《简易直播控制器》项目研发日志

姓名XXX 学号XXX 学院XXXX

1. **项目背景**

* **痛点和需求**

在直播时，可能需要频繁切换场景、调整直播内容、调整声音效果等。市面上的控制台价格普遍比较昂贵（如Elgato Stream Deck XL价格为1999元），因此考虑自制一个简易的控制台。

* **理想功能目标**

实现一键完成场景切换、播放声音、文本内容修改等功能。

* **项目组成和机理**
  + 向NodeMCU中写入程序，并将其与电脑、TM1638连接起来。
  + 在TM1638上按下按键，电脑上的直播即可实现对应的变化，如场景切换等。
* **项目中的难点和制约**
  + 难点1：捕捉obs的场景、来源等信息需要对obs的工作原理进行研究。
  + 难点2：从外部对各种声音输入输出进行控制需要研究。
  + 难点3：可能需要编写大量程序。

1. **项目计划简表**

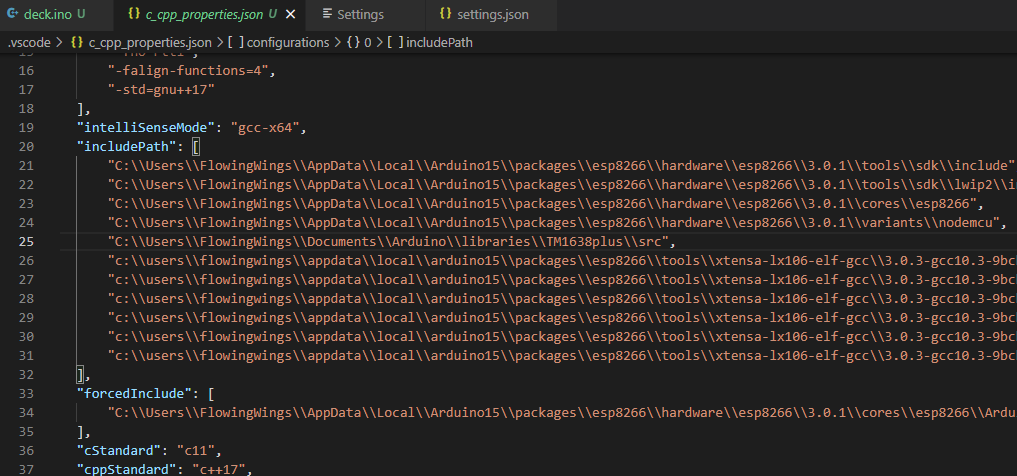
|  |  |  |
| --- | --- | --- |
| **步骤** | **功用** | **估计用时** |
| 配置VS Code环境 | 利用自动补全提高工作效率 | 1小时 |
| 编写要写入NodeMCU的程序 | 在控制器和主机之间传递信息 | 1小时 |
| 编写要在主机运行的程序 | 实现具体的各种功能 | 6小时 |
| 连接各组件并运行 | 实际应用 | 1小时 |
| **预期1月12日以前，完成** 90% | | |

