

IoT lab2: 搭建Nimble GATT服务器

3220102866 陈奕萱

1 实验目的和要求

- 掌握Nimble GATT 服务器工作原理；
- 掌握如何在RIOS-OS系统使用Nimble蓝牙协议栈；
- 掌握如何搭建可读写的GATT服务器；
- 掌握如何手机或电脑测试GATT服务器；
- 通过蓝牙获取设备的运动状态以及LED状态，且控制设备模型预测阈值、预测频率以及LED状态。

2 实验内容

Nimble GATT服务器搭建实验：

- ESP32设备蓝牙广播
- ESP32设备搭建自定义GATT可读写服务特性
- 通过服务特性，用户获取设备运动信息、LED状态信息
- 通过服务特性，用户控制设备预测阈值、预测频率和LED状态

3 实验背景

- RIOT 操作系统
 - RIOT(<https://github.com/RIOT-OS/RIOT>) 是一个开源的微控制器操作系统，旨在满足物联网(IoT)设备和其他嵌入式设备的需求。它支持一系列通常在物联网(IoT)中发现的设备:8位, 16位和32位微控制器。RIOT基于以下设计原则:节能、实时功能、内存占用小、模块化和统一的API访问，独立于底层硬件(该API提供部分POSIX遵从性)。
- Nimble
 - NIMBLE协议 (NimBLE) 是一个轻量级的蓝牙低功耗 (BLE, Bluetooth Low Energy) 栈，用于实现低功耗设备之间的通信。它的设计目标是提供一个高效、灵活且资源占用少的BLE协议栈，特别适用于资源受限的嵌入式设备。NimBLE协议具有轻量级、跨平台支持、模块化设计、高效性和开源等特点。它支持多种操作系统，包括FreeRTOS、Mynewt、Zephyr等，使得它可以在各种嵌入式平台上运行。其模块化设计使开发者可以根据需要启用或禁用特定功能，优化内存和CPU资源的使用。此外，NimBLE优化了资源使用和功耗，非常适合用于电池供电的IoT设备。作为一个开源项目，NimBLE基于Apache 2.0协议发布，支持最新的BLE5.0标准，包括增强的广播功能、2M PHY、长范围和高数据速率传输。
 - NimBLE在各种应用场景中都有广泛的应用，如可穿戴设备、智能家居和工业物联网。在这些场景中，NimBLE通过其低功耗和高效性，实现了设备之间的可靠通信。总之，NimBLE协议由于其轻量级、高效性和灵活性，成为了许多低功耗蓝牙应用的理想选择。
- NimBLE GATT
 - NimBLE GATT (Generic Attribute Profile) 服务器是NimBLE协议栈中的一个关键组件，用于管理蓝牙设备之间的属性数据交换。它通过定义和维护服务、特性及描述符来组织和提供数据给GATT客户端。GATT服务器可以定义多个服务，每个服务由一组相关的特性组成，这些特性是数据的最小单位，包含属性值及可选的描述符。描述符提供了关于特性值的附加信息，如格式、范围和单位等。GATT服务器支持的主要操作包括读取 (Read) 、写入 (Write) 、通

知 (Notify) 和指示 (Indicate) , 允许客户端与服务器交互, 以获取或设置数据。NimBLE GATT服务器的服务和特性可以在初始化时配置, 并且在设备运行期间保持有效。开发者可以动态地添加或删除服务和特性, 同时利用丰富的事件回调接口来处理客户端连接、特性读写请求等事件。

- 实现NimBLE GATT服务器时, 开发者需要按照以下步骤进行: 初始化NimBLE栈, 设置设备名称和广播参数; 定义服务、特性和描述符, 并将它们注册到GATT数据库中; 设置回调函数以处理读写操作、连接状态变化等事件; 启动BLE广播, 使得GATT客户端可以发现并连接到服务器; 最后, 处理客户端发起的读写请求, 并根据需要发送通知或指示。

4 主要仪器设备

- PC
- ESP32-WROOM-32、MPU6050 惯性传感器、LED RGB 灯。

5 实验题目简答

5.1 想象一下, Nimble GATT服务器在生活中的应用场景有哪些?

智能家居设备互联:

- 智能灯泡:** 智能灯通过BLE与手机应用连接, 用户可以通过应用控制灯光的颜色、亮度等功能, 甚至可以通过语音助手进行控制。
- 智能门锁:** 用户通过手机蓝牙连接智能门锁, 可以远程解锁和锁定门禁, 或查看门锁状态, 方便家庭安全管理。
- 温控系统:** 空调或暖气设备通过BLE连接到手机或其他智能设备, 用户可以在任何房间里通过手机调整温度, 甚至在外出时进行远程控制。

穿戴式设备:

- 智能手表/手环:** 这些设备通过BLE与手机相连, 实现心率监测、步数统计、来电通知等功能。Nimble GATT服务器在这种场景中主要用于传输数据, 比如心率、运动记录等, 并将其发送到手机应用程序。
- 智能眼镜:** 蓝牙通过BLE连接手机后, 可以将通知、导航指令等信息通过眼镜显示给用户。

健康监测设备:

- 血糖监测仪:** 使用Nimble GATT服务器, 血糖监测设备可以通过BLE将血糖数据传输到手机或云端进行分析和存储, 用户和医生可以实时查看数据, 调整治疗方案。
- 血压计、心电监测设备:** 通过BLE, 用户的健康数据可以实时传输到移动设备, 用户可以随时监测自己的健康状况。

健身器材互联:

- 智能跑步机:** 跑步机通过蓝牙与手机连接, 用户可以实时监测运动数据, 并自动同步到健身应用中, 方便制定和跟踪健身计划。
- 健身自行车:** 通过BLE将锻炼数据 (如距离、速度、消耗卡路里等) 传输至手机或健身平台, 并提供个性化健身建议。

智能玩具:

- 蓝牙遥控玩具:** 使用BLE, 用户可以通过手机App与遥控汽车、机器人等玩具互动, 控制玩具的动作、灯光和声音效果, 提升互动体验。

- **学习工具**: 一些儿童智能学习设备可以通过BLE与家长的设备互联，实时查看学习进度和成果，或者家长可以远程控制设备以限制使用时间。

个人财务管理:

- **智能钱包**: 通过BLE与手机连接，智能钱包可以在丢失时向用户手机发送提醒，也能记录使用信息，帮助用户管理消费记录。

安全与监控系统:

- **智能摄像头**: 通过蓝牙低功耗技术，摄像头与用户的手机连接，当检测到异常活动时，摄像头通过BLE通知用户手机，帮助用户实时监控家中安全情况。

5.2 Nimble协议栈的特点是什么？

1. 轻量级设计

- Nimble专为低资源设备设计，占用的内存和处理能力较少，非常适合运行在微控制器（MCU）和嵌入式系统上。例如，它能够在内存有限（如几十KB RAM）的设备上运行。

2. 模块化架构

- Nimble协议栈采用模块化设计，开发者可以根据需要选择和定制不同的功能模块。例如，开发者可以选择仅启用GATT、L2CAP或其他部分功能，以节省系统资源并适配不同的设备需求。
- 这种模块化结构也使得协议栈易于集成、扩展和维护，开发者可以根据具体应用场景添加或删除模块。

3. 支持主机和控制器模式

- Nimble协议栈既可以作为蓝牙主机（Host）来处理高层协议，如GATT和ATT，也可以作为控制器（Controller）处理底层协议，如L2CAP、HCI等。它能够很好地处理不同的蓝牙通信角色，支持BLE的设备连接、数据传输和管理。
- **Controller**: 实现低层部分，如直接访问硬件的接口、物理层和链路层功能。
- **Host**: 实现蓝牙核心协议栈的高层功能，如GATT、SMP、ATT、L2CAP等。

4. 高效低功耗

- Nimble协议栈专为BLE设计，支持BLE 5.0/5.1规范，具备低功耗特性，适合对电池寿命要求较高的设备，如可穿戴设备、传感器等。通过优化的睡眠模式和低功耗数据传输模式，Nimble可以有效减少能耗。

5. 跨平台支持

- Nimble协议栈可以与不同的操作系统（如Apache Mynewt、Zephyr、FreeRTOS等）集成，也可以独立于操作系统运行。它具有良好的可移植性，适合多种嵌入式平台和架构。

6. 支持多种蓝牙角色和模式

- **多角色支持**: Nimble能够同时支持不同的BLE角色，如中央设备（Central）、外围设备（Peripheral）、广播器（Broadcaster）和观察者（Observer），并且可以在这些角色之间无缝切换。
- **多连接支持**: Nimble允许设备同时与多个蓝牙设备建立连接，这对于复杂的物联网环境尤为重要。

7. 灵活的API接口

- Nimble提供简洁且灵活的API接口，便于开发者进行BLE相关的应用开发。它支持C语言编写的API，可以灵活调用和配置蓝牙设备的功能，如启动BLE服务、扫描设备、建立连接、传输数据等。

8. 安全性支持

- Nimble支持蓝牙安全功能，如配对（Pairing）、加密（Encryption）和身份验证（Authentication），包括BLE中的安全管理协议（SMP），为数据传输和设备连接提供必要的安全保障。

9. BLE Mesh支持

- Nimble协议栈支持蓝牙Mesh网络，这是BLE 5.0中的重要特性。它允许多个设备在网状网络中进行通信，特别适用于智能家居、楼宇自动化等场景。

10. 开源和社区驱动

- Nimble协议栈是开源的，拥有活跃的社区支持。它依赖于Apache许可证，允许用户自由使用、修改和分发。开发者可以通过社区贡献代码、提交问题或改进文档，获得持续的技术支持和更新。

5.3 蓝牙协议中UUID使用哪种字节序，为什么设置的UUID和收到的UUID顺序刚好反过来，是否整个蓝牙包字节序都相同，不同协议层的字节序情况如何？

蓝牙协议栈大多数层都采用小端字节序，这样可以与主流的嵌入式设备（例如x86和ARM架构）保持一致。UUID在设置和接收时表现为字节序反转，主要是因为它在传输时采用了小端字节序。

在不同协议层，蓝牙协议保持了一致的字节序，主要为小端格式，保证了数据流在链路层、L2CAP、ATT和GATT层之间的高效传输与兼容性。

5.4 简述客户端蓝牙和服务端蓝牙建立连接的过程？

客户端与服务端蓝牙连接的过程主要包括以下步骤：

1. **广播和扫描**：服务端广播自身信息，客户端扫描广播包。
2. **连接请求**：客户端发现目标设备后发起连接请求，服务端响应并建立物理连接。
3. **连接参数协商**：双方协商调整连接参数以优化性能。
4. **服务发现**：客户端探索服务端的服务和特性。
5. **特性交互**：客户端读取或写入服务端的特性数据。
6. **订阅通知或指示**：客户端可以订阅服务端的通知，进行实时数据更新。
7. **断开连接**：通信结束后，客户端或服务端可以断开连接。

