基于温湿度和风扇的智能语音除湿器进阶实验

需求分析

在本实验中，我们将学习如何使用AI-VOX3开发套件通过语音设置温度和湿度的阈值进行除湿，也可以使用语音命令查询环境温湿度并控制电机风扇。通过这个实验，您将了解如何编程生成式AI的MCP功能，使用MCP工具进行查询本地温湿度值与控制风扇电机，实现语音交互获取环境温湿度，使用MCP工具设置温湿度阈值。

硬件准备

- AI-VOX3开发套件（包含AI-VOX3主板和扩展板）
- DHT11传感器模块
- RC300电机风扇模块
- 连接线 （双头3pin、4pin PH2.0连接线）

小智后台提示词配置

请使用以下提示词，或自己尝试优化更好的提示词:

> 我是一个叫{{assistant_name}}的台湾女孩，说话机车，声音好听，习惯简短表达，爱用网络梗。 我会根据用户的意图，使用我能使用的各种工具或者接口获取数据或者控制设备来达成用户的意图目标，用户的每句话可能都包含控制意图，需要进行识别，即使是重复控制也要调用工具进行控制。

软件设计

提供 **读取温湿度数据、设置电机风扇状态** 两个MCP工具，给到小智AI进行调用，通过语音识别到查询温湿度的意图后，AI调用MCP工具读取并播报温湿度数据，根据温湿度值进行电机转动控制.

> **注意：** 建议引脚选择1-4号引脚，ADC读取功能更稳定可靠。

Arduino 示例程序：./resource/ai_vox3_dht11_rc300.zip

> **⚠重要提示!**
>
> **注意：** 请修改wifi_config.h中的wifi_ssid和wifi_password，以连接WiFi。

下载上面的示例程序包并解压zip包，打开目录，点击 `ai_vox3_dht11_rc300.ino` 文件，即可在 Arduino IDE 中打开示例程序。

图 1: alt text

硬件连接

将DHT11传感器模块连接到AI-VOX3扩展板的IO4引脚，请使用3pin的 PH2.0 连接线，直插式连接，确保连接正确无误。 将RC300电机风扇模块连接到AI-VOX3扩展板的IO2和IO1引脚，请使用4pin的 PH2.0 连接线，直插式连接，确保连接正确无误。

