
  ledc_channel_config(&ledc_channel);
  ledc_timer_pause(LEDC_HIGH_SPEED_MODE, LEDC_TIMER_0);
  ledc_timer_resume(LEDC_HIGH_SPEED_MODE, LEDC_TIMER_0);
}

void noTone(int pin) {
  ledc_stop(LEDC_HIGH_SPEED_MODE, LEDC_CHANNEL_0, 0);  // Stop the PWM signal
}

// Data Communication--------------------------------------------------
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\nLast Packet Send Status: ");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

void OnDataRecv(const esp_now_recv_info *recv_info, const uint8_t
*incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Bytes received: ");
  Serial.println(len);

  if (myData.warningCalled) {
    warning(myData.warningMessage.c_str());  // Call warning function if
warning flag is set
  }
  if (myData.waterFilled) {
    // run motor part
    Serial.println("TOF detected threshold and motor part called");
    waterFilled = true;
  }
  if (myData.resetCalled) {
    Serial.println("RESET called");
    resetCircuit();
  }
  if (myData.callRED) {
    digitalWrite(RED_LED_PIN, HIGH);
    delay(5000);
    digitalWrite(RED_LED_PIN, LOW);
    myData.callRED = false;
  }
}

void sendData() {
  esp_err_t result = esp_now_send(espBoardA_address, (uint8_t *)&myData,
sizeof(myData));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
  } else {
    Serial.println("Error sending the data");
  }

  // Reset the gyroscope data flag after sending
  myData.gyroscopeDataSent = false;
  // Reset the motor part completed flag after sending
  myData.motorPartCompleted = false;
  // Reset the warning flag after sending
  myData.warningCalled = false;
}

// Reset Code---------------------------------------------------------
void resetCircuit() {
  Serial.println("ESP32 will reset in 1 seconds...");
  delay(1000);
  esp_restart();
}

// Warning Function---------------------------------------------------
void warning(const char *message) {
  // Set the warning flag and message
  myData.warningCalled = true;
  myData.warningMessage = message;
  // Send the data to the other board
  sendData();

  // Activate buzzer and LED
  Buzzer(EMERGENCY_SOUND);
  digitalWrite(RED_LED_PIN, HIGH);

  // Display warning message on Serial Monitor
  Serial.print("Warning: ");
  Serial.println(message);

  // Send warning message to the other ESP32 board
  struct_message warningData = myData;
  myData.warningMessage = message;
  esp_now_send(espBoardA_address, (uint8_t *)&warningData,
sizeof(warningData));

  // Keep the warning state for 5 seconds
  delay(5000);

  // Deactivate LED
  digitalWrite(buzzerPin, LOW);

  // turn off the warning flag
  myData.warningCalled = false;
}

// Success Function-----------------------------------------------------
void success() {
  // Activate buzzer and LED
  completed = true;
  Buzzer(AFTER_WORK_SOUND);
  digitalWrite(GREEN_LED_PIN, HIGH);

  // Keep the success state for 5 seconds
  delay(5000);

  // Deactivate buzzer and LED
  digitalWrite(GREEN_LED_PIN, LOW);
}
```