这一过程保证了蓝牙设备之间的高效、低功耗数据交换。

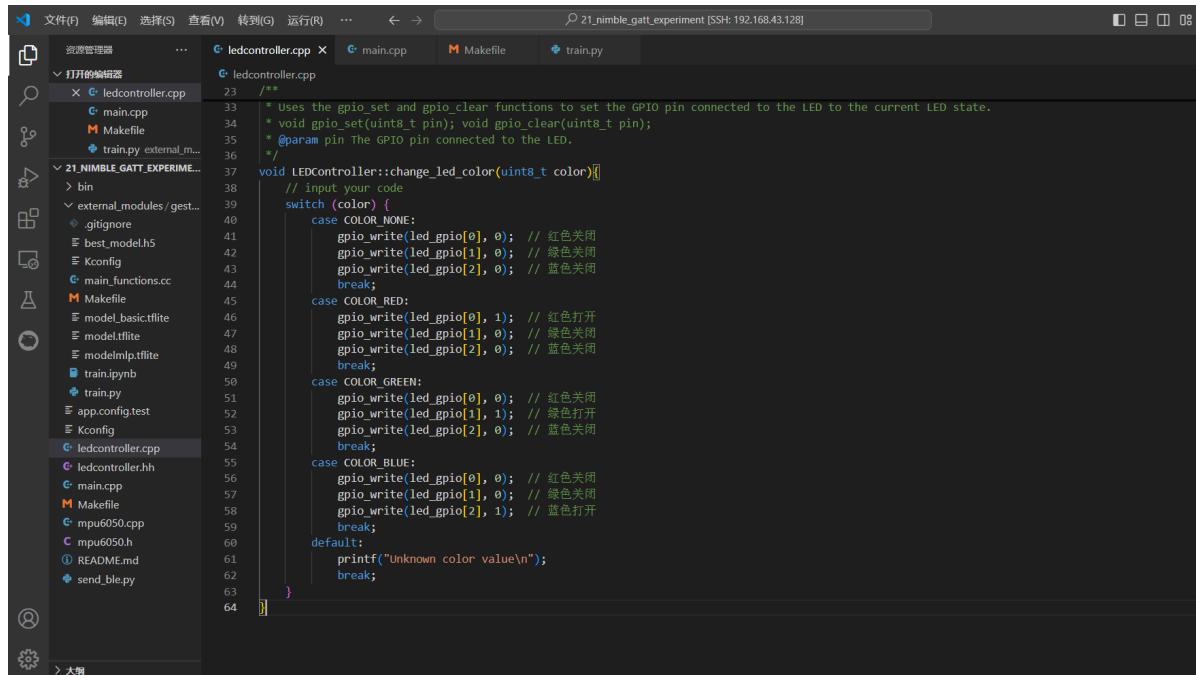
6 实验数据记录和处理

6.1 基础设备控制实验

a. 基础设备控制

1. ledcontroller.cpp

完全来自前面 lab



The screenshot shows a terminal window with the title "21_nimble_gatt_experiment [SSH: 192.168.43.128]". The left pane displays a file tree for the project "21_NIMBLE_GATT_EXPERIMENT". The right pane shows the content of the file "ledcontroller.cpp". The code defines a function "void LEDController::change_led_color(uint8_t color)" which uses GPIO pins to set the color of three LEDs (red, green, blue) based on the input color parameter.

```
23  /* Uses the gpio_set and gpio_clear functions to set the GPIO pin connected to the LED to the current LED state.
24  * @param pin The GPIO pin connected to the LED.
25  */
26
27 void LEDController::change_led_color(uint8_t color){
28     // input your code
29     switch (color) {
30         case COLOR_NONE:
31             gpio.write(led_gpio[0], 0); // 红色关闭
32             gpio.write(led_gpio[1], 0); // 绿色关闭
33             gpio.write(led_gpio[2], 0); // 蓝色关闭
34             break;
35         case COLOR_RED:
36             gpio.write(led_gpio[0], 1); // 红色打开
37             gpio.write(led_gpio[1], 0); // 绿色关闭
38             gpio.write(led_gpio[2], 0); // 蓝色关闭
39             break;
40         case COLOR_GREEN:
41             gpio.write(led_gpio[0], 0); // 红色关闭
42             gpio.write(led_gpio[1], 1); // 绿色打开
43             gpio.write(led_gpio[2], 0); // 蓝色关闭
44             break;
45         case COLOR_BLUE:
46             gpio.write(led_gpio[0], 0); // 红色关闭
47             gpio.write(led_gpio[1], 0); // 绿色关闭
48             gpio.write(led_gpio[2], 1); // 蓝色打开
49             break;
50         default:
51             printf("Unknown color value\n");
52             break;
53     }
54 }
```

2. train.py

修改使用了 mlp 模型，修改了程序中的部分参数

3. main.cpp

1. led 线程：这里设计了 led 灯可以由蓝牙实现状态输入，蓝牙输入 0100 0000, 0200 0000, 0300 0000 分别表示红色、蓝色、绿色的常亮状态，输入 0500 0000 表示退出常亮状态。

```
void *_led_thread(void *arg)
{
    (void) arg;
    LEDController led(LED_GPIO_R, LED_GPIO_G, LED_GPIO_B);
    led.change_led_color(0);
    while(1){
        msg_t msg;
        msg_receive(&msg);

        if (led_always == 5) { // 当 led_always 为5时，按照运动状态改变LED颜色
            if (msg.content.value == Stationary) {
                led.change_led_color(COLOR_NONE);
                printf("[LED_THREAD]: LED TURN OFF!!\n");
            } else if (msg.content.value == Tilted) {
                led.change_led_color(COLOR_RED);
                printf("[LED_THREAD]: LED TURN RED!!\n");
            } else if (msg.content.value == Rotating) {
                led.change_led_color(COLOR_BLUE);
                printf("[LED_THREAD]: LED TURN BLUE!!\n");
            }
        }
    }
}
```

```

        } else if (msg.content.value == Moving) {
            led.change_led_color(COLOR_GREEN);
            printf("[LED_THREAD]: LED TURN GREEN!!\n");
        }
    } else { // 当 led_always 不为5时, LED灯常亮特定颜色
        if (led_always == 0) {
            led.change_led_color(COLOR_NONE);
            printf("[LED_THREAD]: LED stay none.\n");
        } else if (led_always == 1) {
            led.change_led_color(COLOR_RED);
            printf("[LED_THREAD]: LED stay red.\n");
        } else if (led_always == 2) {
            led.change_led_color(COLOR_BLUE);
            printf("[LED_THREAD]: LED stay blue.\n");
        } else if (led_always == 3) {
            led.change_led_color(COLOR_GREEN);
            printf("[LED_THREAD]: LED stay green.\n");
        }
    }
    delay_ms(10);
}
return NULL;
}

```

2. predict motion 后使 led 灯亮不同颜色的灯，并将 predict 的结果传给蓝牙模块（这里使用了一个全局变量 `current_motion`）

```
static int current_motion = 0; //默认无状态则为静止
```

```

int data_len = SAMPLES_PER_GESTURE - 0,
delay_ms(200);
// Main loop
int ret = 0;
string motions[class_num] = {"Stationary", "Tilted", "Rotating", "Moving"};

while (1) {
    delay_ms(predict_interval_ms);
    // Log the delay between predictions
    LOG_INFO("[MOTION_THREAD] Prediction gap: %d ms\n", predict_interval_ms);

    // Read sensor data
    get_imu_data(mpu, imu_data);

    // Log the current threshold before prediction
    LOG_INFO("[MOTION_THREAD] Using threshold: %.2f\n", threshold);

    ret = predict(imu_data, data_len, threshold, class_num);

    // Send message to LED thread based on motion
    msg_t msg;
    msg.content.value = ret;
    msg_send(&msg, _led_pid); // Send message to LED control thread

    // Log the prediction result
    LOG_INFO("[MOTION_THREAD] Prediction result: %d, %s\n", ret, motions[ret].c_str());

    // Update global motion state
    current_motion = ret;
    LOG_INFO("Predict: %d, %s\n", ret, motions[ret].c_str());
}
return NULL;
}

```

3. 添加自定义 uuid 来创建可读写特性

```

// input your code, 自定义想要的UUID
/* UUID = 1bce38b3-d137-48ff-a13e-033e14c7a335 */
static const ble_uuid128_t gatt_svr_svc_rw_demo_uuid
    = {[128], {0x15, 0xa3, 0xc7, 0x14, 0x3e, 0x03, 0x3e, 0xa1, 0xff,
              0x48, 0x37, 0xd1, 0xb3, 0x38, 0xce, 0x1b}};

//35f2led readwrite
static const ble_uuid128_t gatt_svr_chr_led_uuid
    = {[128], {0x62, 0x17, 0x99, 0x7e, 0x50, 0x27, 0x38, 0xba, 0x3b,
              0x4f, 0x70, 0x30, 0x86, 0x83, 0xf2, 0x35}};

//00ffmotion read
static const ble_uuid128_t gatt_svr_chr_motion_uuid
    = {[128], {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
              0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00}};

//0201threshold readwrite
static const ble_uuid128_t gatt_svr_chr_threshold_uuid
    = {[128], {0x13, 0x24, 0x35, 0x46, 0x57, 0x68, 0x79, 0x8a, 0x91,
              0xac, 0xbd, 0xce, 0xdf, 0xf0, 0x01, 0x02}};

//0302frequency readwrite
static const ble_uuid128_t gatt_svr_chr_frequency_uuid
    = {[128], {0x14, 0x25, 0x36, 0x47, 0x58, 0x69, 0x7a, 0x8b, 0x92,
              0xad, 0xbe, 0xcf, 0xe0, 0xf1, 0x02, 0x03}};

