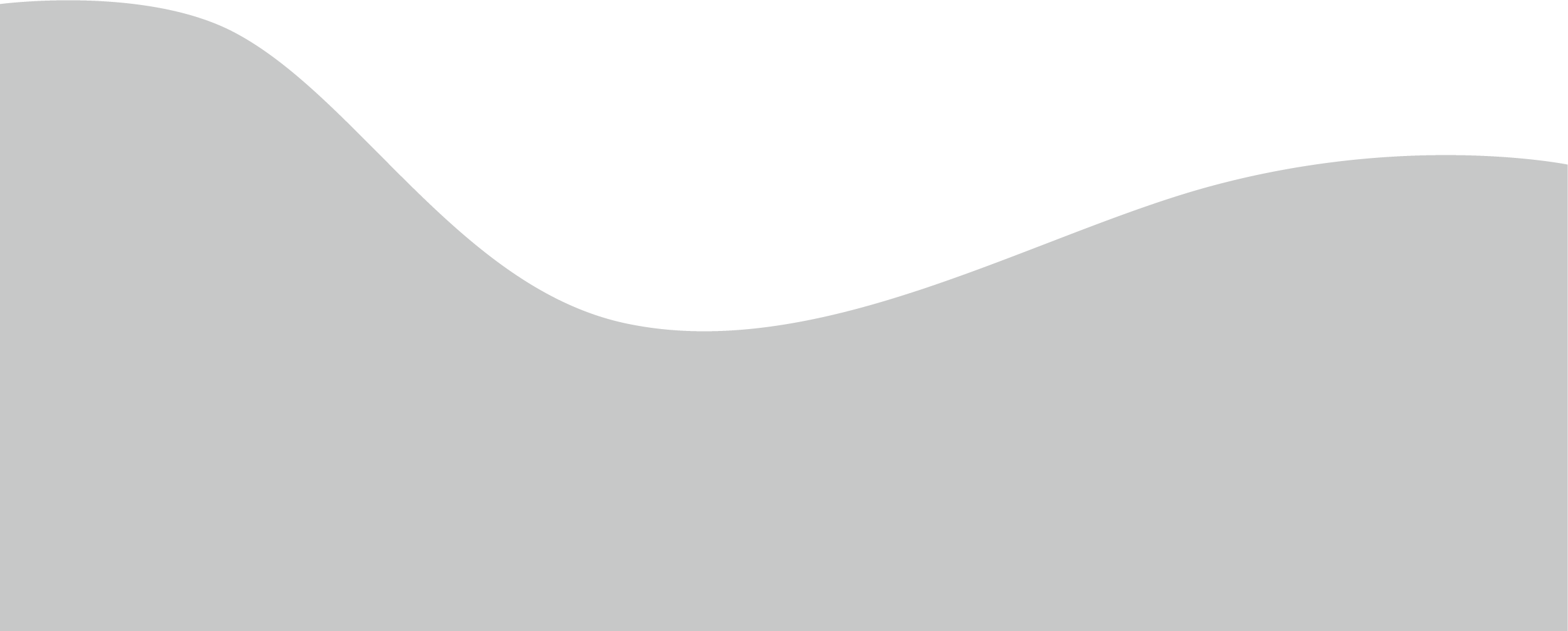


**课程设计报告**





|  |  |
| --- | --- |
| 设计题目 ： | 六自由度平台测控系统 |
| 指导教师 ： | 胡琼 |
| 专业班级 ： | 机械T2201班 |
| 学生学号 ： | 8203221008；8204222705；8203221201 |
| 学生姓名 ： | 谢绚冰 、薛丽杉、张赫曦 |



机电工程学院 制

##### 六自由度平台测控系统

##### 摘要

本项目为六自由度平台测控系统的设计。基于Stewart并联机器人的逆解理论知识，建立了六自由度平台运动学数学模型。结合运动学分析和牛顿迭代法，对六自由度平台的位置姿态进行迭代求解，解决了平台运动精度与解算效率的难题，精度达到 0.1。运用SolidWorks软件进行初步六自由度平台设计，并基于丝杆滑台机构对六自由度平台进行简化优化设计，提高工艺的合理性和经济性。本设计分析硬件电路知识，采用模块化封装MPU6050，集成化设计原理图并构建PCB。本设计基于数学原理知识，将六自由度平台的逆解以及位置控制封装在系统中，同时实现伺服报警以及嵌入式人机交互模块，对于六自由度平台测控系统设计建造具有一定的现实指导意义。

关键词：Stewart 并联机器人 控制

目录

[第一章 综合实践内容及要求 1](#_Toc19748)

[1.1 六自由度电动平台介绍 1](#_Toc16651)

[1.2 六自由度电动平台测控系统功能分析 1](#_Toc1521)

[1.3 嵌入式控制部分原理 2](#_Toc29146)

[1.4 测控系统具体要求 3](#_Toc26880)

[1.5 设计实践要求 3](#_Toc21216)

[第二章 六自由度平台运动学分析 4](#_Toc26490)

[2.1 六自由度平台三维空间坐标系平移变换 4](#_Toc12912)

