华东师范大学软件学院课程项目报告

课程名称：创客实践	成绩：
姓名：张梓卫	学号：10235101526
指导老师：陈闻杰	日期：2024/8/10
项目名称：基于 LLM 的 *ESP32* 智能设备多功能控制系统	班级：软工 3 班

目录

一 项目简介 **2**

1 项目介绍 . . . 2
2 项目主要功能及应用场景 . . . 2
3 硬件物料 . . . 2
4 To Do List . . . 3
5 项目环境及工具列表 . . . 3

二 技术原理 **3**

1 ESP32 的使用及其硬件接口 . . . 3
2 基于 LLM、TTS 语音识别与合成 . . . 3
3 FreeRTOS、Ticker 多任务调度 . . . 3
4 基于 HTTP、ESP-NOW 通信协议的使用 . . . 4
5 基于 MQTT 物联网通信协议的使用 . . . 4
6 多种传感器综合实践 . . . 4
6 .1 MPU6050 六轴传感器 . . . 4
6 .2 DHT22 湿温度传感器 . . . 4
6 .3 LM393 光照传感器 . . . 4
6 .4 MQ 系列气体传感器 . . . 4
6 .5 火焰传感器 . . . 4

三 实现方法及步骤 **4**

1 基于 LLM 的语音识别与 TTS 语音合成 . . . 4
1 .1 通过 HTTP 请求返回 RFC 1123 格式的时间戳 . . . 5
1 .2 通过 Hmac-sha256 算法生成鉴权参数 . . . 6
1 .3 调用 LLM API 接口进行联网对话 . . . 7
2 基于 MQTT 协议的物联网搭建 . . . 8
2 .1 构建 OneNET 平台鉴权 token . . . 8
2 .2 连接到 MQTT 服务器 . . . 10
2 .3 与 MQTT 服务器进行数据交互 . . . 10
3 微信小程序连接 MQTT 服务器 . . . 13
3 .1 微信小程序连接物联网 . . . 13
3 .2 构建小程序 UI . . . 13
4 基于 ESP-NOW 通信协议的使用 . . . 15
4 .1 获取主板唯一 MAC 地址 . . . 15
4 .2 扫描 I2C 设备地址 . . . 16

4.3 主机（Master）、从控（Slave）代码 . . . 16
4.4 数据传输的关键步骤 . . . 17
4.5 ESP 与 Wifi 建立通信 . . . 17
4.6 发送 HTTP 请求 . . . 18
5 多种传感器综合实践 . . . 19
5.1 DHT22 湿温度传感器 . . . 19
5.2 基于卡尔曼滤波的 MPU6050 姿态解算 . . . 19
5.3 MQ 系列气体传感器 . . . 20
5.4 雨滴传感器 . . . 21
5.5 LM393 光照传感器 . . . 21
5.6 火焰传感器 . . . 21

四 项目心得总结 21
1 编程技巧 . . . 22
1.1 宏定义与头文件规范 . . . 22
1.2 宏定义与构造函数接口的冲突 . . . 22
1.3 使用”F” 宏处理字符串 . . . 22
1.4 PlatformIO Libraries 处理方案 . . . 22
1.5 ESP32 崩溃后使用 Backtrace 追踪错误代码 . . . 22
2 个人体验 . . . 23
3 未来设想 . . . 23

五 附录 24
1 项目布局及构建 . . . 24
2 参考资料 . . . 24

一 项目简介

1 项目介绍

本项目以 ESP32 为主控，使用 LLM（讯飞星火大模型）运行语音识别与在线联网回答。基于 MQTT、HTTP、ESP-NOW 协议实现远程云端传输、近程数据传输。通过外接传感器实现动态数据读取，在线数据反馈，实现了基于物联网的智能外接设备控制系统。

课程项目源代码及相关文件链接：https://github.com/Shichien/Maker-practice

2 项目主要功能及应用场景

本项目通过多种传感器测量室内示数：如 CH_4 H_2 CH_3COOH LPG 等，实时通过 MQTT 协议传输至云端，在各地均可通过微信小程序或 OneNET 平台调试器进行远程读数，获取室内情况，以及远程控制室内各模块。项目同时实现全智能无线连接，通过语音识别判断用户需求，实时语音播报室内信息；连接大模型，模块可联网实现提问与回答。

通过 ESP-NOW 协议绕过 WiFi，实现伪 AC + AP 组网模式，多接入点可实现任意规模大小接入，通过 ESP-NOW 协议实现短距离无线通信，可以连接任意接口进行函数调用输出。

火焰传感器与光照传感器、雨滴传感器可实时监控室内的特殊天气、事件情况，传输给云端进行数据处理，报警等操作。

3 硬件物料

- ESP-WROOM-32 开发板、ESP-8266-Mod 开发板
- 1.8 寸 RGB-TFT OLED、SSD 1306 OLED
- INMP441 MEMS 麦克风
- MAX98357 I2S 音频放大器

- LB 喇叭
- DHT22、MPU6050、LM393、MQ5、MQ135 传感器
- MH-RD Raindrops Module
- 无刷电机（Fan Module）

具体的接线图及引脚定义图请见【附件】

4 To Do List

- 上传时删除库文件中的 Sample 例子
- 使用 Ollama 部署本地推断模型与 ESP32 交汇
- 基于阿里云进行 MQTT 协议通信以及远程控制、点灯等操作
- 在 PlatformIO 平台上有一个库是 Aliyun IOT SDK，注意后续可能要用到
- 使用 MPU6050 进行云台自控制

5 项目环境及工具列表

本项目设计的环境为 Windows 11，开发板编程使用 Clion Platform，选择 Upesy_Wroom 开发环境，Arduino 框架，模拟仿真平台使用 Wokwi Simulator，串口监视器选用 VOFA+ 1.3.10，波特率为 115200（8N1），代码使用 Vim 编写，课程报告使用 LaTeX 撰写，并使用 Git 进行版本管理。

二 技术原理

1 ESP32 的使用及其硬件接口

- Espressif 32ESP32 集成了 2.4 GHz Wi-Fi 和 Bluetooth 4.2 LE（低功耗）功能，使其能够在无线网络环境中轻松实现连接和数据传输。
- 提供了丰富的外设接口，包括多达 34 个 GPIO 引脚、ADC（模数转换器）、DAC（数模转换器）、SPI、I2C、UART、PWM、CAN 总线。

2 基于 LLM、TTS 语音识别与合成

随着 LLM 在近年来在世界各地火热传播，一位机器学习工程师与我说：“现在市场上是个软件就得说自己用了 AI，不然不好意思提供服务了”，此句稍带滑稽的话语让我萌生了使用 LLM 的念头。灵活高效、易于使用。在本项目中，主要用于流式语音识别及语音生成转换（基于讯飞的星火大模型 Spark Ultra 4.0）。

3 FreeRTOS、Ticker 多任务调度

FreeRTOS 是一个广泛用于嵌入式开发的实时操作系统，提供多任务调度、任务间通信、优先级管理等功能，vTaskDelay() 是 FreeRTOS 提供的任务延迟函数，用于让当前任务进入阻塞状态，释放 CPU 给其他任务使用。ESP32 已经包含了 FreeROTS 内核，使用 vTaskDelay，可以避免 Delay() 函数造成的整体阻塞。

同时，Ticker 计时器可以使用一行代码实现多任务调度的管理：`ticker.attach(Time,FunctionName);`

4 基于 HTTP、ESP-NOW 通信协议的使用

ESP-NOW 是一种轻量级、低功耗的点对点（P2P）通信协议，专为短距离设备之间的快速数据传输而设计。不依赖于传统的 Wi-Fi 网络和路由器。ESP-NOW 允许多个设备（资料显示，最多 20 个，不然会超过 ESP32 的功耗）直接进行无线数据传输，适用于需要快速和实时通信的物联网应用场景。

HTTP 在客户端与服务器之间传输超文本数据，是互联网通信的基础，HTTPClient 库提供各种接口，包括 GET、POST 请求，可以连接服务器后返回响应：HTML、JSON 数据等，使用 ESP32 可以远程连接各种 API 获取数据，如使用www.baidu.com 获取 RFC1123 时间戳。

5 基于 MQTT 物联网通信协议的使用

MQTT 是一个轻量级的发布/订阅消息传输协议，广泛用于物联网设备之间的数据通信。提供在不稳定网络环境中的低带宽、高延迟或间歇性连接情况下的可靠通信。客户端可以是任何设备或应用程序（如传感器、服务器、移动设备），它们可以作为发布者，也可以作为订阅者接收消息。