```

4. 定义各个服务和特性

```

static const struct ble_gatt_svc_def gatt_svr_svcs[] = {
/*
 * Service: Read/Write Demo
 */
/* given characteristics */
// input your code, 请按需求更改。
{
    /* Service: Read/Write Demo */
    .type = BLE_GATT_SVC_TYPE_PRIMARY,
    .uuid = (ble_uuid_t*)&gatt_svr_svc_rw_demo_uuid.u,
    .characteristics = (struct ble_gatt_chr_def[]) {
        /* Characteristic: Read/Write Demo write */
        .uuid = (ble_uuid_t*)&gatt_svr_chr_led_uuid.u,
        .access_cb = gatt_svr_chr_access_rw_demo,
        .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_WRITE,
        // .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_WRITE_NO_RSP,
    }, {
        .uuid = (ble_uuid_t*)&gatt_svr_chr_motion_uuid.u,
        .access_cb = gatt_svr_chr_access_rw_demo,
        .flags = BLE_GATT_CHR_F_READ,
    }, {
        .uuid = (ble_uuid_t*)&gatt_svr_chr_threshold_uuid.u,
        .access_cb = gatt_svr_chr_access_rw_demo,
        .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_WRITE,
    }, {
        .uuid = (ble_uuid_t*)&gatt_svr_chr_frequency_uuid.u,
        .access_cb = gatt_svr_chr_access_rw_demo,
        .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_WRITE,
    }, {
        /* No more characteristics in this service */
    }, {
        /* No more services */
    },
};

```

5. 设置特性访问函数

```

switch(ctxt->op){
    case BLE_GATT_ACCESS_OP_READ_CHR:
        if (ble_uuid_cmp(ctxt->chr->uuid, &gatt_svr_chr_motion_uuid.u) == 0) {
            // 读取当前预测的运动状态
            rc = os_mbuf_append(ctxt->om, &current_motion, sizeof(current_motion));
            LOG_INFO("[READ] current_motion = %d\n", current_motion);
            return rc;
        } else if (ble_uuid_cmp(ctxt->chr->uuid, &gatt_svr_chr_led_uuid.u) == 0) {
            // 读取当前的led灯状态
            rc = os_mbuf_append(ctxt->om, &current_motion, sizeof(current_motion));
            LOG_INFO("[READ] led_status = %d\n", current_motion);
            return rc;
        } else if (ble_uuid_cmp(ctxt->chr->uuid, &gatt_svr_chr_threshold_uuid.u) == 0) {
            // 读取当前阈值
            rc = os_mbuf_append(ctxt->om, &threshold, sizeof(threshold));
            LOG_INFO("[READ] threshold = %.2f\n", threshold);
            return rc;
        } else if (ble_uuid_cmp(ctxt->chr->uuid, &gatt_svr_chr_frequency_uuid.u) == 0) {
            // 读取数据采集频率
            rc = os_mbuf_append(ctxt->om, &collect_interval_ms, sizeof(collect_interval_ms));
            LOG_INFO("[READ] collect_interval_ms = %d\n", collect_interval_ms);
            return rc;
        }
        break;
}

```

```

switch(ctx->op){
    case BLE_GATT_ACCESS_OP_READ_CHR:
    case BLE_GATT_ACCESS_OP_WRITE_CHR:
        uint16_t om_len;
        om_len = OS_MBUF_PKTLEN(ctx->om);
        if (ble_uuid_cmp(ctx->chr->uuid, &gatt_svr_chr_threshold_uuid.u) == 0) {
            // 写入新的阈值
            rc = ble_hs_mbuf_to_flat(ctx->om, &threshold, sizeof(threshold), &om_len);
            LOG_INFO("[WRITE] new value of threshold: %.2f\n", threshold);
            //predict_interval_ms[om_len] = '\0';
        } else if (ble_uuid_cmp(ctx->chr->uuid, &gatt_svr_chr_frequency_uuid.u) == 0) {
            // 写入新的采集频率
            rc = ble_hs_mbuf_to_flat(ctx->om, &collect_interval_ms, sizeof(collect_interval_ms), &om_len);
            LOG_INFO("[WRITE] new value of collect_interval_ms: %d\n", collect_interval_ms);
            //collect_interval_ms[om_len] = '\0';
        } else if (ble_uuid_cmp(ctx->chr->uuid, &gatt_svr_chr_led_uuid.u) == 0) {
            // 写入新的LED灯常亮
            rc = ble_hs_mbuf_to_flat(ctx->om, &led_always, sizeof(led_always), &om_len);
            if(Led_always != 5){
                msg_t msg;
                msg.content.value = led_always;
                msg_send(&msg, _led_pid);
            }else{
                LOG_INFO("[WRITE] led continuing prediction: \n");
            }
        }
        break;
    default:
        rc = 1;
        return BLE_ATT_ERR_UNLIKELY;
}

```

b. 手机端或电脑端读写GATT服务特性过程截图(如读到的ESP32设备运动状态信息和LED灯状态信息，以及写操作结束后，APP或程序结果截图)

1. 读 Stationary 状态下 LED 不亮灯

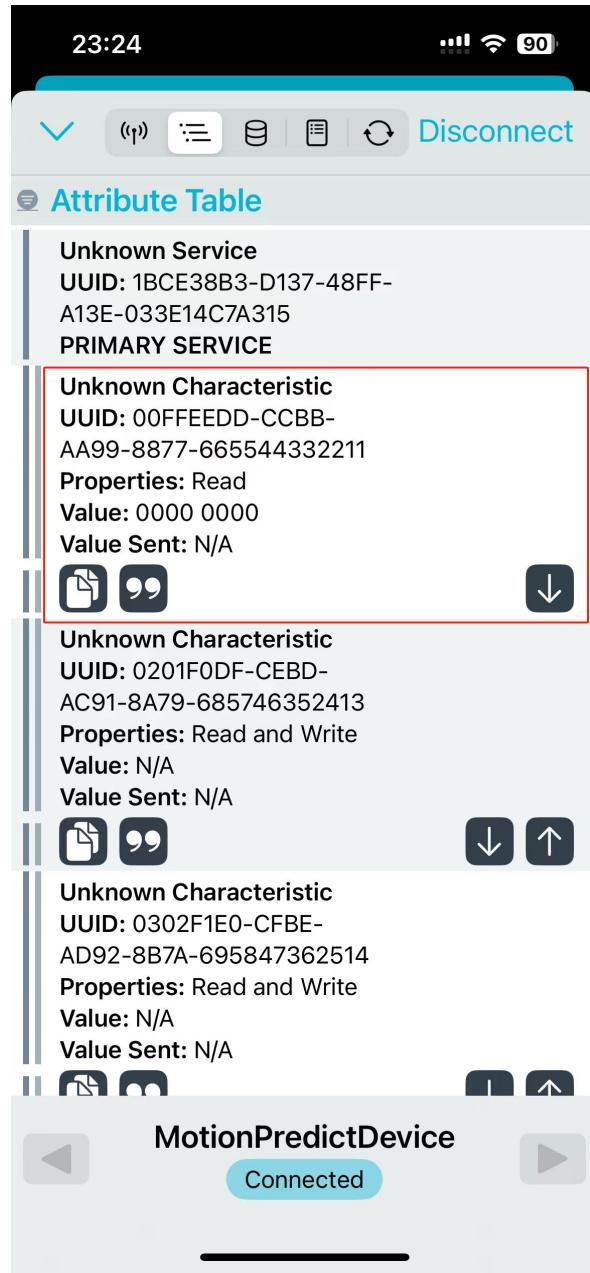
LOG 输出：

```

2024-10-15 08:23:57,271 # [MOTION_THREAD] Prediction result: 0, stationary
2024-10-15 08:23:57,274 # Predict: 0, Stationary
2024-10-15 08:23:57,275 # [LED_THREAD]: LED TURN OFF!!
2024-10-15 08:23:57,765 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 08:23:57,868 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 08:23:57,873 # -----

```

手机蓝牙连接读取：

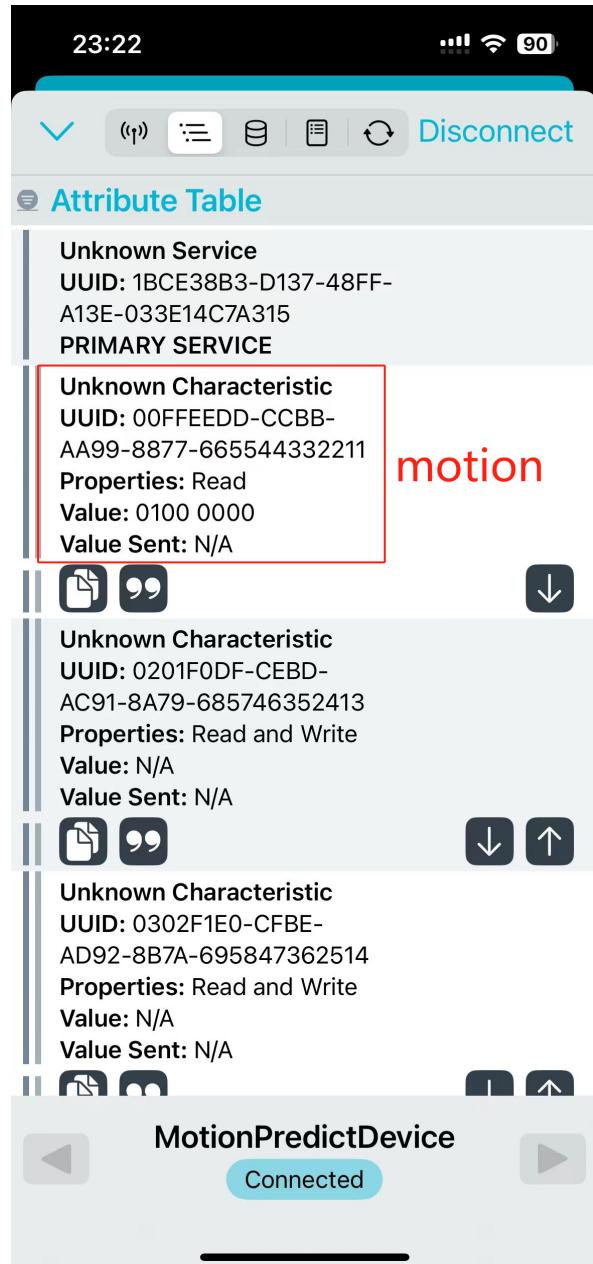


2. 读 tilted 状态下 LED 亮红灯

LOG 输出：

```
2024-10-15 08:17:12,192 # [MOTION_THREAD] Prediction result: 1, Tilted
2024-10-15 08:17:12,193 # Predict: 1, Tilted
2024-10-15 08:17:12,194 # [LED_THREAD]: LED TURN RED!!
2024-10-15 08:17:12,684 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 08:17:12,787 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 08:17:12,792 # -----
```

