ADITYA VISHE BT22CIV091

EXPERIMENT NO. 06

Aim:

To design and implement a proximity-based smartwatch system using ESP32, BLE, and PIR sensors to control various smart home devices (lights, AC, coffee maker) based on user motion and time of day.

Time	Light intensity	AC temp.	Coffee maker
7:00 am to 11:00 am	50%	25	ON
11:00 am to 4:00 pm	75%	20	OFF
4:00 pm to 7:00 pm	100%	25	ON
7:00 pm to 10:00 pm	100%	20	OFF
10:00 pm to 7:00 am	25%	15	OFF

Apparatus:

1. **Hardware:**
   - ESP32 (to act as the main control server)
   - PIR (Passive Infrared) Sensor (to detect motion)
   - Smartwatch (client device with BLE capabilities)
   - Smart light system with dimming feature
   - Smart AC with temperature control
   - Coffee maker with smart switch
   - BLE module (if needed for interfacing)
2. **Software:**
   - Arduino IDE (for ESP32 and PIR sensor programming)

- BLE library for ESP32
- Web server or notification service (optional, for sending text alerts)

Theory:

This proximity-based smartwatch system for smart home automation uses Bluetooth Low Energy (BLE) communication, PIR sensors, and time-based device control to enhance user convenience and energy efficiency.

1. BLE Communication:
Bluetooth Low Energy (BLE) enables efficient, low-power communication between the smartwatch (client) and the ESP32 (server). The smartwatch sends signals to ESP32 when it detects motion, prompting the server to control connected devices like lights, AC, and coffee makers.

2. PIR Sensor for Motion Detection:
A PIR sensor on the smartwatch detects motion to activate the display and communicate with the ESP32 server, which only turns devices on when the user is present. This setup conserves power by limiting device use to necessary moments.

3. Time and Motion-Based Device Control:
The system follows a preset schedule to control devices based on the time of day. For example, in the morning, the system sets a 50% light intensity, a 25°C AC temperature, and turns on the coffee maker. This time-specific automation optimizes energy use while adapting to user needs throughout the day.

4. Energy Efficiency and Notifications:
To save battery, the smartwatch display turns off after 15 seconds of inactivity. The system also sends notifications to the smartwatch, like when coffee is ready, to keep the user informed without manual checks.

In summary, this system leverages BLE, PIR sensors, and time-based controls to create a responsive, energy-efficient smart home setup, making daily routines more comfortable and convenient.

Procedure:

1. **Setup and Initialization:**
   - **ESP32 Setup**: Configure ESP32 to operate as a BLE server, with the smartwatch as the BLE client.
   - **Smartwatch Configuration**: Set up smartwatch to detect motion using a PIR sensor. Program it to turn on the display when motion is detected and turn off after 15 seconds of inactivity.

- **Device Control Setup**: Program ESP32 to control smart home devices based on time and motion status.

2. **Programming:**

- **Server Code :**

```
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

#define LIGHT_PIN 15   // Light control pin
#define AC_PIN 14      // AC control pin
#define COFFEE_PIN 13  // Coffee maker control pin

// BLE Service and Characteristic UUIDs
const char* SERVICE_UUID = "12345678-1234-5678-1234-56789abcdef0";
const char* CHAR_UUID = "abcd1234-5678-1234-5678-abcdef123456";

BLEServer* pServer;
BLEService* pService;
BLECharacteristic* pChar;

void setup() {
  pinMode(LIGHT_PIN, OUTPUT);
  pinMode(AC_PIN, OUTPUT);
  pinMode(COFFEE_PIN, OUTPUT);

  Serial.begin(9600);

  // Initialize BLE
  BLEDevice::init("SmartHome");
  pServer = BLEDevice::createServer();
  pService = pServer->createService(SERVICE_UUID);

  pChar = pService->createCharacteristic(
    CHAR_UUID,
    BLECharacteristic::PROPERTY_WRITE
  );

  pChar->setCallbacks(new MyCallbacks());
  pService->start();

  BLEAdvertising* pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->start();
  Serial.println("Smart Home System Initialized.");
}

class MyCallbacks : public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic* pCharacteristic) {
    std::string value = pCharacteristic->getValue();
    if (value == "motion") {
      handleMotion();
    }
  }
};

void handleMotion() {
  String time = getCurrentTime();  // Get the current time
  controlDevices(time);  // Control devices based on time

  if (digitalRead(COFFEE_PIN) == HIGH) {
    Serial.println("Coffee ready! Sending notification...");
    // Simulate sending a notification back to the client
    Serial.println("Coffee is ready!");
  }
}

void controlDevices(String time) {
  if (time >= "07:00" && time < "11:00") {
    setLight(50);
    setAC(25);
    digitalWrite(COFFEE_PIN, HIGH);  // Turn ON coffee maker
  } else if (time >= "11:00" && time < "16:00") {
    setLight(75);
    setAC(20);
    digitalWrite(COFFEE_PIN, HIGH);  // Turn ON coffee maker
  } else if (time >= "16:00" && time < "19:00") {
    setLight(100);
    setAC(25);
    digitalWrite(COFFEE_PIN, LOW);  // Turn OFF coffee maker
  } else if (time >= "19:00" && time < "22:00") {
    setLight(30);
    setAC(20);
    digitalWrite(COFFEE_PIN, LOW);  // Turn OFF coffee maker
  } else {
    setLight(25);
    setAC(15);
  digitalWrite(COFFEE_PIN, LOW);  // Turn OFF coffee maker
 }
}

void setLight(int intensity) {
 analogWrite(LIGHT_PIN, map(intensity, 0, 100, 0, 255));
 Serial.print("Light intensity set to: ");
 Serial.println(intensity);
}

void setAC(int temp) {
 digitalWrite(AC_PIN, HIGH);
 Serial.print("AC temperature set to: ");
 Serial.print(temp);
 Serial.println(" °C");
}

String getCurrentTime() {
 // Placeholder for real-time clock or NTP logic
 return "08:00";  // Example: 8:00 AM
}
```