OneNET 是中国移动开发的物联网（IoT）开放平台，支持多种主流通信协议，如 MQTT、HTTP、CoAP 等，在此我使用 MQTT 协议进行传感器的上传。

6 多种传感器综合实践

Adafruit 开源硬件公司各常用模块以及 MH-Sensor 系列是一系列传感器产品，包括温度传感器、湿度传感器、气压传感器光照传感器等。这些传感器可以用于测量环境参数，例如温度、湿度、气压、光照等。

6 .1 MPU6050 六轴传感器

MPU6050 集成了三轴加速度计和三轴陀螺仪，能够测量 X、Y、Z 三个方向上的加速度和角速度。加速度计的测量范围为 ±2g、±4g、±8g 和 ±16g。陀螺仪的测量范围为 ±250°/s、±500°/s、±1000°/s 和 ±2000°/s。采用 IIC 总线接口通信。

6 .2 DHT22 湿温度传感器

温度测量范围: -40°C 到 80°C，精度为 ±0.5°C。湿度测量范围: 0% 到 100% 相对湿度，精度为 ±2% RH。

DHT22 使用单线数字信号输出，易于与微控制器进行接口通信。

6 .3 LM393 光照传感器

LM393 的工作原理基于电压比较。当正输入电压高于负输入电压时，输出为低电平（接近 0V）。当正输入电压低于负输入电压时，输出为高电平（受上拉电阻决定），内置光敏电阻，可提供模拟信号。

6 .4 MQ 系列气体传感器

MQ 传感器内部通常由一个加热元件和一个氧化锡（SnO2）半导体组成。当特定气体通过传感器时，氧化锡半导体的电导率会发生变化，这种变化可以被检测并转化为气体浓度。每种 MQ 传感器的灵敏度曲线不同，对某些气体更敏感。例如，MQ-2 对甲烷、丙烷和烟雾敏感，而 MQ-7 对一氧化碳更敏感。

注意：**MQ 系列传感器一次只能检测一种气体的浓度值，但对于不同的气体分子，拥有不同的指数函数曲线，在定义不同的 A、B 值时读取**

6 .5 火焰传感器

火焰传感器可以用来探测火源或其它波长在 760 纳米～1100 纳米范围内的光源。探测角度达 60 度，对火焰光谱特别灵敏。注意，手机手电筒中也含有对应波长的光源。

三 实现方法及步骤

1 基于 LLM 的语音识别与 TTS 语音合成

要想在数字音频设备之间进行通信，可以使用 I2S 同步串行通信协议。

在 ESP32 中，有实现 I2S 协议的库：https://github.com/schreibfaul1/ESP32-audioI2S

首先要进行语音识别的实现，我选择使用 INMP441 MEMS 全向麦克风进行实时录音。接下来聚焦的痛点是：如何将录音数据传输到 LLM 中进行语音识别？

幸运的是，讯飞开放平台中提供了流式语音识别的 API，可以将录音数据实时传输到 LLM 中进行语音识别。**以下，我们将语音转文字（Speech To Text）简写为 STT，文字转语音（Text To Speech）简写为 TTS**。下一步即调用 API 接口，进入官方文档中获取相关的参数。

图 1: STT API 接口参数列表

在调用接口时，需要按照下方所示的 Url 鉴权方法进行请求地址加鉴权参数的形式鉴权。

图 2: Url 鉴权方法

以下分别按照步骤进行鉴权：

1 .1 通过 HTTP 请求返回 RFC 1123 格式的时间戳

```
void getRFC1123Time() {
    HTTPClient http;
    http.begin("https://www.baidu.com");

    // 从 HTTP 响应头中获取Date
    const char *headerKeys[] = {"Date"};
    http.collectHeaders(headerKeys, sizeof(headerKeys) / sizeof(headerKeys[0]));
    int httpCode = http.GET() ; // 必须要先设置收集字段，再发送HTTP请求，否则收集不到
    Date = http.header("Date");
    http.end();

    // Debug
```

```
    Serial.println("Date:" + Date);
    Serial.println("httpCode:" + httpCode);
}
```

通过 HTTP 请求返回 RFC1123 格式的时间戳

1 .2 通过 Hmac-sha256 算法生成鉴权参数

Hmac-sha256 算法介绍部分见附件。此处我使用 mbedTLS 开源加密库实现算法。

```
String getOriginalSignature(String host, String date, String path) {
    return "host: " + host + "\n" + "date: " + date + "\n" + "GET " + path + " HTTP/1.1";
}

String getUrl(String Spark_url, String host, String path, String date) {
    String originalSignature = getOriginalSignature(host, date, path);
    const size_t originalSignatureLength = originalSignature.length(); // 原始签名的长度
    const size_t APIKeyLength = APISecret.length(); // API Key 的长度
    // 使用 HMAC-SHA256 进行加密
    unsigned char hmac[32]; // 存储HMAC结果
    mbedtls_md_context_t ctx; // HMAC上下文
    mbedtls_md_type_t md_type = MBEDTLS_MD_SHA256;
    mbedtls_md_init(&ctx);
    mbedtls_md_setup(&ctx, mbedtls_md_info_from_type(md_type), 1);
    mbedtls_md_hmac_starts(&ctx, (const unsigned char *) APISecret.c_str(), APIKeyLength);
    mbedtls_md_hmac_update(&ctx, (const unsigned char *) originalSignature.c_str(), originalSignatureLength);
    mbedtls_md_hmac_finish(&ctx, hmac);
    mbedtls_md_free(&ctx);

    // 将HMAC结果进行Base64编码
    String signature_sha_base64 = base64::encode(hmac, sizeof(hmac) / sizeof(hmac[0]));

    // 替换Date字符串中的特殊字符
    date.replace(",", "%2C");
    date.replace(" ", "+");
    date.replace(":", "%3A");

    // 构建Authorization原始字符串
    String authorization_origin = "api_key=\"" + APIKey + "\", algorithm=\"hmac-sha256\", headers=\"host date
    request-line\", signature=\"" + signature_sha_base64 + "\"";

    // 将Authorization原始字符串进行Base64编码
    String authorization = base64::encode(authorization_origin);

    // 构建最终的URL
    String FinalUrl = Spark_url + '?' + "authorization=" + authorization + "&date=" + date + "&host=" + host;
    Serial.println(FinalUrl); // Debug
    return FinalUrl;
}
```

通过 Hmac-sha256 算法生成鉴权参数

鉴权完毕，在串口调试助手中返回了正确的鉴权 Url：

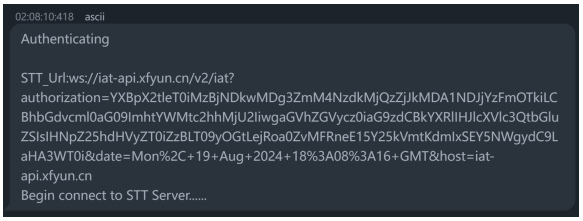


图 3: 鉴权 Url

1 .3 调用 LLM API 接口进行联网对话

官方文档中所示示例如下，使用 DynamicJson 创建动态 Json 对象，按顺序构造 Json 即可。

图 4: LLM API 参数构造

```
DynamicJsonDocument getParameters(const char *appid, const char *domain, const char *role_set) {
    DynamicJsonDocument data(1500);
    // 构建Json的嵌套对象
    JsonObject header = data.createNestedObject("header");
    JsonObject parameter = data.createNestedObject("parameter");
    JsonObject chat = parameter.createNestedObject("chat");
    JsonObject payload = data.createNestedObject("payload");
    JsonObject message = payload.createNestedObject("message");
    JsonArray textArray = message.createNestedArray("text");
    JsonObject systemMessage = textArray.createNestedObject();
    // 将某些对象赋予值
    header["app_id"] = appid;
    header["uid"] = "1234";
    chat["domain"] = domain;
    chat["temperature"] = 0.6;
    chat["max_tokens"] = 1024;
    systemMessage["role"] = "system";
    systemMessage["content"] = role_set;

    return data;
}
```

调用 LLM API 接口进行联网对话

注意到在 *payload* → *message* → *text* 中，我们可以向 LLM 发送多条文本，以实现多轮对话。故可以创建一个动态数组存储多轮对话的结果，在此可以使用 <Vector>，于是有：

```
    // 反序列化：将历史对话 Strings 返回到一个 Json 对象
    for (const auto &jsonStr: historicalDialogue) {
        DynamicJsonDocument tempDoc(512);
        DeserializationError error = deserializeJson(tempDoc, jsonStr);
        if (!error) textArray.add(tempDoc.as<JsonVariant>());
    }
    return data;
```