手机蓝牙连接读取：

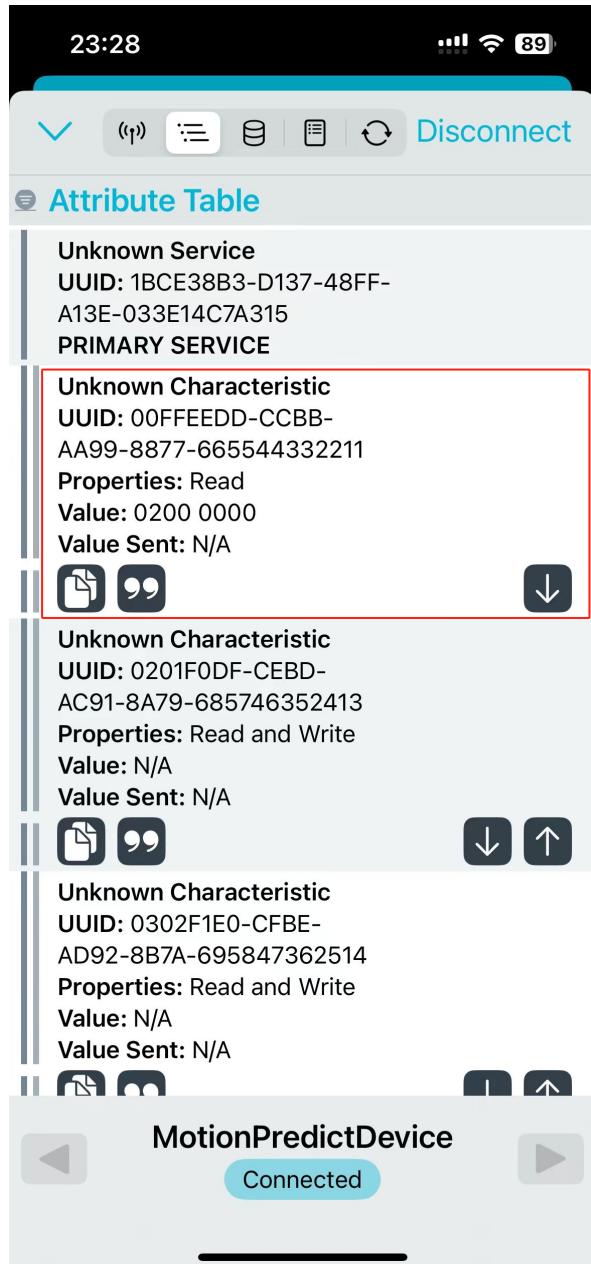


3. 读 Rotating 状态下 LED 亮蓝灯

LOG 输出:

```
2024-10-15 08:25:31,037 # Motion prediction: 2
2024-10-15 08:25:31,042 # [MOTION_THREAD] Prediction result: 2, Rotating
2024-10-15 08:25:31,043 # Predict: 2, Rotating
2024-10-15 08:25:31,045 # [LED_THREAD]: LED TURN BLUE!!
2024-10-15 08:25:31,535 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 08:25:31,637 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 08:25:31,642 # -----
```

手机蓝牙读取:

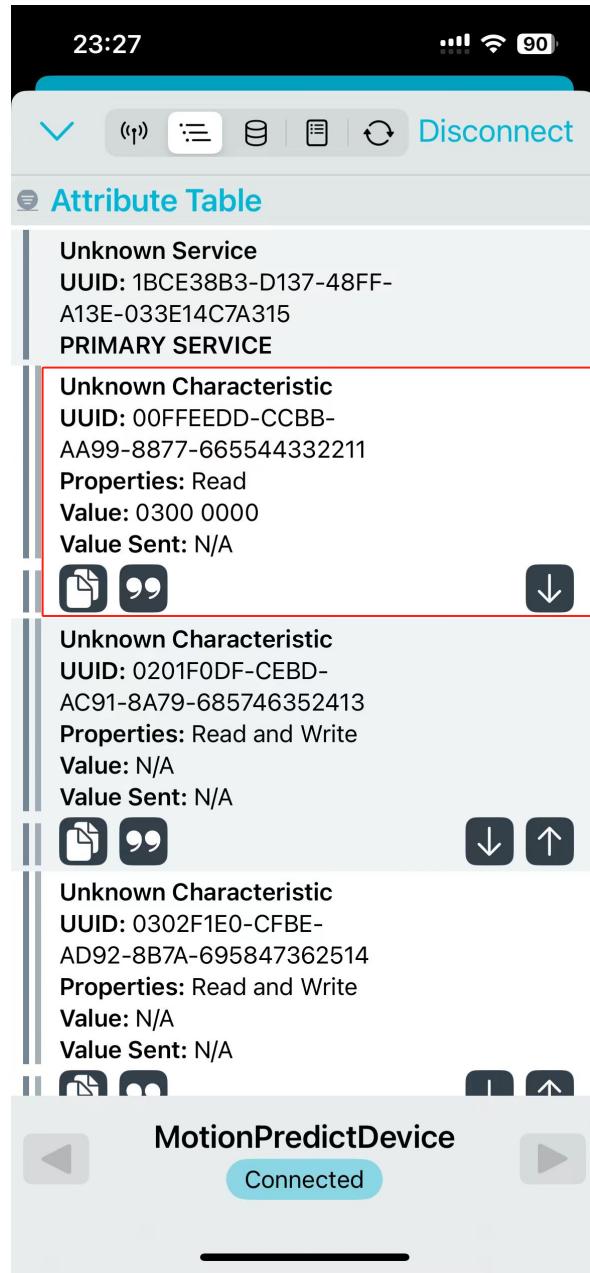


4. 读 Moving 状态下 LED 亮绿灯

LOG 输出：

```
2024-10-15 08:30:00,731 # Motion prediction: 3
2024-10-15 08:30:00,736 # [MOTION_THREAD] Prediction result: 3, Moving
2024-10-15 08:30:00,736 # Predict: 3, Moving
2024-10-15 08:30:00,738 # [LED_THREAD]: LED TURN GREEN!!
2024-10-15 08:30:01,228 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 08:30:01,331 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 08:30:01,336 # -----
```

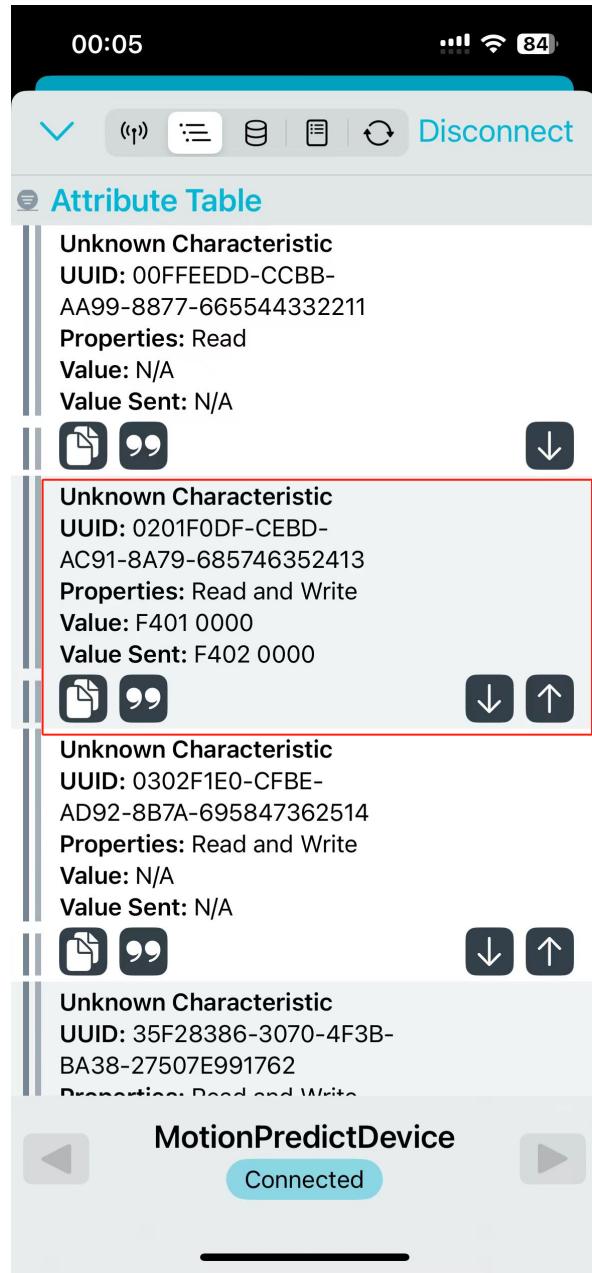
手机蓝牙读取：



c. 用户端蓝牙控制ESP32设备前后，设备变化（可适当添加日志，如从侧面体现预测间隙的时间延迟，打印模型预测阈值变化，LED状态变化等）

1. 修改预测间隙的时间延迟

手机端蓝牙修改：

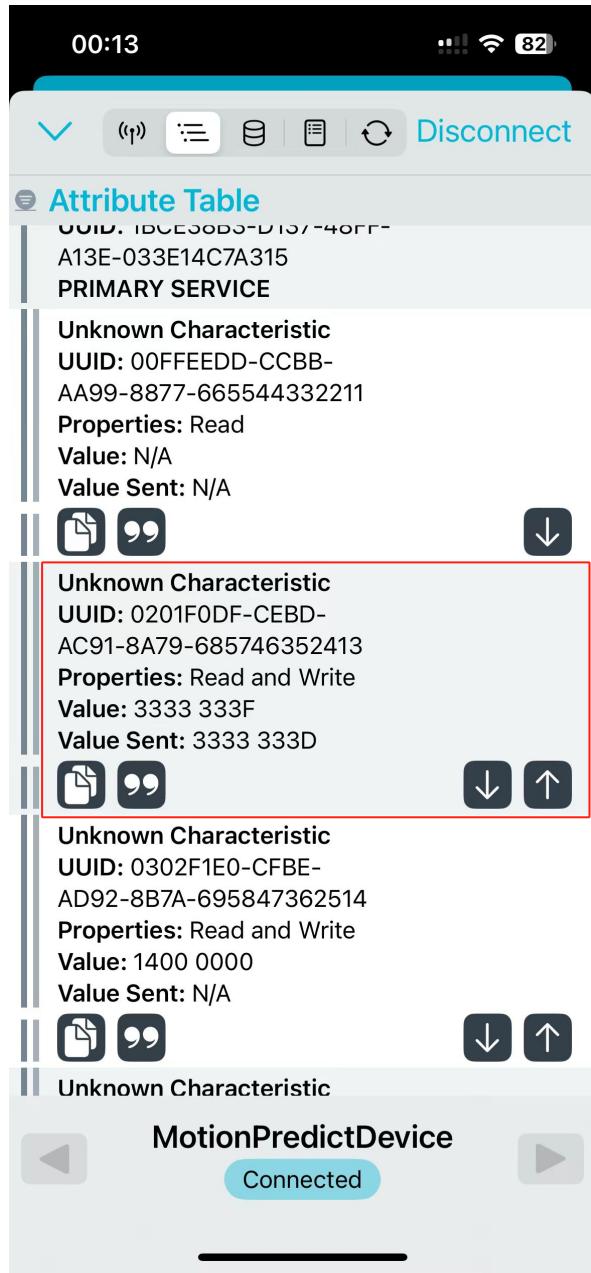


LOG 输出：

```
2024-10-15 09:04:02,313 # Predict: 0, Stationary
2024-10-15 09:04:02,315 # [LED_THREAD]: LED TURN OFF!!
2024-10-15 09:04:03,061 # [MOTION_THREAD] Prediction gap: 756 ms
2024-10-15 09:04:03,163 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 09:04:03,168 # -----
```