1. **研发日志**

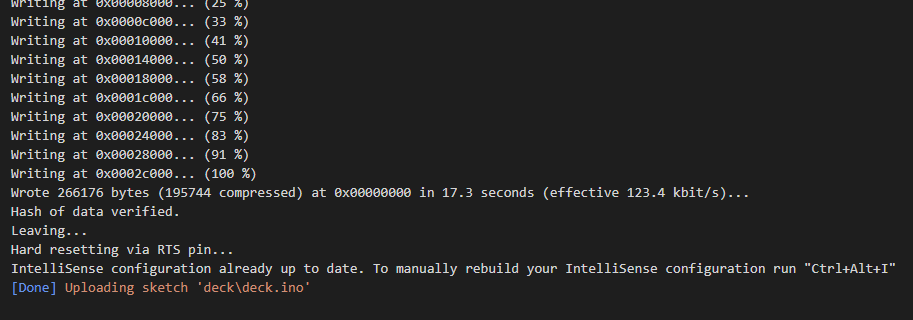
**步骤1：配置VS Code环境**

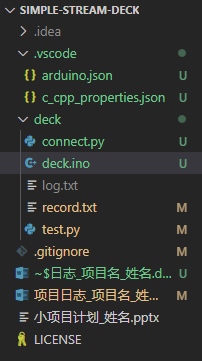
* **具体操作：**（可描述代码段、连线图、引用资料说明）
  + 在VS Code中下载微软官方的Arduino插件
  + 对插件进行设置。
    - 这个插件并不能独立完成Arduino IDE的功能，只是提供一个美观的前端然后调用已有的Arduino IDE而已，所以需要自行安装Arduino IDE并在插件设置中提供其位置。并且不能使用Windows商店中的Arduino IDE，因为那个是在沙盒中运行的，只能使用显式安装的Arduino IDE。设置方法是左下角齿轮 – Settings – 搜索Arduino – Arduino: Path – 填入Arduino IDE安装的位置。
    - 为了加快验证文件的速度，可以在.ino/../.vscode/arduino.json中添加一项”output”:”../build”，这样每次编译会在原先的基础上进行，速度会快不少。
  + 下载需要的开发板和库。
    - 本项目的开发板NodeMCU不在默认列表里，需要补充额外的URL。在VS Code中，步骤为左下角齿轮 – Settings – 搜索Arduino – Arduino: Additional Urls – Edit in settings.json – 添加一项"arduino.additionalUrls": ["http://arduino.esp8266.com/stable/package\_esp8266com\_index.json"]，然后按F1 - 搜索Arduino: Board Manager，刷新再搜索NodeMCU就能找到了。
      * 下载好之后，可以在右下角<Select Board>中选择这个开发板。
    - VS Code安装库的方法和Arduino IDE类似，也是通过库管理器直接安装。库管理器的打开方式是F1 – 搜索Arduino: Library Manager – 搜索所需的库，如TM1638plus，然后安装即可。但麻烦的地方在于安装好之后，运行一个.ino代码时，VS Code会在上级文件夹的/.vscode处 生成一个c\_cpp\_properties.json。需要把想包含的头文件的位置手动地放到这个json的configurations – includePath中。
      * 如果不知道库安装在哪里，可以参考"C:\\Users\\<user name>\\AppData\\Local\\Arduino15\\packages”，或者自己在电脑中搜索安装的库。Windows 10自带的搜索工具效率极低，推荐用Everything代替。
* **实现效果：**

现在可以在VS Code中编写和验证.ino代码并将其上传到NodeMCU了。



在配置文件中，手动设置的头文件所在的位置

上传到NodeMCU成功

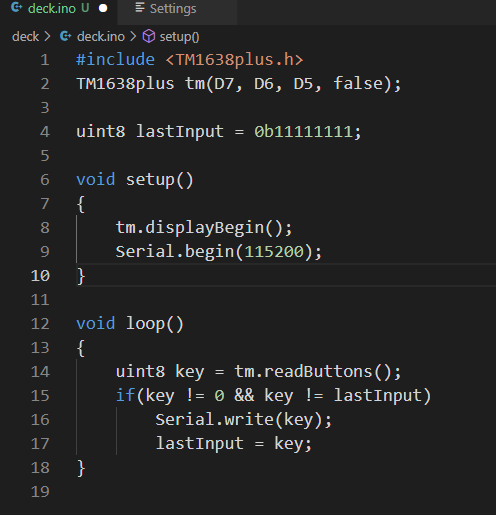


项目的结构

* **评价分析：**
  + 用时大约5小时，远超预期。一路上踩了特别多的坑。主要是因为VS Code的设置方式比较特殊，尤其是对于本项目，由于是C语言编写，所以VS Code总会自动生成一个优先级更高的配置文件c\_cpp\_properties.json覆盖掉全局性的设置。研究VS Code和Arduino IDE的设置区别并寻找解决方案花费了大量时间。
  + 考虑到.ino代码可能会比较简单，这次配置VS Code环境消耗的时间远超过自动补全能够节省的时间。
  + 但对Arduino的工作方式和VS Code的使用有了更深了解，所以也很有收获。
  + 如果要模仿的话，建议直接复制deck.ino到Arduino IDE中运行。