多轮对话实现

限于代码长度限制，下面给出后续流程的脑图。

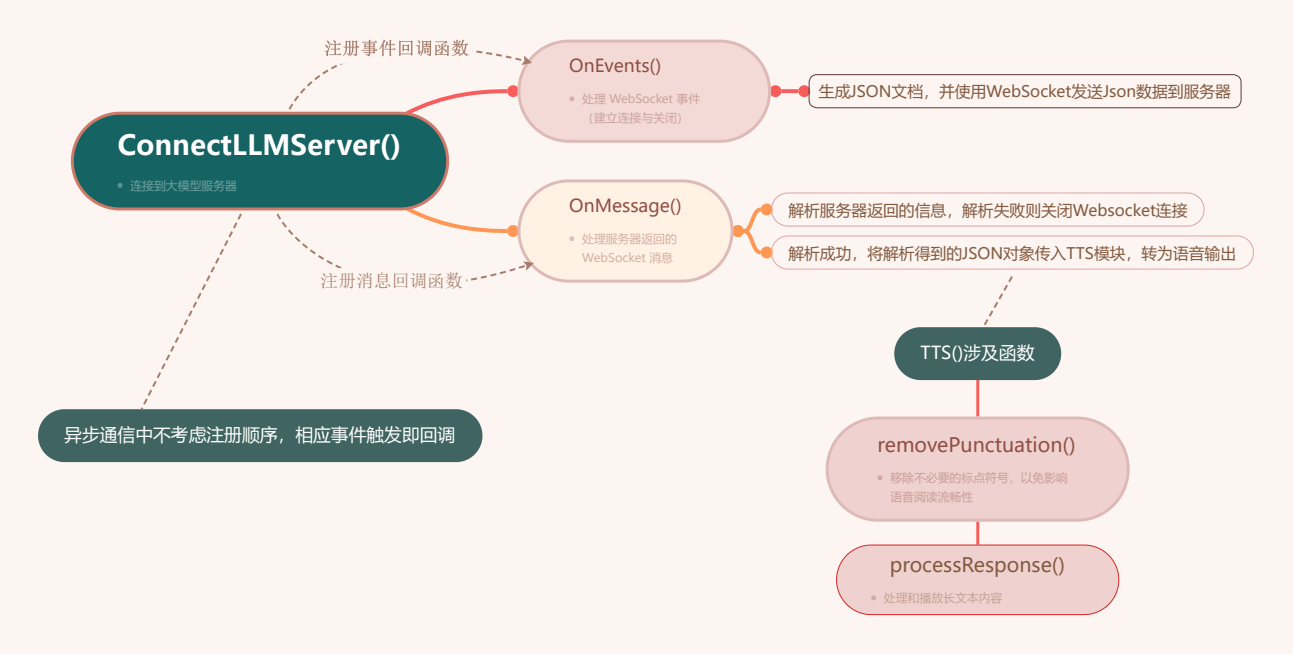


图 5: LLM 与 TTS 交互的流程及实现

2 基于 MQTT 协议的物联网搭建

中国移动的 OneNET 平台专门为物联网应用开发，我们可以使用 ESP32 连接到 MQTT 服务器，之后再构建手机 APP 来获取数据，真正实现信息互联。

2 .1 构建 OneNET 平台鉴权 token

OneNET 平台：https://open.iot.10086.cn/ 我使用的是 OneJSON 数据结构，MQTT 协议上传。

将一些基础信息置于此地，后续需要时在这里查看：（鉴权 Token 在后节获得）

- 设备密钥：VzZrREE0T2hYeDZ0dDFQSUZZWWRIZ29yMGxrRUdGY2g=
- 设备名称：DHT22
- 产品 ID：W9TI0JaXlu
- 产品 Access：8fnx/QsBBf/BUeWpU68j4bG7iwNVm315g+vzc8c
- 鉴权 Token:version=2018-10-31&res=products%2FW9TI0JaXlu%2Fdevices%2FDHT22&et=1759306118&method=md5
- &sign=JRsZv0peZQOTmr7Eg8lGJg%3D%3D

在 PlatformIO 中使用 PubSubClient 库，可以用于在 Arduino 框架下实现 MQTT 通信。下面，通过 OneNET 平台的官方文档找到了接口：https://open.iot.10086.cn/doc/v5/fuse/detail/919

图 6: OneNET 平台接口

按照文档：产品、设备创建时，平台为每类产品、每个设备均分配了唯一的 key，设备登录时需要使用通过 key 计算出的访问 token 来进行访问安全认证。官方提供了 token 计算工具，按照 res 使用场景根据步骤获得 token。

图 7: OneNET 平台 token 计算

在调用讯飞 API 时我使用的就是 Sha-256 算法，根据官方文档，支持 md5、Sha-1、Sha-256，故我使用 Sha-256 作为尝试。注意，官方文档中提到：

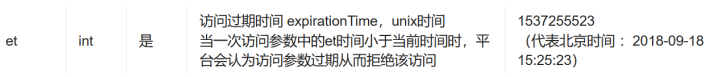


图 8: et 代表过期时间

故使用 Unix 时间戳转换工具，将到期时间设置为 2025 年 10 月 1 日。

Unix 时间戳转换网址为：https://www.jyshare.com/front-end/852/ ，转换得到 1759306118。

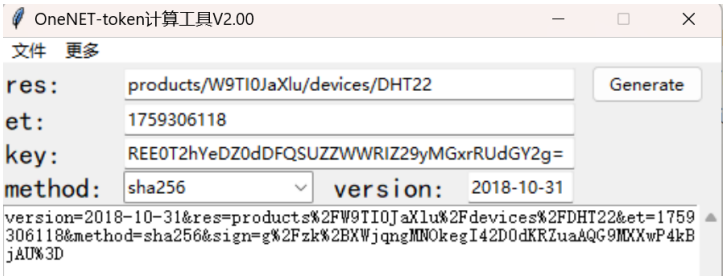


图 9: 计算工具

2 .2 连接到 MQTT 服务器

连接到 MQTT 服务器的关键部分代码如下所示：

```
WiFiClient espClient;
PubSubClient client(espClient);

void connectOneNet() {
    client.setServer(MQTT_Server.c_str(), MQTT_Port);
    bool isConnected = client.connect(MQTT_Device_ID.c_str(), MQTT_Product_ID.c_str(), MQTT_Token.c_str());
}

void loop() {
    client.loop();
}
```

MQTT Connect

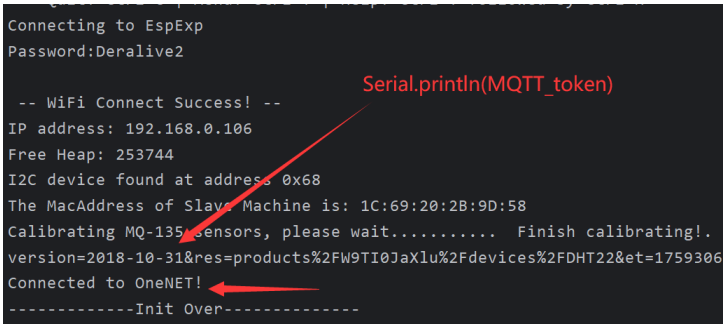
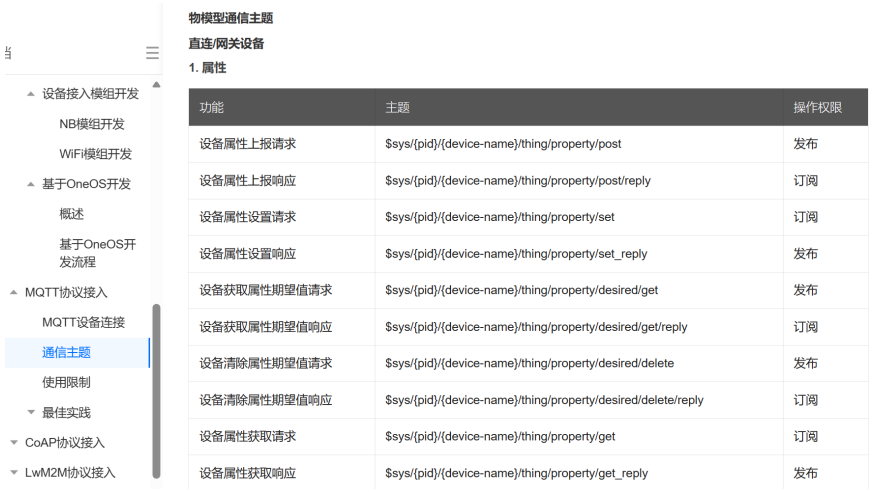