2. 修改模型预测阈值变化

手机端蓝牙修改：



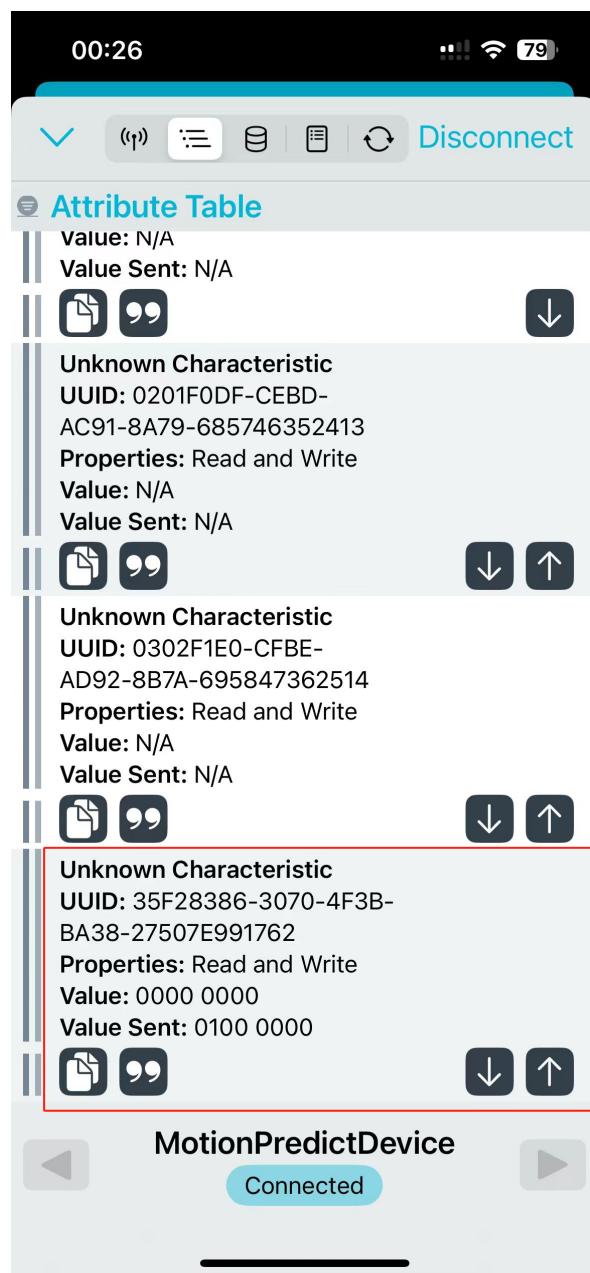
LOG 输出:

```
2024-10-15 09:12:37,750 # [MOTION_THREAD] Prediction result: 1, Tilted
2024-10-15 09:12:37,751 # Predict: 1, Tilted
2024-10-15 09:12:37,752 # [LED_THREAD]: LED TURN RED!!
2024-10-15 09:12:38,242 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 09:12:38,345 # [MOTION_THREAD] Using threshold: 0.04
2024-10-15 09:12:38,350 # -----
```

3. led 灯状态控制

这里设计了 led 灯可以由蓝牙实现状态输入，蓝牙输入 0100 0000, 0200 0000, 0300 0000 分别表示红色、蓝色、绿色的常亮状态，输入 0500 0000 表示退出常亮状态。

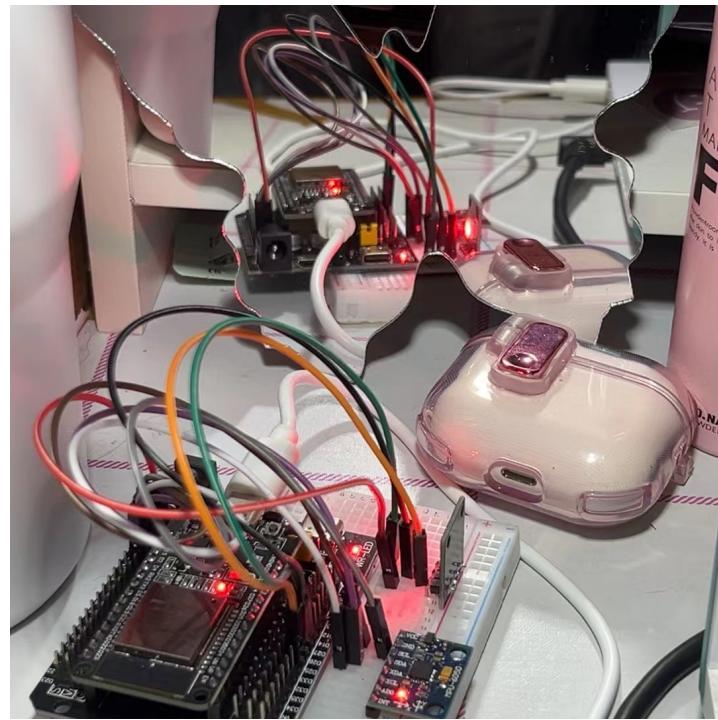
i. 手机端蓝牙设置红灯常亮



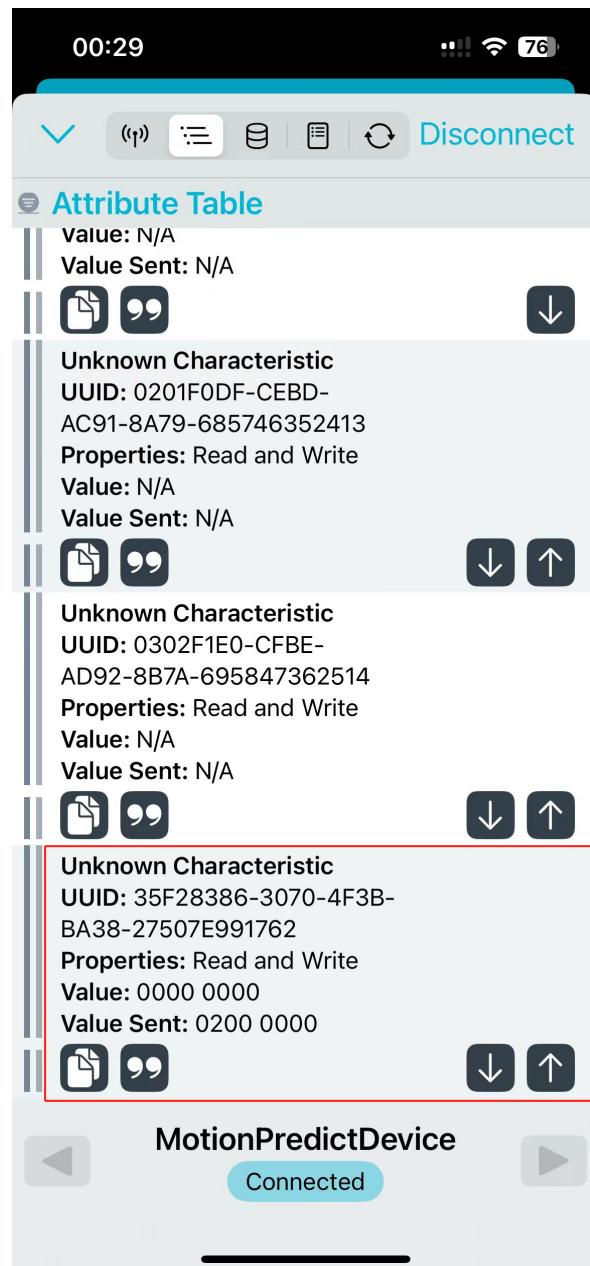
LOG 输出:

```
2024-10-15 09:27:28,229 # Predict: 0, Stationary
2024-10-15 09:27:28,230 # [LED_THREAD]: LED stay red.
2024-10-15 09:27:28,720 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 09:27:28,823 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 09:27:28,828 # -----
```

LED 灯光显示:



ii. 手机端蓝牙设置蓝灯常亮



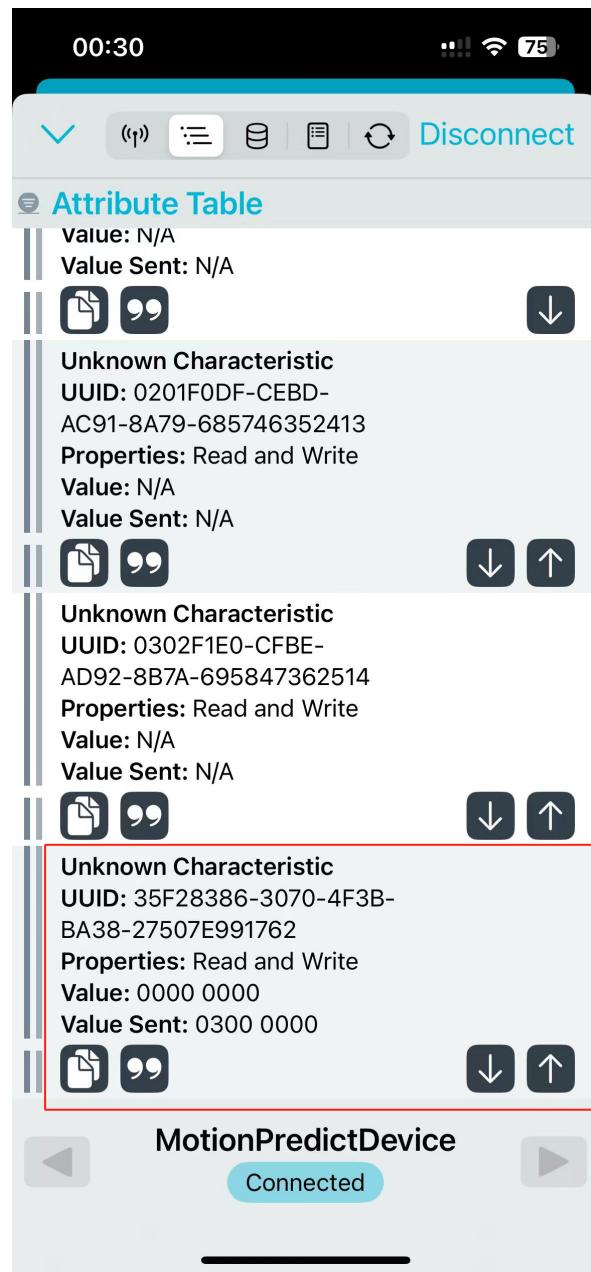
LOG 输出:

```
2024-10-15 09:30:10,699 # Predict: 0, Stationary
2024-10-15 09:30:10,700 # [LED_THREAD]: LED stay blue.
2024-10-15 09:30:11,190 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 09:30:11,293 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 09:30:11,298 # -----
```