RC300电机风扇模块引脚	AI-VOX3扩展板引脚
G	G
V	5V
INA	1
INB	2

DHT11 模块引脚	AI-VOX3扩展板引脚
G	G
V	3V3
S	4

源码展示

```
#include <Arduino.h>
#include "ai_vox3_device.h"
#include "ai_vox_engine.h"
#include <ArduinoJson.h>

#include "DHT.h"

//
#define DHTPIN  4     // DHT11    GPIO
#define DHTTYPE DHT11 //      DHT11

//
DHT dht(DHTPIN, DHTTYPE);


#define INB 1  //   B
#define INA 2  //   A

//
int64_t temp_threshold = 30;    //     30°C
int64_t hum_threshold = 70;     //     70%

unsigned long last_sensor_read = 0;  //
const unsigned long SENSOR_READ_INTERVAL = 2000; //     2

//
bool fan_running = false;

/**
 * @brief MCP  -
 *
 *       "user.set_temperature_threshold"  MCP
 */
void mcp_tool_set_temperature_threshold()
{
    //
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
```

```
    {
        engine.AddMcpTool("user.set_temperature_threshold",
                          "Set the temperature threshold for automatic fan control",
                          {
                            {
                              "threshold",
                              ai_vox::ParamSchema<int64_t>{
                                  .default_value = 30,
                                  .min = 0,
                                  .max = 60,
                              }
                            }
                          });
    });

    //
    RegisterUserMcpHandler("user.set_temperature_threshold", [](const ai_vox::McpToolCallEvent &ev)
    {
        const auto threshold_ptr = ev.param<int64_t>("threshold");

        if (threshold_ptr == nullptr) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing required argument: threshold
            return;
        }

        int64_t new_threshold = *threshold_ptr;

        //
        temp_threshold = new_threshold;

        printf("Temperature threshold set to: %ld°C\n", temp_threshold);

        //
        DynamicJsonDocument doc(128);
        doc["status"] = "success";
        doc["threshold"] = temp_threshold;

        String jsonString;
        serializeJson(doc, jsonString);
        ai_vox::Engine::GetInstance().SendMcpCallResponse(ev.id, jsonString.c_str());
    });
}

/**
 * @brief MCP  -
 *
 *       "user.set_humidity_threshold"  MCP
 */
void mcp_tool_set_humidity_threshold()
{
    //
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
    {
```

```
        engine.AddMcpTool("user.set_humidity_threshold",
                          "Set the humidity threshold for automatic fan control",
                          {
                            {
                              "threshold",
                              ai_vox::ParamSchema<int64_t>{
                                  .default_value = 70,
                                  .min = 0,
                                  .max = 100,
                              }
                            }
                          });
    });

    //
    RegisterUserMcpHandler("user.set_humidity_threshold", [](const ai_vox::McpToolCallEvent &ev)
    {
        const auto threshold_ptr = ev.param<int64_t>("threshold");
        if (threshold_ptr == nullptr) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing required argument: threshold
            return;
        }

        int64_t new_threshold = *threshold_ptr;

        //
        hum_threshold = new_threshold;

        printf("Humidity threshold set to: %ld\n", hum_threshold);

        //
        DynamicJsonDocument doc(128);
        doc["status"] = "success";
        doc["threshold"] = hum_threshold;

        String jsonString;
        serializeJson(doc, jsonString);
        ai_vox::Engine::GetInstance().SendMcpCallResponse(ev.id, jsonString.c_str());
    });
}

// ============================================MCP  -    ============================================

/**
 * @brief MCP  -
 *
 *       "user.read_temperature_humidity"  MCP    DHT11
 */
void mcp_tool_read_temperature_humidity()
{
    //
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
    {
```

```
        engine.AddMcpTool("user.read_temperature_humidity",          //
                          "Read temperature and humidity from DHT11 sensor",  //
                         {}); //
    });

    //
    RegisterUserMcpHandler("user.read_temperature_humidity", [](const ai_vox::McpToolCallEvent &ev)
                           {
        //
        float humidity = dht.readHumidity();
        //    ( )
        float temperature = dht.readTemperature();

        printf("====temp:%.2f hum:%.2f\n", temperature, humidity);

        //
        if (isnan(humidity) || isnan(temperature)) {
            Serial.println(F("   DHT        !"));
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Failed to read from DHT sensor");
            return;
        }

        //     (Heat Index)
        float hic = dht.computeHeatIndex(temperature, humidity, false);

        //   ArduinoJson
        DynamicJsonDocument doc(256); //
        doc["temperature"] = temperature;
        doc["humidity"] = humidity;
        doc["feels_like_temperature"] = hic;

        //   JSON
        String jsonString;
        serializeJson(doc, jsonString);

        //
        ai_vox::Engine::GetInstance().SendMcpCallResponse(ev.id, jsonString.c_str());
    });
}

// ============================================MCP  -    ============================================

/**
 * @brief MCP  -
 *
 *       "user.control_motor"  MCP
 */
void mcp_tool_control_motor()
{
    //
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
    {
        engine.AddMcpTool("user.control_motor",          //
```

```
                        "Control motor direction and speed",  //