图 10: MQTT 服务器连接成功

2 .3 与 MQTT 服务器进行数据交互

连接到服务器后，数据的上传和下载自然成为接下来要攻克的主题。这两部分分别从属于“发布”与“订阅”的操作权限。



功能	主题	操作权限
设备属性上报请求	$sys/{pid}/{device-name}/thing/property/post	发布
设备属性上报响应	$sys/{pid}/{device-name}/thing/property/post/reply	订阅
设备属性设置请求	$sys/{pid}/{device-name}/thing/property/set	订阅
设备属性设置响应	$sys/{pid}/{device-name}/thing/property/set_reply	发布
设备获取属性期望值请求	$sys/{pid}/{device-name}/thing/property/desired/get	发布
设备获取属性期望值响应	$sys/{pid}/{device-name}/thing/property/desired/get/reply	订阅
设备清除属性期望值请求	$sys/{pid}/{device-name}/thing/property/desired/delete	发布
设备清除属性期望值响应	$sys/{pid}/{device-name}/thing/property/desired/delete/reply	订阅
设备属性获取请求	$sys/{pid}/{device-name}/thing/property/get	订阅
设备属性获取响应	$sys/{pid}/{device-name}/thing/property/get_reply	发布

图 11: MQTT 发布

往下翻，需要向服务器传输 JSON 格式的字符串，如下图所示：

图 12: MQTT Send Data With JSON Format

关键函数如下所示，注意要看物模型通信主题中哪一个是订阅，哪一个是发布。使用对应的函数 `client.subscribe()` 或 `client.publish()`，同时，设置回调函数 mqttCallback()，用于处理服务器返回的数据。回调函数解析服务器返回的数据，若为 200 则代表数据解析成功。

此处的每一个函数返回的都是 boolean 类型（可以通过点入函数声明的部分，即在 PubSubClient.cpp 内查阅），使用返回值来做对应的串口输出，用于提示我是否完成了此次请求。关键代码如下所示：

```
void connectOneNet() {
    client.setServer(MQTT_Server.c_str(), MQTT_Port);
    client.setCallback(mqttCallback); // 回调函数解析服务器返回的数据，若为 200 则代表数据解析成功

    bool isConnected = client.connect(MQTT_Device_ID.c_str(), MQTT_Product_ID.c_str(), MQTT_Token.c_str());
    bool isSubscribeProperty = client.subscribe(ONENET_TOPIC_PROP_SET); //订阅设备属性设置请求
    bool isSubscribeUpdata = client.subscribe(ONENET_TOPIC_PROP_POST_REPLY); //订阅设备属性上报响应

    if (isConnected) {
        Serial.println(F("Connected to OneNET!"));
    } else {
        Serial.println(F("Failed to connect OneNeT!"));
    }
    // 更多 if，结构一致，为节省篇幅，省略。

    ticker.attach(3,OneNET_Prop_Post); // 使用 Ticker 对象，每隔 3 秒发送一次设备属性上报。
}

void OneNET_Prop_Post() {
    if (client.connected()) {
```

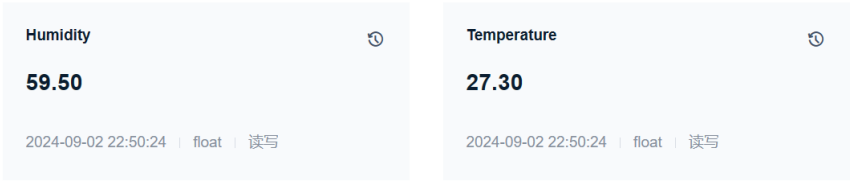
```
            char parameters[256];
            char jsonBuff[256];

            // 构建用来传输至 MQTT 服务器的 JSON 数据
            sprintf(parameters, "{\"temperature\":{\"value\":%.2f},\"humidity\":{\"value\":%.2f}}",dht22Data.
    temperature,dht22Data.humidity);
            sprintf(jsonBuff,ONENET_TOPIC_PROP_FORMAT,postMessageID++,parameters);
            // 按照文档，#define ONENET_TOPIC_PROP_FORMAT "{\"id\":\"%u\",\"version\":\"1.0\",\"params\":%s}"

            bool isUpData = client.publish(ONENET_TOPIC_PROP_POST, jsonBuff);
        }
    }

    void mqttCallback(char* topic, byte* payload, unsigned int length) {
        // 将接收到的消息转换为字符串
        char message[length + 1];
        strncpy(message, (char*)payload, length);
        message[length] = '\0';

        // 打印主题和消息内容
        Serial.print("Received message from topic: ");
        Serial.println(topic);
        Serial.print("Message: ");
        Serial.println(message);
    }
```

MQTT Publish and Subscribe

将信息发送至 MQTT 服务器及服务器接收得到的结果如图所示：

```
{"temperature":{"value":27.30},"humidity":{"value":59.10}}
{"id":"140","version":"1.0","params":{"temperature":{"value":27.30},"humidity":{"value":59.10}}}
Send Data To MQTT Server Successfully!
Received message from topic: $sys/W9TI0JaXlu/DHT22/thing/property/post/reply
Message: {"id":"140","code":200,"msg":"success"}
```

(a) 将信息发送至 MQTT 服务器



(b) OneNET 服务器接收到数据

图 13: MQTT 发送与接收

接下来要做的事情和上面的重复，把其他传感器和想要的数据传输到上面即可。

图 14: MQTT 服务器接收到的数据

3 微信小程序连接 MQTT 服务器

3 .1 微信小程序连接物联网

随着安卓平台逐渐没落，多端开发应运而生，此时以 Javascript、WXML 布局的小程序便可以多端联合开发，而不用担心兼容性问题。在此，我使用微信开发者工具 Stable 1.06.2407120 进行微信小程序开发，通过远程连接 OneNET 平台，随时随地，实时显示数据于手机当中。

3 .2 构建小程序 UI

基于 Nodejs、Javascript 语言，使用 flex 弹性布局，可以使得框架的构建更为简单。此处的 UI 图标均采用互联网免费公开资料。

```
<!-- Title Header -->
<view class = "container">
    <view class = "head_box">
        <image src = "/images/OneNET.jpg" mode =""/>
    <view>{{title}}</view>
      </view>
<!-- Weather Module -->
      <view class = "weather_box">
        <view class = "welcome_text">
              {{welcome}}
        </view>
          <view class = "flex">
              <view class = "width50">
                <image src="/images/Weather.jpg" style="width: 200rpx; margin-top: 30rpx; margin-left: 30rpx
;" mode="widthFix"/>
              </view>
              <view>
                <view class= "location_text">
                    <image src="/images/located.jpg" style ="width: 20rpx; margin-top: 10rpx;" mode="
widthFix"/><text
                     style="font-size: 24rpx;">{{location}}</text>
                </view>
                <view>
                    <view class = "temperature_text">
                        {{temperature}}°C
                    </view>
                </view>
              </view>
          </view>
      </view>
<!-- Device Module -->
```

Listing 1: 微信小程序 UI

再根据实际需要配置实现情况 index.wxss, index,js 文件，初步实现框架如下所示：

图 15: 微信小程序 UI

根据 OneNET 平台提供的 API，我们需要使用 HTTP 的 GET 请求连接到 OneNET 服务器，这里没办法使用 Websocket 协议（2024-02 更新），

图 16: OneNET GET 请求示例

按照示例模板，使用 API-KEY，根据之前生成的鉴权参数（见图 9），生成签名，并将其加入到请求头中，即可完成 HTTP 请求。

```
getInfo() {
let that = this;
wx.request({
  url: "https://iot-api.heclouds.com/thingmodel/query-device-property?product_id=W9TI0JaXlu&device_name=DHT22",
  header: {
    "authorization": "version=2018-10-31&res=products%2FW9TI0JaXlu%2Fdevices%2FDHT22&et=1759306118&method=md5&sign=JRsZv0peZQOTmr7Eg8lGJg%3D%3D"
  },
  method: "GET",
  success: res => {
    console.log("获取成功", res);
      },
});
},
```

