语音控制摇头风扇进阶实验

课程目标

在本实验中，我们将学习如何使用AI-VOX3开发套件通过语音命令控制基于SG90舵机的摇头速度和基于RC300电机风扇风速等功能。通过这个实验，您将了解如何编程生成式AI的MCP功能，并将其与舵机和电机控制逻辑结合起来，实现智能语音交互控制摇头风扇。

- 学习RC300电机风扇的基本使用方法
- 学习如何使用MCP工具控制电机风扇开关和风速

硬件准备

- AI-VOX3开发套件（包含AI-VOX3主板和扩展板）
- RC300电机风扇模块
- 连接线 （双头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 **设置风扇风速挡位** 的MCP工具，给到小智AI进行调用，通过语音识别到控制风扇风速挡位的意图后，AI调用MCP工具控制电机风扇风速。

Arduino 示例程序：resource/ai_vox3_rc300.zip

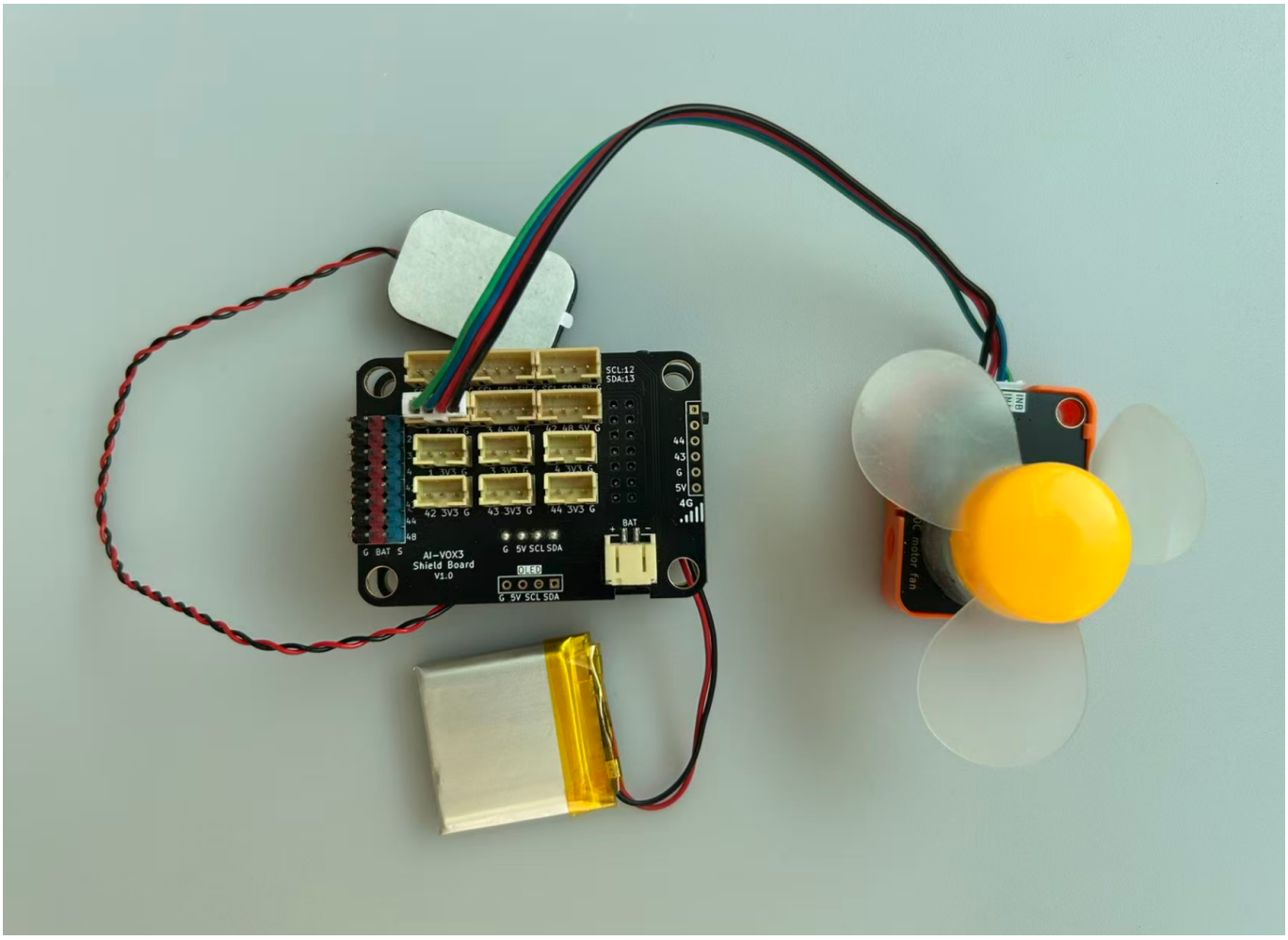
> ⚠**重要提示！**
>
> **注意：** 请修改wifi_config.h中的wifi_ssid和wifi_password，以连接WiFi。