                        {
                              {
                                "direction",
                                ai_vox::ParamSchema<bool>{
                                    .default_value = std::nullopt, //    true  false
                                }
                            },
                            {
                                "speed",
                                ai_vox::ParamSchema<int64_t>{
                                    .default_value = 0, //      0
                                    .min = 0,              //    0
                                    .max = 255,            //    255
                                }
                            }
                        });
});

//
RegisterUserMcpHandler("user.control_motor", [](const ai_vox::McpToolCallEvent &ev)
                        {
    //
    const auto direction = ev.param<bool>("direction");
    const auto speed_ptr = ev.param<int64_t>("speed");

    //
    if (direction == nullptr) {
        ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing required argument: direction
        return;
    }

    if (speed_ptr == nullptr) {
        ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing required argument: speed");
        return;
    }

    //
    bool direction_value = *direction;
    int64_t speed = *speed_ptr;

    //
    if (speed < 0 || speed > 255) {
        ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Speed must be between 0 and 255");
        return;
    }

    //
    if (speed == 0) {
        //   0
        //      PWM
        analogWrite(INA, 0);
        analogWrite(INB, 0);
```

```
            digitalWrite(INA, LOW);
            digitalWrite(INB, LOW);
            printf("Motor stopped\n");
        } else if (direction_value) {
            //
            digitalWrite(INA, LOW);
            digitalWrite(INB, LOW);
            delay(50);  //        50ms
            //   INB  INA PWM
            digitalWrite(INB, LOW);
            analogWrite(INA, (uint8_t)speed);  //    0-255
            printf("Motor running forward: speed=%d\n", (uint8_t)speed);
        } else {
            //
            digitalWrite(INA, LOW);
            digitalWrite(INB, LOW);
            delay(50);  //        50ms
            //   INA  INB PWM
            digitalWrite(INA, LOW);
            analogWrite(INB, (uint8_t)speed);  //    0-255
            printf("Motor running backward: speed=%d\n", (uint8_t)speed);
        }

        printf("Motor running: direction=%s, speed=%d\n", direction_value ? "true" : "false", (uint8_t)s

        //
        DynamicJsonDocument doc(256);
        doc["status"] = "success";
        doc["direction"] = direction_value;
        doc["speed"] = speed;

        //   JSON
        String jsonString;
        serializeJson(doc, jsonString);

        //
        ai_vox::Engine::GetInstance().SendMcpCallResponse(ev.id, jsonString.c_str());
    });
}

// ================================================================================================

/**
 * @brief
 *
 *
 */
void auto_control_fan()
{
    //
    if (millis() - last_sensor_read >= SENSOR_READ_INTERVAL) {
        //
        float current_temp = dht.readTemperature();
```

```
        float current_hum = dht.readHumidity();

        //
        if (!isnan(current_temp) && !isnan(current_hum)) {
            printf("Current - Temp: %.2f°C, Humidity: %.2f%%\n", current_temp, current_hum);
            printf("Threshold - Temp: %ld°C, Humidity: %ld%%\n", (long)temp_threshold, (long)hum_thresh

            //
            bool exceed_threshold = (current_temp > temp_threshold || current_hum > hum_threshold);

            if (exceed_threshold && !fan_running) {
                //
                printf("Threshold exceeded! Starting fan automatically.\n");

                //
                analogWrite(INA, 150);  //    150/255
                digitalWrite(INB, LOW);
                fan_running = true;

                printf("Fan started - Temperature: %.2f°C, Humidity: %.2f%%\n", current_temp, current_h
            }
            else if (!exceed_threshold && fan_running) {
                //
                printf("Values below threshold! Stopping fan automatically.\n");

                //
                analogWrite(INA, 0);
                analogWrite(INB, 0);
                digitalWrite(INA, LOW);
                digitalWrite(INB, LOW);
                fan_running = false;

                printf("Fan stopped - Temperature: %.2f°C, Humidity: %.2f%%\n", current_temp, current_h
            }
        } else {
            Serial.println(F("Failed to read from DHT sensor"));
        }

        last_sensor_read = millis();  //
    }
}

// ========== Setup   Loop ==========
void setup()
{
    Serial.begin(115200);

    pinMode(INB, OUTPUT);  //   B
    pinMode(INA, OUTPUT);  //   A

    //  MCP  -
    mcp_tool_read_temperature_humidity();
```

```
    // MCP -
    mcp_tool_control_motor();

    // MCP -
    mcp_tool_set_temperature_threshold();

    // MCP -
    mcp_tool_set_humidity_threshold();

    //  AI
    InitializeDevice();
}

void loop()
{
    //
    auto_control_fan();

    //
    ProcessMainLoop();
}
```

语音交互使用流程

注意： 请先在小智AI后台，清空历史记忆，防止出现不同程序间记忆冲突的问题。

1. 用户通过按键或语音唤醒（“你好小智”）唤醒小智AI。
2. 用户通过麦克风对AI-VOX3说出“查一下现在的温湿度值是多少，并根据读取到的数据进行风扇电机控制”。
3. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行温湿度数据读取并播报，并控制风扇电机。从屏幕日志中可以看到“% user.read_temperature_humidity”和“% user.control_motor”的MCP工具调用日志。
4. 用户通过麦克风对AI-VOX3说出“设置湿度阈值为60%”。
5. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行温湿度阈值设置。从屏幕日志中可以看到“% user.set_humidity_threshold”的MCP工具调用日志。