Listing 2: HTTP GET 请求示例

获取 JSON 数据格式如下所示：

图 17: OneNET JSON 数据格式

根据 data.Array 中的参数，每个数组返回的值代表不同的参数，后续只需读取逻辑即可。

```
const sensorList = that.data.sensorList;
sensorList[0].value = res.data.data[4].value;
sensorList[1].value = res.data.data[1].value;
sensorList[2].value = res.data.data[0].value;
this.setData({
```

```
        sensorList: sensorList
    });
```

Listing 3: 读取 JSON 数据

再使用 WXML 布局，将数据显示在 UI 上，即可完成微信小程序的连接物联网功能。

```
  <!-- Sensor UI -->
  <view class="sensors-system-title">
    传感器设备信息
  </view>
  <view class="sensors-system">
    <view wx:for="{{sensorList}}" class="system-info">
      <view class="sensors-system-box1">
        <image src="{{item.img}}" style="height: 80rpx;" mode="heightFix"/>
      </view>
      <view class="sensors-system-box2">
        <view>{{item.parameter}}</view>
        <view>{{item.value}}{{item.unit}}</view>
        <view>{{item.name}}</view>
      </view>
    </view>
  </view>
</view>
```

Listing 4: WXML 布局

图 18: 微信小程序完整 UI

4 基于 ESP-NOW 通信协议的使用

在实现各种功能的过程中，总会发现一块开发板的引脚不够用的情况，此时我考虑使用多块开发板进行开发。除使用 I2C、SPI、UART 等协议进行开发板之间的通信之外，本身支持 Wifi 的 ESP32 中还有一个由乐鑫公司开发的协议 ESP-NOW，可以无线（即占用更少的引脚）实现两块 ESP 之间的通信。

我使用 ESP32 作为主控，主控中调配语音识别大模型，ESP8266 中作为从控，调用各种传感器，如 DHT11 湿温度模块。

通过 ESP-NOW 协议实现两块 ESP 之间的通信，读取从控中获取的参数。根据官方手册中的 Examples：ESP-NOW.html，我仿照其代码成功实现了通信。

4 .1 获取主板唯一 MAC 地址

```
    void setup() {
        Serial.begin(115200); // 每一块板的MAC地址是唯一的
        WiFi.mode(WIFI_MODE_STA); // 知道MAC地址后就可以向主机发送信息
        Serial.println(WiFi.macAddress());
    }
```

获取主板唯一 MAC 地址

通过上传代码，我将我所需开发板的 MAC 地址记录在此：

- 主板：BC:DD:C2:D0:07:B4
- 从控（32）：1C:69:20:2B:9D:58
- 从控（32E）：EC:64:C9:90:6B:74
- 从控（32 Side）：BC:DD:C2:CD:07:00
- 从控（8266）：待使用

4.2 扫描 I2C 设备地址

需要使用 IIC 协议通信的传感器（如 MPU6050），必须需要设备的 IIC 地址（如 `MPU.begin(0x68,&Wire);`）使用 Wire.h 库中的接口扫描，得到 $0x68$。I2C 接口不止可以插一根，根据不同的通信地址可以在同一个引脚处插入多个 IIC 通信。

```
    Wire.begin();  // 初始化I2C总线
    Serial.begin(115200);

    // 扫描所有可能的I2C地址
    for (uint8_t address = 1; address < 127; address++) {
        Wire.beginTransmission(address);
        uint8_t error = Wire.endTransmission();

        if (error == 0) {
            Serial.print("I2C device found at address 0x");
            if (address < 16) Serial.print("0");
            Serial.print(address, HEX);
        }
    }
```

扫描 I2C 设备地址

4.3 主机（Master）、从控（Slave）代码

主机（Master）代码：

在根据 Examples 的运作时，我的 ESP-NOW 发送数据一直是失败的，查阅资料后发现要加入一行代码：

`peerInfo.ifidx = WIFI_IF_STA;`

暂不理解为什么其他示例没有这行代码也可以运行，参考资料：https://esp32.com/viewtopic.php?p=132946。

```
// 定义接收到的数据结构
typedef struct struct_message {
    float accelX;
    float accelY;
    float accelZ;
    float temperature;
} struct_message;

struct_message incomingData;

void onDataRecv(const uint8_t * mac, const uint8_t *data, int len) {
    // 将接收到的字节数据解释为结构体类型
    struct_message *receivedData = (struct_message *)data;
}

void setup {
    WiFi.mode(WIFI_STA); // 设置为站点模式
    // 设置对等设备信息
    esp_now_peer_info_t peerInfo;
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    peerInfo.ifidx = WIFI_IF_STA; // 注意！此步骤一定要有
```

```
    // 注册接收回调函数
    esp_now_register_recv_cb(onDataRecv);
}
```

主机（Master）代码

从机关键代码，其余部分与主机相似：

```
// 目标主机的MAC地址（需要替换为实际的主机MAC地址）
uint8_t masterMACAddress[] = {0xBC,0xDD,0xC2,0xD0,0x07,0xB4};

// 发送回调函数
void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    Serial.print("Last Packet Send Status: ");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void loop {
    esp_err_t result = esp_now_send(masterMACAddress, (uint8_t *)&myData, sizeof(myData));
}
```

从机（Slave）代码

4.4 数据传输的关键步骤

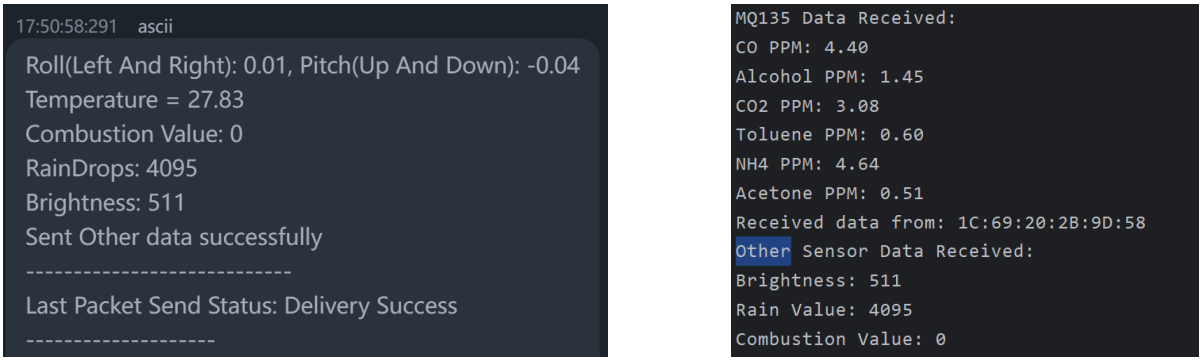
由于在数据传输时，在主机（Master）处只注册了一个回调函数，但从机（Slave）发送的数据有很多种（我定义了不同的结构体来存储变量），考虑到可能出现结构体对齐问题，不妨将各部分结构体多定义出一个变量，使得接收时可以先判断这个结构体的大小以判断接收到的是什么传感器的数据，接下来再进行数据解析。

```
// 8 字节
typedef struct DHT22Message {
    float temperature;
    float humidity;
} DHT22Message;
.......
```

```
// 12 字节
typedef struct MPU6050Message {
    float UpAndDown;
    float LeftAndRight;
    float temperature;
} MPU6050Message;
```

```
// 16 字节
typedef struct MQ5Message {
    float h2PPM;
    float ch4PPM;
    float lpgPPM;
    float flagMQ5;
} MQ5Message;
```

主机读取结构体大小以判断数据来源



(a) 从机发送数据 (b) 主机读取数据

图 19: 基于 ESP-NOW 的数据传输成功

4.5 ESP 与 Wifi 建立通信

首先，进入 TP-Link 路由器的管理界面，室内路由器使用 Mesh 组网，故使用默认配置地址 192.168.0.1。

注意，ESP 系列硬件仅支持 2.4GHz 的无线网络通信，故开启 2.4GHz 的 Wifi。

图 20: TP-Link 路由器管理界面

通过 Wifi.h 库中的接口，进入 WifiSTA.cpp 查看所有接口（见附件-1），使用 Wifi.begin() 函数连接到路由器。

```
Leaving...
Hard resetting via RTS pin...
--- Terminal on COM7 | 9600 8-N-1
--- Available filters and text transformations: colorize, debug, default,
e
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Wifi Connecting.....
Successful Connection
IP Address:192.168.0.108
```