**步骤2：**编写要写入NodeMCU的程序

* **具体操作**
  + 在权衡和尝试多种方案之后，选择了比较简单的方案：NodeMCU只负责从TM1638读取按键并将其传递到主机，由connect.py读取这一输入并处理。
* **实现效果**

****

简单的代码

* **评价分析**
  + 和connect.py一共耗时大约6小时。代码本身比较简单，但消耗时间较大。一是因为对开发板的工作方式不熟悉；二是因为调试时要同时改动这个代码和connect.py，比较费劲。
  + 为了只在按键输入发生变化时传递，引入了变量uint8 lastInput，当前输入只有在与lastInput不同时才会被写入serial。

**步骤3：**编写要在主机运行的程序

* **具体操作**
  + 下载并安装OBS，创建若干场景，设置麦克风和摄像头。最重要的是，为每个想按键控制的功能设计一个热键。稍后在Python中通过模拟按键来触发这些功能。
    - 最好不要有任何热键是另一个热键的子序列。比如将切换场景的热键设置为Ctrl+Alt+C、同时将打开/关闭摄像头的热键设置为Ctrl+Alt+Shift+C。这样设置尽管不会被OBS拒绝，但使用时会导致打开/关闭摄像头时场景也会切换。
    - 为了实现文本的更改，OBS要添加一个来源，选择文本类型，并且勾选从文件读取。选择好要读取的文件。
  + 创建connect.py文件，写入代码。
    - 代码中的recordFile = open()的第一个参数应该和上一步中确定的OBS读取的文件相统一。
    - 代码中调用pyautogui.hotkey()的地方都是在模拟键盘输入来按下热键，它们应该和上一步中在OBS中设置的热键统一。
* **实现效果**

****

部分热键

* **评价分析**
  + 和deck.ino一共耗时大约6小时。原因同上。
  + 为了便于调试，创建了一个日志文件，每次操作都会向其中写入信息。这在帮助我调试时发挥了较大作用。

**步骤4：**测试

* **具体操作**
  + 连接各组件。
    - 连接方式：主机 – NodeMCU – TM1638。
  + 将deck.ino上传到开发板。
  + 运行OBS。
  + 运行connect.py。
  + 按键，观察效果。
* **实现效果**
  + 成功实现通过按键实现场景切换、麦克风打开/关闭、摄像头打开/关闭、文本修改。
* **评价分析**
  + 总共耗时约1小时，比较快。
  + 调试时发现了之前说的热键问题。

1. **总结展望**

* **目前本项目实际的完成情况是：**
  + 完成了90%
    - 现在可以在TM1638上进行场景切换、麦克风打开/关闭、摄像头打开/关闭、文本修改了。
  + 项目已上传至GitHub。
  + 没完成的功能包括播放音频、视频。也许可以通过让python执行命令行来实现。
* **本项目还可以继续完成/改进之处：**
  + TM1638的按键过少，导致只能实现8个功能。也许可以考虑将同时按下多个键映射为新的功能。
    - 老师给了4\*4的键盘，但是它需要7根公-公线才能连接，我没有这些线，开始做的太晚，于是也没有时间获得这些线了。
  + 实现功能选择的方式是大量的if分支，过于丑陋，扩展性很差。
  + 实现控制OBS的方式是模拟键盘，很简单但也过于粗暴，有误触发其他操作的危险。未来可能可以通过结合一些OBS插件或者OBS提供的API来进行更完善的控制。
  + 有人称通过pyautogui按键有一些延迟，我在大部分情况没有感知到这一延迟。
  + 战绩的修改是通过修改一个文本文件完成的，延迟约为2-3秒，有较大优化空间。