下载上面的示例程序包并解压zip包，打开目录，点击 `ai_vox3_rc300.ino` 文件，即可在 Arduino IDE 中打开示例程序。

硬件连接

将RC300电机模块连接到AI-VOX3扩展板的IO1、IO2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电机模块引脚	AI-VOX3扩展板引脚
G	G
V	5V
INA	1
INB	2



源码展示

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

// ==========================================硬件配置
==========================================
constexpr auto kServoPin = 42;      // 舵机控制引脚 (GPIO 42)
constexpr auto kFanPin = 32;        // 风扇电机控制引脚 (GPIO 32)
```

```
#define INB 1  // 定义风扇电机B端口
#define INA 2  // 定义风扇电机A端口

constexpr uint32_t kMinPulse = 500;
constexpr uint32_t kMaxPulse = 2500;
constexpr uint16_t kMaxServoAngle = 180;

// 摇头范围定义
constexpr uint16_t kHeadLeftAngle = 0;   // 摇头左极限角度
constexpr uint16_t kHeadRightAngle = 180; // 摇头右极限角度
constexpr uint16_t kHeadCenterAngle = 90; // 摇头中间位置

// 风扇速度等级定义
constexpr uint8_t kFanSpeedLevels[4] = {0, 120, 190, 255}; // 0%, 47%, 66%, 100%
占空比

auto servo = em::Servo(kServoPin, 0, kMaxServoAngle, kMinPulse, kMaxPulse);

// 全局状态变量
struct FanState {
    uint8_t headSwingLevel = 0;    // 摇头挡位 (0-3)
    uint8_t fanSpeedLevel = 0;     // 风扇挡位 (0-3)
    uint32_t lastUpdate = 0;       // 上次更新时间
    bool isOscillating = false;    // 是否正在摆动
    uint16_t currentAngle = kHeadCenterAngle;  // 当前舵机角度
    bool directionRight = true;    // 摆动方向标志
};

FanState fanState;

// 设置风扇速度
void setFanSpeed(uint8_t level) {
    if (level > 3) level = 3;

    uint8_t speed = kFanSpeedLevels[level];

    // 正转：INB低电平，INA使用PWM控制转速
    digitalWrite(INB, LOW);
    analogWrite(INA, speed);  // 设置转速 (0-255)
    printf("Motor running forward: speed=%d\n", speed);

    fanState.fanSpeedLevel = level;
    printf("Fan speed set to level: %d (PWM: %d)\n", level, speed);
}

// 设置摇头挡位
void setHeadSwingLevel(uint8_t level) {
    if (level > 3) level = 3;

    fanState.headSwingLevel = level;

    if (level == 0) {
        // 0挡：停止摆动，回到中央位置
        servo.Write(kHeadCenterAngle);
```

```
        fanState.currentAngle = kHeadCenterAngle;
        fanState.isOscillating = false;
        printf("Head swing stopped, returned to center\n");
    } else {
        // 开始摆动
        fanState.isOscillating = true;
        printf("Head swing started at level: %d\n", level);
    }
}

// 摇头摆动控制函数
void handleHeadOscillation() {
    if (!fanState.isOscillating) return;

    // 根据挡位决定摆动速度，挡位越高摆动越快
    uint32_t oscillationInterval;
    switch(fanState.headSwingLevel) {
        case 1:
            oscillationInterval = 1500;  // 低速：0.5秒间隔
            break;
        case 2:
            oscillationInterval = 1000;  // 中速：0.3秒间隔
            break;
        case 3:
            oscillationInterval = 500;  // 高速：0.1秒间隔
            break;
        default:
            oscillationInterval = 1500;  // 默认低速
            break;
    }

    if (millis() - fanState.lastUpdate >= oscillationInterval) {
        // 改变舵机角度实现摆动
        if (fanState.directionRight) {
            fanState.currentAngle += 5;
            if (fanState.currentAngle >= kHeadRightAngle) {
                fanState.directionRight = false;
            }
        } else {
            fanState.currentAngle -= 5;
            if (fanState.currentAngle <= kHeadLeftAngle) {
                fanState.directionRight = true;
            }
        }
        printf("Servo angle updated: %d\n", fanState.currentAngle);
        servo.Write(fanState.currentAngle);
        fanState.lastUpdate = millis();
    }
}

// ==========================================MCP工具 - 控制摇头挡位
==========================================