- **Client Code :**

```
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEClient.h>

#define PIR_PIN 4  // PIR sensor connected to GPIO 4
unsigned long lastMotionTime = 0;
const unsigned long DISPLAY_TIMEOUT = 15000;  // 15 seconds timeout

// BLE Server Address and UUIDs
static BLEUUID serviceUUID("12345678-1234-5678-1234-56789abcdef0");
static BLEUUID charUUID("abcd1234-5678-1234-5678-abcdef123456");
BLEAddress serverAddress("XX:XX:XX:XX:XX:XX");  // Replace with actual MAC
address

BLEClient* client;

void setup() {
 pinMode(PIR_PIN, INPUT);
 Serial.begin(9600);
```

```
  // Initialize BLE
  BLEDevice::init("Smartwatch");
  client = BLEDevice::createClient();
}

void loop() {
  // Check for motion detection
  if (digitalRead(PIR_PIN) == HIGH) {
    lastMotionTime = millis();  // Reset timeout timer
    sendMotionDetected();       // Send BLE message to the server
    Serial.println("Motion detected. Display ON.");
  }

  // Turn off display after 15 seconds of inactivity
  if (millis() - lastMotionTime > DISPLAY_TIMEOUT) {
    Serial.println("Turning off display due to inactivity.");
  }
}

void sendMotionDetected() {
  if (client->connect(serverAddress)) {
    Serial.println("Connected to server. Sending motion signal...");

    BLERemoteService* pService = client->getService(serviceUUID);
    if (pService) {
      BLERemoteCharacteristic* pChar = pService->getCharacteristic(charUUID);
      if (pChar) {
        pChar->writeValue("motion");  // Send 'motion' signal to the server
      }
    }
    client->disconnect();  // Disconnect after sending data
  } else {
    Serial.println("Failed to connect to server.");
  }
}
```

3. **Testing and Validation**:
   - Connect the smartwatch with ESP32 and test motion detection with the PIR sensor.
   - Observe if the display turns on and off based on motion.
   - Check device control (lights, AC, coffee maker) as per Table 1 and verify response times and notifications.

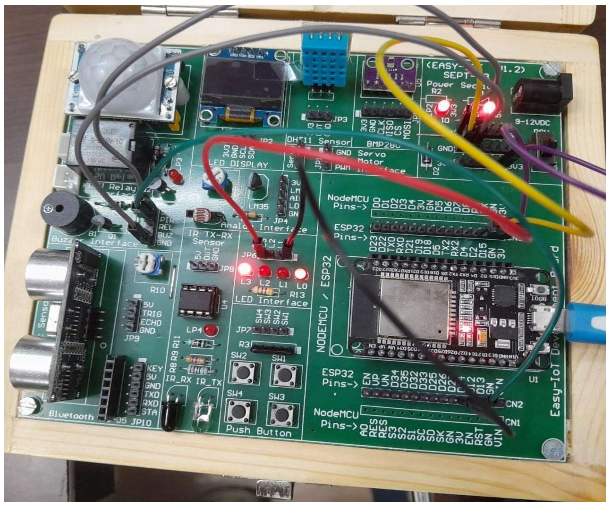
Observations:

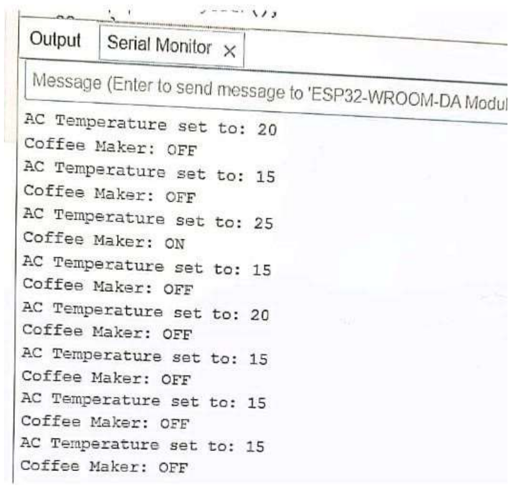
- Record how long it takes for each device to activate after motion detection.
- Note any delays in BLE communication between the smartwatch and ESP32.
- Verify if notifications for the coffee maker are sent correctly when the coffee maker is turned on.

Conclusion:

The designed proximity-based smartwatch system using ESP32 and BLE successfully controls smart home devices based on motion detection and time-specific settings. This project demonstrates effective use of BLE communication and energy-efficient display management, proving that the system is viable for smart home automation.

Results :
connections and Serial monitor readings:





```
AC Temperature set to: 20
Coffee Maker: OFF
AC Temperature set to: 15
Coffee Maker: OFF
AC Temperature set to: 25
Coffee Maker: ON
AC Temperature set to: 15
Coffee Maker: OFF
AC Temperature set to: 20
Coffee Maker: OFF
AC Temperature set to: 15
Coffee Maker: OFF
AC Temperature set to: 15
Coffee Maker: OFF
AC Temperature set to: 15
Coffee Maker: OFF
```