LED 灯光显示:



iii. 手机端蓝牙设置绿灯常亮



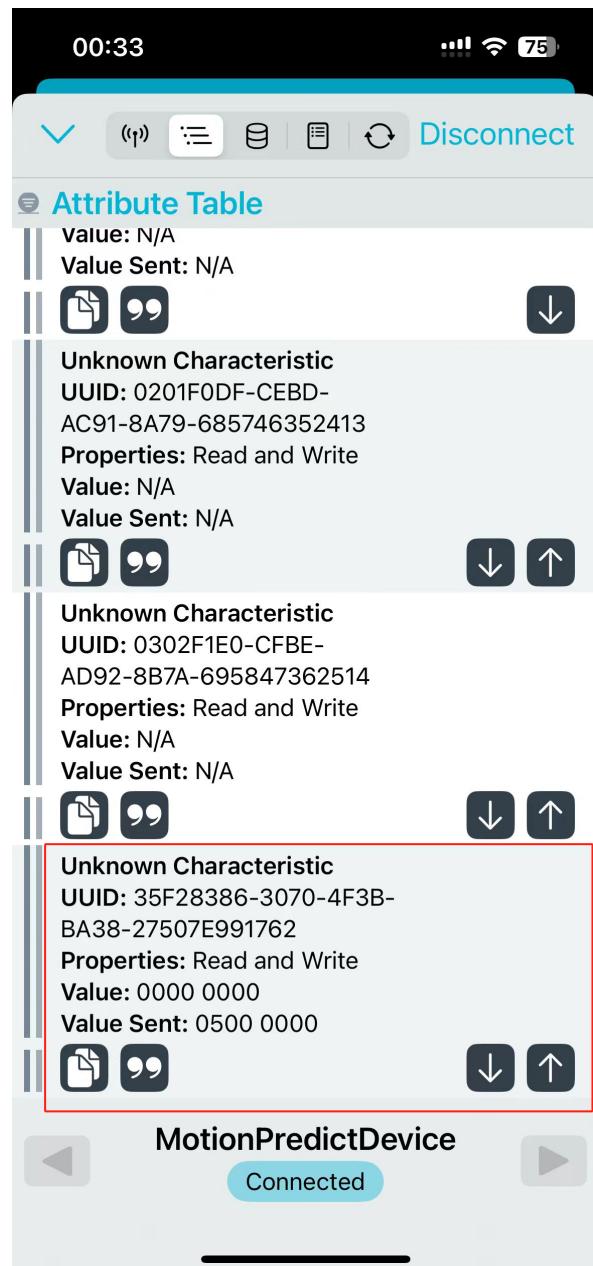
LOG 输出:

```
2024-10-15 09:30:59,744 # Predict: 0, Stationary
2024-10-15 09:30:59,747 # [LED_THREAD]: LED stay green.
2024-10-15 09:31:00,236 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 09:31:00,339 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 09:31:00,344 # -----
```

LED 灯光显示:



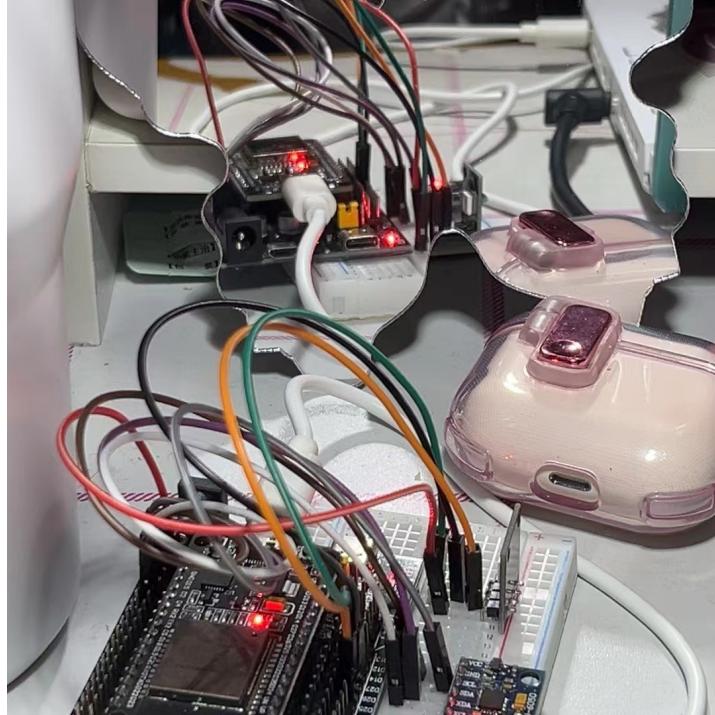
iiii. 手机端蓝牙设置退出常亮模式



LOG 输出:

```
2024-10-15 09:34:17,158 # [MOTION_THREAD] Prediction result: 0, Stationary
2024-10-15 09:34:17,160 # Predict: 0, Stationary
2024-10-15 09:34:17,162 # [LED_THREAD]: LED TURN OFF!!
2024-10-15 09:34:17,652 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-15 09:34:17,754 # [MOTION_THREAD] Using threshold: 0.70
2024-10-15 09:34:17,759 # -----
```

LED 灯光显示:



d. 若自定义设备蓝牙MAC地址，需要默认的MAC地址和当前MAC地址信息截图以及手机或电脑扫描到的MAC地址截图？或多用户连接，请展示展示相关信息的截图？否则，则忽略.

```
// 获取蓝牙设备的默认 MAC 地址
uint8_t own_addr_type;
uint8_t own_addr[6];

uint8_t custom_addr[6] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x02};
int r = 0;
r = ble_hs_id_set_rnd(custom_addr);
if(r != 0){
    LOG_INFO("Failed to set customized mac address.");
    return 1;
}

ble_hs_id_infer_auto(0, &own_addr_type);
ble_hs_id_copy_addr(own_addr_type, own_addr, NULL);
```

默认的蓝牙地址见上面截图，当前的 MAC 地址:

```
↳xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xmain(): This is RIOT! (Version: main()): This is RIOT! (Version: 2024.10-devel-190-g20f732)
2024-10-21 06:41:47,998 # LED Controller initialized with (RGB: GPIO26, GPIO25, GPIO27)
2024-10-21 06:41:48,000 # [MAIN] LED_PID: 6
2024-10-21 06:41:48,001 # [MAIN_THREAD] DEVICE_ID:0x34
2024-10-21 06:41:48,001 # [MAIN] MOTION_PID: 7
2024-10-21 06:41:48,002 # Default MAC address: 02:05:04:03:02:01
2024-10-21 06:41:48,293 # [MOTION_THREAD] Prediction gap: 500 ms
2024-10-21 06:41:48,395 # [MOTION_THREAD] Using threshold: 0.70
2024-10-21 06:41:48,400 # -----
```

7 实验结果与分析

基于该实验，总结开发一个NimBLE GATT服务器的基本流程。

开发一个NimBLE GATT服务器的基本流程可以分为以下步骤：

1. 初始化NimBLE环境

在实现GATT服务器之前，首先需要初始化NimBLE的BLE环境。这包括：

- 初始化NimBLE堆栈。
- 配置GAP (Generic Access Profile) 相关参数，例如设备名称、MAC地址等。
- 开启广告功能，使设备可以被扫描到并连接。

2. 定义GATT服务与特性

NimBLE GATT服务器的核心部分是定义服务和特性。每个服务包含一个或多个特性，特性可以是可读、可写、可通知等。

步骤：

- 定义GATT服务和特性UUID。
- 为每个特性设置访问权限（读/写）。
- 为每个特性绑定回调函数，在设备读/写该特性时触发。

3. 实现GATT服务回调函数

每个特性都可以绑定一个回调函数来处理读写请求。当客户端对特性进行读写时，这个回调函数会被调用。

读操作：

- 当客户端请求读取某个特性时，通过 `os_mbuf_append()` 将数据返回给客户端。

写操作：

- 当客户端写入数据时，通过 `ble_hs_mbuf_to_flat()` 将数据从客户端缓冲区中读取出来，并根据需求处理数据。

4. 启动GATT服务

定义好GATT服务和特性后，需要将它们注册到NimBLE GATT服务器中。这个过程包括计算服务配置并启动GATT服务器。

5. 启动设备广告

设备需要通过广播（广告）来让客户端发现。可以配置不同的广告模式，例如可连接模式或不可连接模式。

6. 处理连接和断开事件

为了处理BLE连接和断开连接事件，您需要实现一个事件处理回调函数，并在其中处理这些BLE状态变化。

7. 运行主程序

最终，将所有初始化和服务注册放入主程序中运行。确保所有服务启动后，设备开始广播。

8. 测试与调试

可以使用以下工具测试BLE GATT服务器：

- **nRF Connect (iOS/Android)**: 连接到您的设备，读写特性数据。
- **gatttool (Linux)**: 命令行工具，可以在Linux系统上与BLE设备进行交互。
- **Bluetooth LE Explorer (Windows)**: Windows平台上的BLE调试工具。