图 21: ESP32 与 Wifi 成功建立连接

4 .6 发送 HTTP 请求

心知天气提供免费的天气 API 查询接口：https://www.seniverse.com/，先用此进行测试，后续可调用更多 API。

API请求方式

每组密钥由“公钥”（参数uid）和“私钥”（参数key）组成，例如：

- 公钥 PKwiV7auWJE3iBJ8d
- 私钥 SMEieQjde1C9eXnbE

心知天气支持两种 API 安全验证方式：

1.“私钥”直接请求方式（更简单）

将 API 密钥中的“私钥”作为 API 请求中的 key 参数值：

比如 https://api.seniverse.com/v3/weather/now.json?key=your_private_key&location=beijing&language=zh-Hans&unit=c

将your_private_key更改为私钥即可

图 22: 心知天气 API

使用 HTTPClient.h 库，向服务器发送 HTTP 请求，并接收服务器的响应。使用代码如下所示：

```
// 创建 HTTPClient 对象并发送 Get 请求
HTTPClient http;
http.begin(url+"?key="+key+"&location="+city+"&language="+language+"&unit="+unit);
int httpCode = http.GET();

// 获取响应状态码及正文
Serial.printf("HTTP Status Code: %d", httpCode);
Serial.print("\n");

String response = http.getString();
Serial.println("Response Data:");
Serial.println(response);

http.end();

// 解析 JSON 数据
```

```
DynamicJsonDocument doc(1024);
deserializeJson(doc, response);

// 从解析后的 JSON 文档中获取值
unsigned int temp = doc["results"][0]["now"]["temperature"].as<unsigned int>();
String info = doc["results"][0]["now"]["text"].as<String>();

Serial.printf("Daytime temperature: %d\n", temp);
Serial.printf("Daytime Weather: %s\n", info);
```

发送 HTTP 请求并解析 JSON 数据

上传并监视，使用 VOFA+ 1.3.10，通信成功，数据解析成功，结果如下图所示：

图 23: 天气数据

5 多种传感器综合实践

5 .1 DHT22 湿温度传感器

与 MPU6050 一致，我们可以构建一个信息结构体，使用 ESP-NOW 协议将其发送给 Master（主机）。

```
typedef struct DHT22Message {
    float temperature;
    float humidity;
} DHT22Message;
```

DHT22 Message

Master 收到信息后，解析其中的数据，并与 MPU6050 的温度进行求平均值处理。注意，DHT22 需要激活信号，在 DHT.h 代码中查阅发现，使用到了 `pinMode(_pin, OUTPUT);` 一瞬间来给 DHT22 发送信号，告诉它需要开始读取并传输信息数据。

成功运行并读取数据的截图（见下图 (图24)）

5 .2 基于卡尔曼滤波的 MPU6050 姿态解算

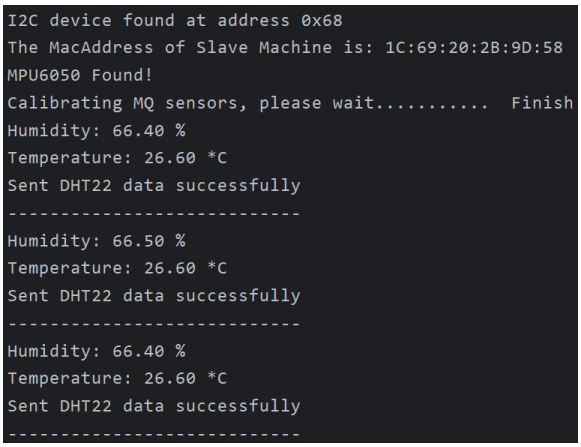
加速度计在静态时解算的姿态的可信度较高，陀螺仪在动态时解算的姿态的可信度较高。所以需要结合两者的结果进行融合。因为陀螺仪解算的姿态需要积分，随着时间的增加，微小的误差会积分出较大的误差，这个误差可以通过加速解解算的 *roll*,*pitch* 减去陀螺仪解算的 *roll*,*pitch*，再乘以一个比例系数代替。

参考资料：MPU6050 姿态解算 2-欧拉角 & 旋转矩阵

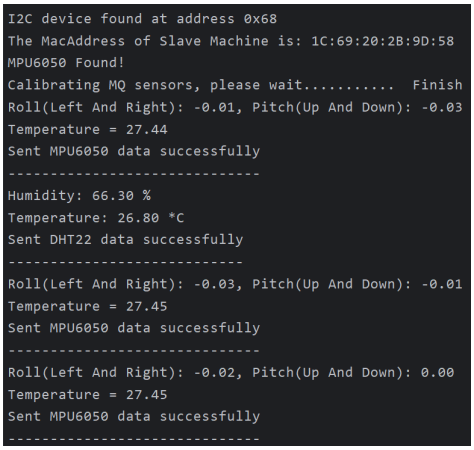
本部分数学知识不在课程学习范围内，故参考了（UP 主：地上的感觉还不错）开源程序：

学习心得 | 基于卡尔曼滤波的 MPU6050 姿态解算

同时注意到 MPU6050 还可以读取环境的温度，故使用该读取温度与 *DHT*22 读取的温度进行综合，求平均值，能够增大环境温度的可信度。



(a) DHT22 运行成功截图



(b) MPU6050 成功读取偏角

图 24: DHT22 和 MPU6050 的运行结果

5 .3 MQ 系列气体传感器

MQUnifiedSensor 库提供了各种库接口。初次使用本库时，将 ADC 引脚定义为 ESP32 中的 GPIO14，报错提示如下所示：

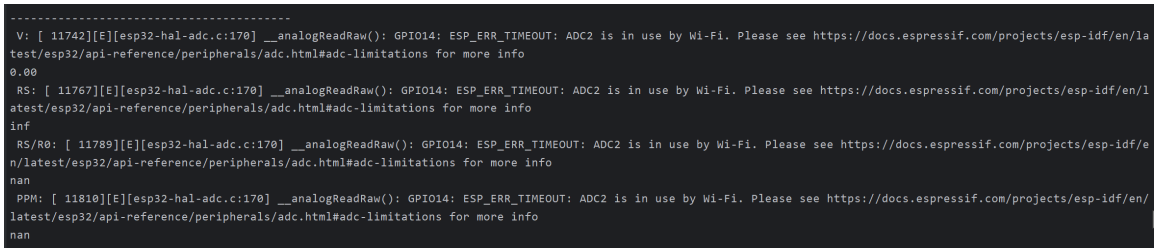


图 25: MQ135 报错

查阅资料后发现，Espressif（ESP32 的制造商）在设计 ESP32 时，必须在多个功能之间做出权衡。例如，Wi-Fi 是一个非常关键的功能，它需要稳定的时间控制和电源管理。因此，当 Wi-Fi 活跃时，它会占用与 ADC2 共享的资源，以确保网络连接的稳定性和性能。

故将 ADC 模块改用为 GPIO32-39 即可（见附件——ESP-Wroom-32 开发板的硬件接口）

MQ-135 是一种空气质量传感器，用于检测空气中的一氧化碳、氮氧化物、酒精、氨气和烟雾等有害气体。工作原理是通过化学反应来检测目标气体的浓度，并将结果转换为电信号输出。

对于需要读取不同的气体，需要设置不同的系数（见附件）

故 loop 代码中应写为：

```
void loopMQ135() {
    MQ135.update();

    // 读取 CO 浓度
    MQ135.setA(605.18);
    MQ135.setB(-3.937);
    float coPPM = MQ135.readSensor();

    // 读取 Alcohol 浓度
    MQ135.setA(77.255);
    MQ135.setB(-3.18);
    float alcoholPPM = MQ135.readSensor();
    // ... 重复即可
}
```

MQ135 不同气体的读取

参考自官方 MQUnifiedsensor/example/MQ-135-ALL.ino，各参数见附件。

```
void loop() {
  MQ135.update(); // Update data, the arduino will read the voltage from the analog pin

  MQ135.setA(605.18); MQ135.setB(-3.937); // Configure the equation to calculate CO concentration val
  float CO = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b value

  MQ135.setA(77.255); MQ135.setB(-3.18); //Configure the equation to calculate Alcohol concentration
  float Alcohol = MQ135.readSensor(); // SSensor will read PPM concentration using the model, a and b

  MQ135.setA(110.47); MQ135.setB(-2.862); // Configure the equation to calculate CO2 concentration va
  float CO2 = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b valu

  MQ135.setA(44.947); MQ135.setB(-3.445); // Configure the equation to calculate Toluen concentration
  float Toluen = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b v

  MQ135.setA(102.2 ); MQ135.setB(-2.473); // Configure the equation to calculate NH4 concentration va
  float NH4 = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b valu
```