/**
```

```
 * @brief MCP工具 - 控制摇头挡位
 *
 * 该函数注册一个名为 "user.control_head_swing" 的MCP工具，用于控制风扇摇头挡位
 */
void mcp_tool_control_head_swing()
{
    // 注册工具声明器，定义工具的名称和描述
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
                              { engine.AddMcpTool("user.control_head_swing",
// 工具名称
                                                  "Control head swing level 0-3",
// 工具描述
                                                  {
                                                      {"level",

ai_vox::ParamSchema<int64_t>{
                                                           .default_value =
std::nullopt, // 挡位参数，默认值为空
                                                           .min = 0,
// 最小挡位为0
                                                           .max = 3,
// 最大挡位为3
                                                       }}}); });

    // 注册工具处理器，收到调用时，控制摇头挡位
    RegisterUserMcpHandler("user.control_head_swing", [](const
ai_vox::McpToolCallEvent &ev)
                           {
        // 解析参数
        const auto level_ptr = ev.param<int64_t>("level");

        // 检查必需参数是否存在
        if (level_ptr == nullptr) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing
required argument: level");
            return;
        }

        // 获取参数值
        int64_t level = *level_ptr;

        // 参数验证
        if (level < 0 || level > 3) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Level must be
between 0 and 3");
            return;
        }

        // 控制摇头挡位
        setHeadSwingLevel(static_cast<uint8_t>(level));
        printf("Head swing level set to: %d\n", static_cast<uint8_t>(level));

        // 创建响应
        DynamicJsonDocument doc(256);
```

```
        doc["status"] = "success";
        doc["level"] = level;
        doc["description"] = level == 0 ? "Head swing OFF" :
                             level == 1 ? "Head swing LOW" :
                             level == 2 ? "Head swing MEDIUM" : "Head swing HIGH";

        // 将 JSON 文档转换为字符串
        String jsonString;
        serializeJson(doc, jsonString);

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

// ==========================================MCP工具 - 控制风扇挡位
==========================================

/**
 * @brief MCP工具 - 控制风扇挡位
 *
 * 该函数注册一个名为 "user.control_fan_speed" 的MCP工具，用于控制风扇风量挡位
 */
void mcp_tool_control_fan_speed()
{
    // 注册工具声明器，定义工具的名称和描述
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
                              { engine.AddMcpTool("user.control_fan_speed",
// 工具名称
                                                  "Control fan speed level 0-3",
// 工具描述
                                                  {
                                                      {"level",

ai_vox::ParamSchema<int64_t>{
                                                           .default_value =
std::nullopt, // 挡位参数，默认值为空
                                                           .min = 0,
// 最小挡位为0
                                                           .max = 3,
// 最大挡位为3
                                                       }}}); });

    // 注册工具处理器，收到调用时，控制风扇挡位
    RegisterUserMcpHandler("user.control_fan_speed", [](const
ai_vox::McpToolCallEvent &ev)
                           {
        // 解析参数
        const auto level_ptr = ev.param<int64_t>("level");

        // 检查必需参数是否存在
        if (level_ptr == nullptr) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing
required argument: level");
```

```
            return;
        }

        // 获取参数值
        int64_t level = *level_ptr;

        // 参数验证
        if (level < 0 || level > 3) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Level must be
between 0 and 3");
            return;
        }

        // 控制风扇挡位
        setFanSpeed(static_cast<uint8_t>(level));
        printf("Fan speed level set to: %d\n", static_cast<uint8_t>(level));

        // 创建响应
        DynamicJsonDocument doc(256);
        doc["status"] = "success";
        doc["level"] = level;
        doc["description"] = level == 0 ? "Fan OFF" :
                             level == 1 ? "Fan LOW" :
                             level == 2 ? "Fan MEDIUM" : "Fan HIGH";

        // 将 JSON 文档转换为字符串
        String jsonString;
        serializeJson(doc, jsonString);

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

// ========== Setup 和 Loop ==========
void setup()
{
    Serial.begin(115200);
    delay(500); // 等待串口初始化

    // 初始化舵机和风扇引脚
    if (!servo.Init())
    {
        printf("Error: Failed to init servo on pin %d\n", kServoPin);
    }

    pinMode(INB, OUTPUT);  // 设置电机B端口为输出模式
    pinMode(INA, OUTPUT);  // 设置电机A端口为输出模式

    // 设置初始状态
    servo.Write(kHeadCenterAngle);
    fanState.currentAngle = kHeadCenterAngle;

    // 注册MCP工具 - 摇头控制功能
```

```
    mcp_tool_control_head_swing();

    // 注册MCP工具 - 风扇控制功能
    mcp_tool_control_fan_speed();

    // 初始化设备服务，包括硬件和AI引擎，必备步骤
    InitializeDevice();

    printf("Smart Oscillating Fan initialized\n");
}

void loop()
{
    // 处理摇头摆动逻辑
    handleHeadOscillation();

    // 处理设备服务主循环事件， 必备步骤
    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“打开风扇”、“摇头设置为2挡”。
3. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行风扇的控制。从屏幕日志中可以看到“% user.control_head_swing”和“% user.control_fan_speed”的MCP工具调用日志。