这套流程能帮助开发者构建一个功能完整的BLE GATT服务器，用于物联网、穿戴设备等场景中的蓝牙低功耗应用。

8 附录：源码

main.cpp

```
using namespace std;
void setup();
int predict(float *imu_data, int data_len, float threshold, int class_num);
#define LED_GPIO_R GPIO26
#define LED_GPIO_G GPIO25
#define LED_GPIO_B GPIO27
#define g_acc (9.8)
#define class_num (4)
#define SAMPLES_PER_GESTURE (10)
#define THREAD_STACKSIZE          (THREAD_STACKSIZE_IDLE)
#define LED_MSG_TYPE_RED          (0x3111)
#define LED_MSG_TYPE_NONE         (0x3110)
static char stack_for_led_thread[THREAD_STACKSIZE];
static char stack_for_motion_thread[THREAD_STACKSIZE];

#define DEMO_BUFFER_SIZE 100
// static char rm_demo_write_data[DEMO_BUFFER_SIZE] = "This characteristic is
read- and writeable!";
#define STR_ANSWER_BUFFER_SIZE 100
// static char str_answer[STR_ANSWER_BUFFER_SIZE];

static int current_motion = 0;
enum MoveState{Stationary, Tilted, Rotating, Moving, MovX, MovY};
static int led_always = 5; //5表示正常predict, 0123分别表示不同颜色的常亮状态
float threshold = 0.7;

static kernel_pid_t _led_pid;
static kernel_pid_t _motion_pid;
struct MPU6050Data
{
    float ax, ay, az; // acceler_x_axis, acceler_y_axis, acceler_z_axis
    float gx, gy, gz; // gyroscope_x_axis, gyroscope_y_axis, gyroscope_z_axis
};
void delay_ms(uint32_t sleep_ms)
{
```

```

ztimer_sleep(ZTIMER_USEC, sleep_ms * US_PER_MS);
return;
}
// static int r_led_state = 0;
void *_led_thread(void *arg)
{
    (void) arg;
    LEDController led(LED_GPIO_R, LED_GPIO_G, LED_GPIO_B);
    led.change_led_color(0);
    while(1){
        // Input your codes
        // Wait for a message to control the LED
        // Display different light colors based on the motion state of the
device.

        msg_t msg;
        msg_receive(&msg);

        if (led_always == 5) { // 当 led_always 为5时, 按照运动状态改变LED颜色
            if (msg.content.value == Stationary) {
                led.change_led_color(COLOR_NONE);
                printf("[LED_THREAD]: LED TURN OFF!!\n");
            } else if (msg.content.value == Tilted) {
                led.change_led_color(COLOR_RED);
                printf("[LED_THREAD]: LED TURN RED!!\n");
            } else if (msg.content.value == Rotating) {
                led.change_led_color(COLOR_BLUE);
                printf("[LED_THREAD]: LED TURN BLUE!!\n");
            } else if (msg.content.value == Moving) {
                led.change_led_color(COLOR_GREEN);
                printf("[LED_THREAD]: LED TURN GREEN!!\n");
            }
        } else { // 当 led_always 不为5时, LED灯常亮特定颜色
            if (led_always == 0) {
                led.change_led_color(COLOR_NONE);
                printf("[LED_THREAD]: LED stay none.\n");
            } else if (led_always == 1) {
                led.change_led_color(COLOR_RED);
                printf("[LED_THREAD]: LED stay red.\n");
            } else if (led_always == 2) {
                led.change_led_color(COLOR_BLUE);
                printf("[LED_THREAD]: LED stay blue.\n");
            } else if (led_always == 3) {
                led.change_led_color(COLOR_GREEN);
                printf("[LED_THREAD]: LED stay green.\n");
            }
        }
        delay_ms(10);
    }
    return NULL;
}

float gyro_fs_convert = 1.0;
float accel_fs_convert;
static int collect_interval_ms = 20;
static int predict_interval_ms = 500;

```

```

void get_imu_data(MPU6050 mpu, float *imu_data){
    int16_t ax, ay, az, gx, gy, gz;
    for(int i = 0; i < SAMPLES_PER_GESTURE; ++i)
    {
        i += 1;
        /* code */
        delay_ms(collect_interval_ms);
        mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        imu_data[i*6 + 0] = ax / accel_fs_convert;
        imu_data[i*6 + 1] = ay / accel_fs_convert;
        imu_data[i*6 + 2] = az / accel_fs_convert;
        imu_data[i*6 + 3] = gx / gyro_fs_convert;
        imu_data[i*6 + 4] = gy / gyro_fs_convert;
        imu_data[i*6 + 5] = gz / gyro_fs_convert;
    }
}

void *_motion_thread(void *arg)
{
    (void) arg;
    // Initialize MPU6050 sensor
    MPU6050 mpu;
    // get mpu6050 device id
    uint8_t device_id = mpu.getDeviceID();
    printf("[IMU_THREAD] DEVICE_ID:0x%x\n", device_id);
    mpu.initialize();
    // Configure gyroscope and accelerometer full scale ranges
    uint8_t gyro_fs = mpu.getFullScaleGyroRange();
    uint8_t accel_fs_g = mpu.getFullScaleAccelRange();
    uint16_t accel_fs_real = 1;

    // Convert gyroscope full scale range to conversion factor
    if (gyro_fs == MPU6050_GYRO_FS_250)
        gyro_fs_convert = 131.0;
    else if (gyro_fs == MPU6050_GYRO_FS_500)
        gyro_fs_convert = 65.5;
    else if (gyro_fs == MPU6050_GYRO_FS_1000)
        gyro_fs_convert = 32.8;
    else if (gyro_fs == MPU6050_GYRO_FS_2000)
        gyro_fs_convert = 16.4;
    else
        printf("[IMU_THREAD] Unknown GYRO_FS: 0x%x\n", gyro_fs);

    // Convert accelerometer full scale range to real value
    if (accel_fs_g == MPU6050_ACCEL_FS_2)
        accel_fs_real = g_acc * 2;
    else if (accel_fs_g == MPU6050_ACCEL_FS_4)
        accel_fs_real = g_acc * 4;
    else if (accel_fs_g == MPU6050_ACCEL_FS_8)
        accel_fs_real = g_acc * 8;
    else if (accel_fs_g == MPU6050_ACCEL_FS_16)
        accel_fs_real = g_acc * 16;
    else
        printf("[IMU_THREAD] Unknown ACCEL_FS: 0x%x\n", accel_fs_g);

    // calculate accelerometer conversion factor
    accel_fs_convert = 32768.0 / accel_fs_real;
}

```

```

float imu_data[SAMPLES_PER_GESTURE * 6] = {0};
int data_len = SAMPLES_PER_GESTURE * 6;
delay_ms(200);
// Main loop
int ret = 0;
string motions[class_num] = {"Stationary", "Tilted", "Rotating", "Moving"};

while (1) {
    delay_ms(predict_interval_ms);
    // Log the delay between predictions
    LOG_INFO("[MOTION_THREAD] Prediction gap: %d ms\n", predict_interval_ms);

    // Read sensor data
    get_imu_data(mpu, imu_data);

    // Log the current threshold before prediction
    LOG_INFO("[MOTION_THREAD] Using threshold: %.2f\n", threshold);

    ret = predict(imu_data, data_len, threshold, class_num);

    // Send message to LED thread based on motion
    msg_t msg;
    msg.content.value = ret;
    msg_send(&msg, _led_pid); // send message to LED control thread

    // Log the prediction result
    LOG_INFO("[MOTION_THREAD] Prediction result: %d, %s\n", ret,
motions[ret].c_str());

    // Update global motion state
    current_motion = ret;
    LOG_INFO("Predict: %d, %s\n", ret, motions[ret].c_str());
}

return NULL;
}

// input your code, 自定义想要的UUID
/* UUID = 1bce38b3-d137-48ff-a13e-033e14c7a335 */
static const ble_uuid128_t gatt_svr_svc_rw_demo_uuid
= {{128}, {0x15, 0xa3, 0xc7, 0x14, 0x3e, 0x03, 0x3e, 0xa1, 0xff,
0x48, 0x37, 0xd1, 0xb3, 0x38, 0xce, 0x1b}};

//35f21ed readwrite
static const ble_uuid128_t gatt_svr_chr_led_uuid
= {{128}, {0x62, 0x17, 0x99, 0x7e, 0x50, 0x27, 0x38, 0xba, 0x3b,
0x4f, 0x70, 0x30, 0x86, 0x83, 0xf2, 0x35}};

//00ffmotion read
static const ble_uuid128_t gatt_svr_chr_motion_uuid
= {{128}, {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00}};

//0201threshold readwrite
static const ble_uuid128_t gatt_svr_chr_threshold_uuid
= {{128}, {0x13, 0x24, 0x35, 0x46, 0x57, 0x68, 0x79, 0x8a, 0x91,
0xac, 0xbd, 0xce, 0xdf, 0xf0, 0x01, 0x02}};

//0302frequency readwrite
static const ble_uuid128_t gatt_svr_chr_frequency_uuid

```

```

= {{128}, {0x14, 0x25, 0x36, 0x47, 0x58, 0x69, 0x7A, 0x8B, 0x92,
    0xAD, 0xBE, 0xCF, 0xE0, 0xF1, 0x02, 0x03}};

static int gatt_svr_chr_access_rw_demo(
    uint16_t conn_handle, uint16_t attr_handle,
    struct ble_gatt_access_ctxt *ctxt, void *arg);

/* define several bluetooth services for our device */
static const struct ble_gatt_svc_def gatt_svr_svcs[] = {
    /*
     * access_cb defines a callback for read and write access events on
     * given characteristics
     */
    // input your code, 请按需求更改。
    {

        /* Service: Read/Write Demo */
        .type = BLE_GATT_SVC_TYPE_PRIMARY,
        .uuid = (ble_uuid_t*) &gatt_svr_svc_rw_demo_uuid.u,
        .characteristics = (struct ble_gatt_chr_def[]) { {
            /* Characteristic: Read/Write Demo write */
            .uuid = (ble_uuid_t*) &gatt_svr_chr_led_uuid.u,
            .access_cb = gatt_svr_chr_access_rw_demo,
            .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_WRITE,
            // .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_WRITE_NO_RSP,
        }, {
            .uuid = (ble_uuid_t*) &gatt_svr_chr_motion_uuid.u,
            .access_cb = gatt_svr_chr_access_rw_demo,
            .flags = BLE_GATT_CHR_F_READ,
        }, {
            .uuid = (ble_uuid_t*) &gatt_svr_chr_threshold_uuid.u,
            .access_cb = gatt_svr_chr_access_rw_demo,
            .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_WRITE,
        }, {
            .uuid = (ble_uuid_t*) &gatt_svr_chr_frequency_uuid.u,
            .access_cb = gatt_svr_chr_access_rw_demo,
            .flags = BLE_GATT_CHR_F_READ | BLE_GATT_CHR_F_WRITE,
        }, {
            0, /* No more characteristics in this service */
        }, }
    },
    {
        0, /* No more services */
    },
};

static int gatt_svr_chr_access_rw_demo(
    uint16_t conn_handle, uint16_t attr_handle,
    struct ble_gatt_access_ctxt *ctxt, void *arg)
{
    (void) conn_handle;
    (void) attr_handle;
    (void) arg;
    // input your code
    int rc = 0;
}