图 26: MQ135 不同气体的系数

后续只需要与其他传感器相似，注册回调函数进行数据发送，即可在 Master 端接收到传感器带来的信息。
同理，我们可以如此操作 MQ5 传感器，以读取 CH4、H2、LPG 的 PPM 值（代表在空气中的浓度）。

```
I2C device found at address 0x68
The MacAddress of Slave Machine is: 1C:69:20:2B:9D:58
MPU6050 Found!
Calibrating MQ sensors, please wait.....  Finish
Roll(Left And Right): 0.04, Pitch(Up And Down): -0.01
Temperature = 27.23
Sent MPU6050 data successfully
------------------------------
Humidity: 65.80 %
Temperature: 26.60 *C
Sent DHT22 data successfully
--------------------------
CO: 1.66 PPM
Alcohol: 0.66 PPM
CO2: 1.52 PPM
Toluene: 0.26 PPM
NH4: 2.51 PPM
Acetone: 0.22 PPM
Sent MQ135 data successfully
--------------------------
```

图 27: MQ135 成功读取参数

5 .4 雨滴传感器

雨滴传感器有两个输出口，分别是 AO 与 DO，为了后续可以开发更多的功能，我们使用 AO（Analog Output）输入。检测是否下雨，如果下雨，则 SG90 舵机启动，模仿衣架收起的功能。

5 .5 LM393 光照传感器

光照传感器同样使用 Analog Output 模拟输出，为后续开发更多功能打基础。
结合雨滴传感器与光照传感器，还可以设置在夜间自动收回衣服。

5 .6 火焰传感器

检测火焰的强度，在没有火焰时 Analogread() 的值为最大值，火焰越近、燃烧程度越大，得到的模拟信号值越小。

四 项目心得总结

在本次课程项目的实践过程中，发现了许多想象中没有出现的问题，并且通过搜索了很多资料，找到了许多方法压缩代码，节省时间，压缩内存空间。例如：

1 编程技巧

1.1 宏定义与头文件规范

定义时，我总希望能够使得 main.cpp 显得不那么冗杂，所以希望开启新的 config.h 来搜集一切宏定义和全局变量。

搜索资料后发现，头文件通常用于声明变量，而不是定义变量。对于全局变量，应该在头文件中使用 extern 关键字来声明，然后在一个源文件中实际定义。并且注释部分的信息（如创建者信息和联系方式）最好放在实现文件（.cpp）中，而不是头文件中。

1.2 宏定义与构造函数接口的冲突

在使用 MQUnifiedsensor.h 库时，public 接口如下所示：

`MQUnifiedsensor(String Placa = "Arduino", float Voltage_Resolution = 5, int ADC_Bit_Resolution = 10, int pin = 1, String type = "CUSTOM MQ");`

我在 configuration.h 中使用 `#define Voltage_Resolution 3.3`，出现以下报错：

```
: note: in expansion of macro 'Voltage_Resolution'
 MQUnifiedsensor(String Placa = "Arduino", float Voltage_Resolution =  5, int ADC_Bit_Resolution = 10, int pin = 1, String type = "CUSTOM MQ");
```

报错原因：宏定义是字符替换，这里编译时会变成"3.3 = 5"，故导致编译错误。

1.3 使用"F" 宏处理字符串

F() 宏：当使用 F() 宏时，字符串常量会存储在 Flash 存储器中，而不是默认的 SRAM 中。这意味着可以减少动态内存的使用，提高程序的运行效率。如果不使用 F() 宏，所有字符串常量（例如"Humidity: "）都会存储在 SRAM 中。对于内存较少的设备（如 Arduino Uno）来说，这会迅速消耗有限的 SRAM，导致“内存不足”错误。

参考资料：https://blog.csdn.net/fang_chuan/article/details/80029546

使用正则表达式将 println\("([^"]*)"\) 替换为 println(F("$1")) 即可。

1.4 PlatformIO Libraries 处理方案

在 Arduino IDE 上编辑时，库是容易导入的，但对于大文件和项目组织的管理，Arduino IDE 的表现令人不满。于是我选择转向了 VSCode + Platform 开发，但 Clion 中许多 VSCode 没有的功能更加吸引我，所以转向了 Jetbrains 家族。

但是初次接触时，我被其复杂性所惊讶，在互联网上搜索相关的导入库的教程也是少之又少，迫不得已到 StackOverflow 上搜索，并将部分优秀结果自我吸收后作了整理写入了个人的 bilibili 专栏。

阅读链接：www.bilibili.com/read/cv37521671

1.5 ESP32 崩溃后使用 Backtrace 追踪错误代码

在上传后，有时会出现如下的报错：

```
Guru Meditation Error: Core  1 panic'ed (LoadProhibited). Exception was unhandled.

Core  1 register dump:
PC      : 0x400f9754  PS      : 0x00060a30  A0      : 0x800d29a5  A1      : 0x3ffb21e0
A2      : 0x3ffbdb7c  A3      : 0x3ffc2218  A4      : 0x00000008  A5      : 0x3ffc0b28
A6      : 0x3ffc48f0  A7      : 0x3ffc48f0  A8      : 0x00000000  A9      : 0x3ffb21d0
A10     : 0x3ffb223c  A11     : 0x400d6370  A12     : 0x00000003  A13     : 0x00000001
A14     : 0x00000015  A15     : 0x3ffb8eb0  SAR     : 0x0000001d  EXCCAUSE: 0x0000001c
EXCVADDR: 0x0000004c  LBEG    : 0x40086b69  LEND    : 0x40086b79  LCOUNT  : 0xffffffff


Backtrace: 0x400f9751:0x3ffb21e0 0x400d29a2:0x3ffb2210 0x400d33e9:0x3ffb2270 0x400d942d:0x3ffb2290
```

图 28: ESP32 崩溃

通过查找资料，可以使用 ESP32 自带的工具：xtensa-esp32s3-elf-gcc 来追溯源代码

命令如下：

```
PS C:\Users\26421\AppData\Local\Arduino15\packages\esp32\tools\xtensa-esp32s3-elf-gcc\esp-2021r2-patch5-8.4.0\bin> .\xtensa-esp32s3-elf-addr2line -pfiaC -e C:\Users\26421\Desktop\Maker\Slave\.pio\build\upesy_wroom\firmware.elf 0x400d29a2:0x3ffb2210 0x400d33e9:0x3ffb2270 0x400d942d:0x3ffb2290
```

其中，-e 参数指定了 elf 文件路径，0x400d29a2:0x3ffb2210 0x400d33e9:0x3ffb2270 0x400d942d:0x3ffb2290 分别是三个崩溃点的地址。效果如下所示：

```
PS C:\Users\26421\AppData\Local\Arduino15\packages\esp32\tools\xtensa-esp32s3-elf-gcc\esp-2021r2-patch5-8.4.0\bin> .\xtensa-esp32s3-elf-addr2line -pfiaC -e C:\Users\26421\Desktop\Maker\Slave\.pio\build\upesy_wroom\firmware.elf 0x400f9751 0x3ffb21e0 0x400d29a2:0x3ffb2210 0x400d33e9:0x3ffb2270 0x400d942d:0x3ffb2290
0x400f9751: tcp_output at /home/runner/work/esp32-arduino-lib-builder/esp32-arduino-lib-builder/esp-idf/components/lwip/lwip/src/core/tcp_out.c:1333 (discriminator 1)
0x3ffb21e0: ?? ??:0
0x400d29a2: _stext at ??:?
0x400d33e9: loopMPU6050() at C:\Users\26421\Desktop\Maker\Slave/src/MPU6050/MPU6050.cpp:94
0x400d942d: __adcAttachPin at C:/Users/26421/.platformio/packages/framework-arduinoespressif32/cores/esp32/esp32-hal-adc.c:122
```

图 29: ESP32 Backtrace 追踪错误代码

2 个人体验

本项目乃是我大一一整年做过体量最大、投入最多的项目，很感谢创客实践让我从对嵌入式一窍不通到变得感兴趣。几乎一切都是从零开始，觉得最有挑战性的部分是涉及到舵机、机械结构的运动学解算，这一部分是需要和外校的同学交流后才知道的。最觉得有收获的部分是，一点一点地实现语音输入和语音识别、到最后 TTS 模块的成功运用。

那么，有了语音识别之后，可以再做一些什么呢？不如加一些传感器读取一下个人身体上、房间内、城市里的各种参数吧！语音识别和输出，显然可以让生活更智能，在项目的构建过程中，也越来越体会到框架的美感。还有将函数重写时，变得简约易读的程序员之美，在完成一切工作后都体现得淋漓尽致！