```

```

switch(ctxt->op){
    case BLE_GATT_ACCESS_OP_READ_CHR:
        if (ble_uuid_cmp(ctxt->chr->uuid, &gatt_svr_chr_motion_uuid.u) == 0)
    {
        // 读取当前预测的运动状态
        rc = os_mbuf_append(ctxt->om, &current_motion,
        sizeof(current_motion));
        LOG_INFO("[READ] current_motion = %d\n", current_motion);
        return rc;
    } else if (ble_uuid_cmp(ctxt->chr->uuid, &gatt_svr_chr_led_uuid.u) ==
0){
        //读取当前的led灯状态
        rc = os_mbuf_append(ctxt->om, &current_motion,
        sizeof(current_motion));
        LOG_INFO("[READ] led_status = %d\n", current_motion);
        return rc;
    }else if (ble_uuid_cmp(ctxt->chr->uuid,
&gatt_svr_chr_threshold_uuid.u) == 0) {
        // 读取当前阈值
        rc = os_mbuf_append(ctxt->om, &threshold, sizeof(threshold));
        LOG_INFO("[READ] threshold = %.2f\n", threshold);
        return rc;
    } else if (ble_uuid_cmp(ctxt->chr->uuid,
&gatt_svr_chr_frequency_uuid.u) == 0) {
        // 读取数据采集频率
        rc = os_mbuf_append(ctxt->om, &collect_interval_ms,
        sizeof(collect_interval_ms));
        LOG_INFO("[READ] collect_interval_ms = %d\n",
collect_interval_ms);
        return rc;
    }
    break;
} case BLE_GATT_ACCESS_OP_WRITE_CHR:
    uint16_t om_len;
    om_len = OS_MBUF_PKTLEN(ctxt->om);
    if (ble_uuid_cmp(ctxt->chr->uuid, &gatt_svr_chr_threshold_uuid.u) ==
0) {
        // 写入新的阈值
        rc = ble_hs_mbuf_to_flat(ctxt->om, &threshold, sizeof(threshold),
&om_len);
        LOG_INFO("[WRITE] new value of threshold: %.2f\n", threshold);
        //predict_interval_ms[om_len] = '\0';
    } else if (ble_uuid_cmp(ctxt->chr->uuid,
&gatt_svr_chr_frequency_uuid.u) == 0) {
        // 写入新的采集频率
        rc = ble_hs_mbuf_to_flat(ctxt->om, &collect_interval_ms,
        sizeof(collect_interval_ms), &om_len);
        LOG_INFO("[WRITE] new value of collect_interval_ms: %d\n",
collect_interval_ms);
        //collect_interval_ms[om_len] = '\0';
    } else if (ble_uuid_cmp(ctxt->chr->uuid, &gatt_svr_chr_led_uuid.u) ==
0) {
        // 写led灯常亮
        rc = ble_hs_mbuf_to_flat(ctxt->om, &led_always,
        sizeof(led_always), &om_len);
        if(led_always != 5){

```

```

        msg_t msg;
        msg.content.value = led_always;
        msg_send(&msg, _led_pid);
    }else{
        LOG_INFO("[WRITE] led continuing prediction: \n");
    }
}
break;
default:
    rc = 1;
    return BLE_ATT_ERR_UNLIKELY;
}

(void) ctxt;
return 1;
}

int main(int argc, char* argv[])
{
    (void)argc;
    (void)argv;
    setup();
    // create led thread
    _led_pid = thread_create(stack_for_led_thread, sizeof(stack_for_led_thread),
    THREAD_PRIORITY_MAIN - 2,
                            THREAD_CREATE_STACKTEST, _led_thread, NULL,
                            "led_controller_thread");
    if (_led_pid <= KERNEL_PID_UNDEF) {
        printf("[MAIN] Creation of receiver thread failed\n");
        return 1;
    }
    else
    {
        printf("[MAIN] LED_PID: %d\n", _led_pid);
    }
    // create motion thread
    _motion_pid = thread_create(stack_for_motion_thread,
    sizeof(stack_for_motion_thread), THREAD_PRIORITY_MAIN - 2,
                            THREAD_CREATE_STACKTEST, _motion_thread, NULL,
                            "motion_predict_thread");
    if (_motion_pid <= KERNEL_PID_UNDEF) {
        printf("[MAIN] Creation of receiver thread failed\n");
        return 1;
    }
    else
    {
        printf("[MAIN] MOTION_PID: %d\n", _motion_pid);
    }

    int rc = 0;
    (void)rc;

    /* verify and add our custom services */
    rc = ble_gatts_count_cfg(gatt_svr_svcs);
}

```

```

assert(rc == 0);
rc = ble_gatts_add_svcs(gatt_svr_svcs);
assert(rc == 0);

/* set the device name */
ble_svc_gap_device_name_set(CONFIG_NIMBLE_AUTOADV_DEVICE_NAME);
/* reload the GATT server to link our added services */
ble_gatts_start();

// 获取蓝牙设备的默认 MAC 地址
uint8_t own_addr_type;
uint8_t own_addr[6];
ble_hs_id_infer_auto(0, &own_addr_type);
ble_hs_id_copy_addr(own_addr_type, own_addr, NULL);

// 打印 MAC 地址
LOG_INFO("Default MAC address: %02x:%02x:%02x:%02x:%02x:%02x\n",
          own_addr[5], own_addr[4], own_addr[3],
          own_addr[2], own_addr[1], own_addr[0]);
/* start to advertise this node */
nimble_autoadv_start(NULL);

return 0;
}

```

train.py

```

# 这里选择 mlp 模型
import glob
import numpy as np
import time
from sklearn.model_selection import train_test_split
DATA_PATH = "../../10_tingml_datasets/"
LABELS = ["Stationary", "Tilted", "Rotating", "Moving"]
SAMPLES_PER_GESTURE = 10
def load_one_label_data(label):
    path = DATA_PATH + label + "*.npy"
    files = glob.glob(path)
    datas = []
    for file in files:
        try:
            data = np.load(file)
            num_slice = len(data) // SAMPLES_PER_GESTURE
            datas.append(data[: num_slice * SAMPLES_PER_GESTURE, :])
        except Exception as e:
            print(e)
    datas = np.concatenate(datas, axis=0)
    datas = np.reshape(datas, (-1, 6 * SAMPLES_PER_GESTURE, ), )
    idx = LABELS.index(label)
    labels = np.ones(datas.shape[0]) * idx
    return datas, labels
all_datas = []
all_labels = []
for label in LABELS:

```

```
datas, labels = load_one_label_data(label)
all_datas.append(datas)
all_labels.append(labels)
dataX = np.concatenate(all_datas, axis=0)
dataY = np.concatenate(all_labels, axis=0)
xTrain, xTest, yTrain, yTest = train_test_split(
    dataX, dataY, test_size=0.2, stratify=dataY
)
print(xTrain.shape, xTest.shape, yTrain.shape, yTest.shape)

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
import tensorflow.keras as keras

# 设置环境变量，控制日志级别
def mlp():
    model = keras.Sequential()
    model.add(keras.layers.Dense(64, activation="relu", input_shape=(6 *
SAMPLES_PER_GESTURE,)))
    model.add(keras.layers.Dropout(0.2))
    model.add(keras.layers.Dense(len(LABELS), activation="softmax"))
    return model

model = mlp()
# 打印模型结构
model.summary()

from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix
optimizer = keras.optimizers.Adam(lr=0.001)
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=optimizer,
    metrics=["sparse_categorical_accuracy"],
)
filepath = "best_model.h5"
checkpoint = ModelCheckpoint(
    filepath,
    monitor="val_sparse_categorical_accuracy",
    verbose=1,
    save_best_only=True,
    mode="max",
)
start_time = time.time()
history = model.fit(
    xTrain,
    yTrain,
    batch_size=8,
    validation_data=(xTest, yTest),
    epochs=50,
    verbose=1,
    callbacks=[checkpoint],
)
# 至此模型训练完毕
```

```

end_time = time.time()
print("Training time for model: {:.2f} seconds".format(end_time - start_time))
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
test_loss, test_accuracy = model.evaluate(xTest, yTest, verbose=0)
print("Test Accuracy for MLP/CNN model: {:.2f}%".format(test_accuracy * 100))

model = load_model(filepath)
predictions = model.predict(xTest)
predictions = np.argmax(predictions, axis=1)
cm = confusion_matrix(yTest, predictions)
print(cm)

model = load_model(filepath)
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open("model_basic.tflite", "wb").write(tflite_model)

data_test = xTest.astype("float32")
data_test = np.reshape(data_test, (-1, 6 * SAMPLES_PER_GESTURE))
data_ds = tf.data.Dataset.from_tensor_slices((data_test)).batch(1)
def representative_data_gen():
    for input_value in data_ds.take(100):
        yield [input_value]
converter.representative_dataset = representative_data_gen
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()
open("model.tflite", "wb").write(tflite_model)

# 量化前后对比
basic_model_size = os.path.getsize("model_basic.tflite")
print("Basic model is %d bytes" % basic_model_size)
quantized_model_size = os.path.getsize("model.tflite")
print("Quantized model is %d bytes" % quantized_model_size)
difference = basic_model_size - quantized_model_size
print("Difference is %d bytes" % difference)

model_quantized_reloaded = tf.lite.Interpreter("model.tflite")

model_quantized_reloaded.allocate_tensors()

model_quantized_input = model_quantized_reloaded.get_input_details()[0]["index"]
model_quantized_output = model_quantized_reloaded.get_output_details()[0]
["index"]
model_quantized_predictions = np.empty(xTest.size)
start_time = time.time()
for i in range(20):
    test_data = np.expand_dims(xTest[i], axis=0).astype("float32")
    print(test_data.shape)
    # Invoke the interpreter

```

```
model_quantized_reloaded.set_tensor(model_quantized_input, test_data)
model_quantized_reloaded.invoke()
model_quantized_prediction = model_quantized_reloaded.get_tensor(
    model_quantized_output
)

print("Digit: {} - Prediction:{}".format(yTest[i],
model_quantized_prediction))
print("")
end_time = time.time()
print("Average Inference Time: {:.4f} seconds".format((end_time - start_time) /
20))
basic_model_size = os.path.getsize("model_basic.tflite")
quantized_model_size = os.path.getsize("model.tflite")
print("MLP Basic Model Size: {} bytes".format(basic_model_size))
print("Quantized Model Size: {} bytes".format(quantized_model_size))
```

运行：

- 进入 external_modules/gesture 文件夹执行 python3.8 train.py.
- 返回 21 文件夹 cd ../../, 执行 make BOARD=esp32-wroom-32 flash term