为了做一个更漂亮的排版，我去学习了 LateX，为了更好的编程和组织文件，我去学习了 PlatformIO，最后为了尝试更多东西，跨入了 SolidWorks 的门槛。在学习的过程中也接触了很多新概念和新玩法，例如卡尔曼滤波、sha256 算法，API 的鉴权处理，甚至到 ws 协议和 wss 协议的区别。

由于本人在家中尝试搭建家中的路由器组网，所以对网络的了解也更深了一层，能够很快切入 ESP32 的 $AC + AP$ 模式，甚至有了用 ESP32 构建 Mesh 网络的想法。

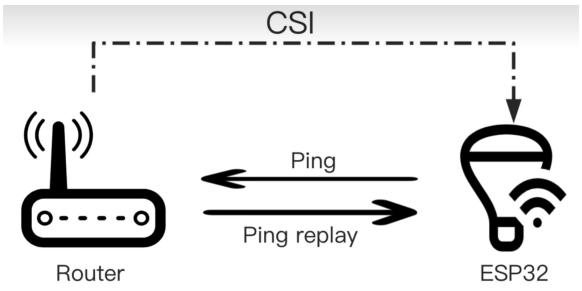
当然，由于很多部分是第一次探索，搜索了很多资料，也不缺少对各种大模型的使用，但在使用大模型的同时，我将大模型部署到了本地，也许这本身更让人愉悦。为什么要去实现本来看起来很困难的事情，因为山就在那里。

3 未来设想

CSI（Channel State Information）用于描述无线通信中的信道特性，可以反映出无线信号在传播过程中受到的多径、衰减、干扰等影响。CSI 在 Wi-Fi 系统中广泛应用，尤其是基于 OFDM（正交频分复用）技术的 Wi-Fi 标准，如 802.11n/ac/ax。在这些标准中，CSI 可以用于优化传输性能、定位和环境感知等应用。

南京大学无线感知前沿技术论坛在 8 月 1 日分享了 CSI 的前沿研究成果，有望实现毫米级的动态感应。参考链接：https://www.bilibili.com/video/BV1Cv411q7Cz，这一部分已有人通过算法实现，原理如同切割磁感线一般，可以通过 Wifi 强度在人为动作后感应信道的变化，从而读取人类活动，应用前景广泛，将来我很可能从事这方面的研究。

参考链接：用 ESP32-S3 和 CSI 技术打造人体感知风扇

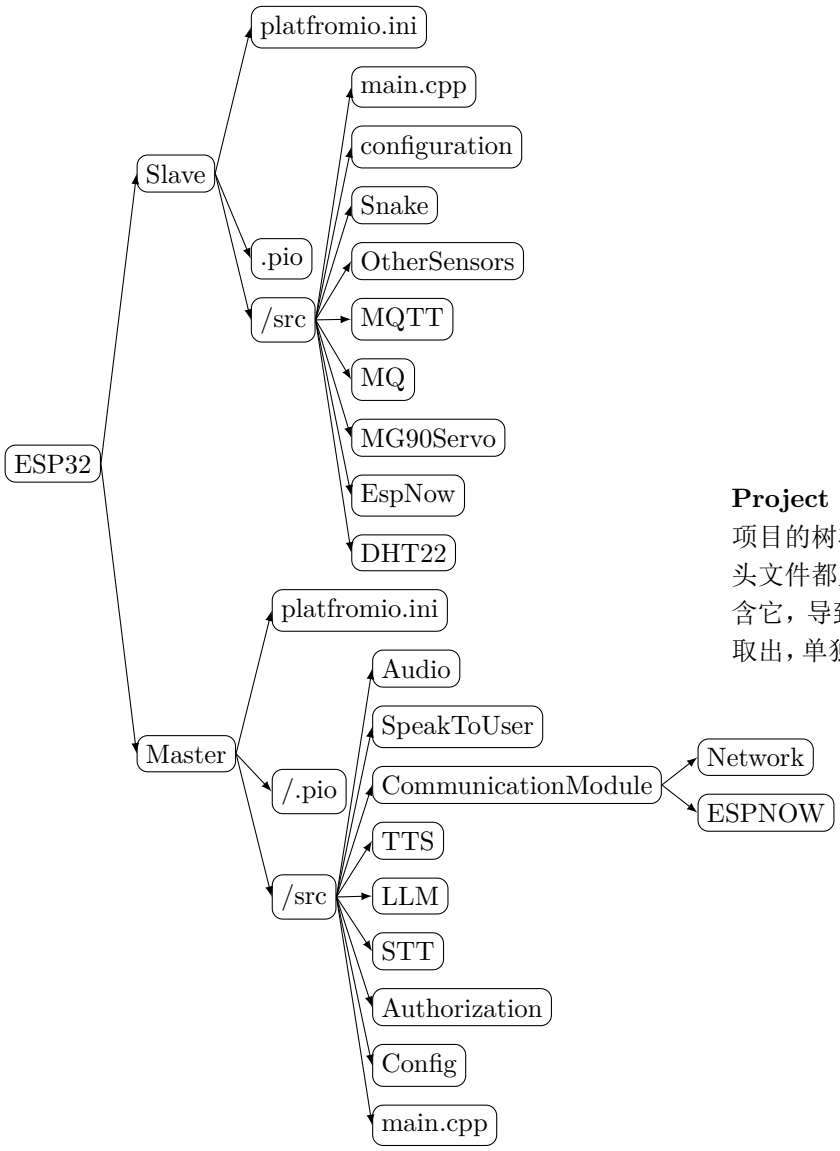


人的动作会干扰Wi-Fi信号的传播路径

图 30: CSI 与 ESP32-S3 结合示意图

五 附录

1 项目布局及构建



Project Structure:
项目的树状结构如图所示，最初编译时我将所有的头文件都置于 Configuration 中，每一份文件都包含它，导致编译非常缓慢，于是将不重合的部分库取出，单独置于需要的文件中，编译速度显著提升。

2 参考资料

- Ollama API 文档：https://github.com/ollama/ollama/blob/main/docs/api.md#create-a-model
- 心知天气 API 文档：https://seniverse.yuque.com/hyper_data/api_v3
- 讯飞 API 接口文档：https://xfyun.cn/doc
- OneNET 平台文档：https://open.iot.10086.cn/doc/v5/fuse/detail/920
- ————————参考了很多 Github 库中的 Example ————————
- ESPRessif 各种库文档：https://docs.espressif.com/projects/arduino-esp32/en/latest/libraries.html
- U8g2 OLED 库：https://github.com/olikraus/u8g2
- ArduinoJson 库：https://arduinojson.org/
- HTTPClient 库：https://github.com/espressif/arduino-esp32/tree/master/libraries/HTTPClient

- ESP-I2S 协议库：https://github.com/schreibfaul1/ESP32-audioI2S
- Adafruit 开源硬件公司各常用模块、传感器介绍网站：https://lastminuteengineers.com/electronics/arduino-projects/
- ——————————————————————————————
- 【波特律动】在线串口调试助手：https://serial.keysking.com/
- VOFA+ 1.3.10：https://www.vofa.com/
- ——————————————————————————————
- MPU6050 姿态解算 2-欧拉角 & 旋转矩阵：https://zhuanlan.zhihu.com/p/195683958
- 学习心得 | 基于卡尔曼滤波的 MPU6050 姿态解算：https://www.bilibili.com/video/BV1sL411F7fu
- 对 MQ 系列传感器采集电压与浓度转换的公式的探索：https://zhuanlan.zhihu.com/p/453499554
- 南京大学 CSI 前沿技术分享：https://www.bilibili.com/video/BV1Cv411q7Cz
- 新版微信小程序连接到 OneNET 平台：https://blog.csdn.net/2401_83704192/article/details/138913230
- ESP32 发声：https://github.com/MetaWu2077/Esp32_VoiceChat_LLMs
- 微信开发者工具使用文档：https://developers.weixin.qq.com/miniprogram/dev/devtools/edit.html
- ——————————————————————————————
- Unix 时间戳转换工具：https://www.jyshare.com/front-end/852/
- ESPNOW 通信不成功：https://esp32.com/viewtopic.php?p=132946
- ——————————————————————————————
- 华东师范大学软件学院实验报告、Beamer 模板（本人所作）：https://github.com/Shichien/ECNU-LateX-Template

Special Thanks

@ Azazo1
@ Luryem