[2.2 六自由度平台运动学求解 5](#_Toc30960)

[2.2.1 反解计算 5](#_Toc10811)

[2.2.2 正解计算 6](#_Toc22316)

[第三章 控制系统机械设计及实物制造 8](#_Toc24659)

[3.1 机械部分三维建模 8](#_Toc23966)

[3.1.1 机械模型设计目标 8](#_Toc364)

[3.1.2 三维建模soildworks软件简介 9](#_Toc20776)

[3.1.3 三维建模基本思路 9](#_Toc17903)

[3.2 丝杆滑台机构 10](#_Toc15191)

[3.2.1 丝杆滑台的组成部分 10](#_Toc8411)

[3.2.2 丝杆滑台的运动原理 11](#_Toc7237)

[3.3 机械部分参数指标 12](#_Toc28306)

[3.4 实物制造与展示 12](#_Toc4287)

[第四章 控制系统的硬件电路设计 13](#_Toc27376)

[4.1 硬件电路设计altiumdesigner 软件简介 13](#_Toc13921)

[4.2硬件电路详细设计 13](#_Toc12658)

[4.2.1 供电部分 13](#_Toc9667)

[4.2.2 晶振和复位部分 14](#_Toc24369)

[4.2.3 外设部分 17](#_Toc23285)

[4.3 PCB 详细设计 23](#_Toc28146)

[4.3.1 核心板PCB设计概述 23](#_Toc26320)

[4.3.2 供电部分PCB设计 23](#_Toc22698)

[4.3.3 晶振和复位部分 PCB 设计 25](#_Toc22508)

[4.3.4 PCB整体设计 26](#_Toc23841)

[4.3.5 上板PCB整体设计 27](#_Toc66)

[第五章 控制系统的程序设计和实现 28](#_Toc2962)

[5.1 程序设计软件简介及框架和逻辑结构 28](#_Toc9398)

[5.1.1程序设计软件简介 28](#_Toc7299)

[5.1.2 程序框架概述 29](#_Toc20000)

[5.1.3代码文件概述 31](#_Toc1579)

[5.2 STM32CubeMX配置 31](#_Toc28568)

[5.3 系统库函数 35](#_Toc22037)

[5.3.1 MyCode文件 35](#_Toc29702)

[5.3.2 HARDWARE文件 37](#_Toc14570)

[5.4 程序设计与解析 39](#_Toc5446)

[5.4.1原始变量与初始化 39](#_Toc10635)

[5.4.2主循环逻辑与代码 42](#_Toc12642)

[5.3.3中断函数逻辑与代码 44](#_Toc18904)

[参考文献 48](#_Toc14003)

[结语 49](#_Toc15068)

[附录 51](#_Toc17891)

[附录A：MPU6050欧拉角解算代码MyCode 51](#_Toc14136)

[附录B：平台位姿解算以及电机驱动代码HARDWARE 54](#_Toc24872)

[附录C：六自由度平台控制系统主函数 58](#_Toc2560)

# 第一章 综合实践内容及要求

## 1.1 六自由度电动平台介绍

六自由度Stewart平台（6-DOF Stewart Platform）是一种高性能的并联机器人，以其发明者Scott Stewart命名，它能够实现空间中6个自由度的独立运动包括三个平动自由度（X、Y、Z方向的直线运动）和三个转动自由度（绕X、Y、Z轴的旋转）。

该平台由一个上平台、一个下平台以及六根连杆（通常由伺服电机驱动的伸缩机构）组成，每根连杆通过球形接头分别连接上下平台，形成六个闭环的驱动链路，如图1.1所示。

#### 

#### 图1.1 六自由度电动平台的示意图

## 1.2 六自由度电动平台测控系统功能分析

（1）采用步进电机控制杆件长度：

在六自由度平台工作时，步进电机控制六条伸缩杆的伸缩运动以改变各杆件的长度。通过比较目标杆件长度与当前电机的长度，来决定电机的转动方向，最终实现上平台能够得到期望的位姿。

（2）采用MPU6050反馈姿态：

MPU6050 是由InvenSense公司生产的全球首款整合性六轴运动处理模块，集成了 3轴陀螺仪 和 3轴加速度计，常用于机器人、无人机、穿戴设备等需要检测运动和姿态的应用中。它可以实时获取运动物体的在[三维坐标系](https://so.csdn.net/so/search?q=%E4%B8%89%E7%BB%B4%E5%9D%90%E6%A0%87%E7%B3%BB&spm=1001.2101.3001.7020" \t "_blank)内的偏转角度，如图2.1所示。

#### 图片1

#### 图1.2 MPU6050芯片的检测轴和方向

旋转的方向按右手法则定义，即右手大拇指指向轴向，四指弯曲的方向即为绕该轴旋转的方向。X 轴角度（滚转角 Roll）即为绕 X 轴旋转方向的角度，Y 轴角度（俯仰角 Pitch）即为绕 Y 轴旋转方向的角度，Z轴角度（偏航角Yaw）即为绕 Z 轴旋转方向的角度，三者合称姿态角/欧拉角（Euler angles）。

（3）采用蜂鸣器和提示灯闪烁进行伺服报警：

通过检查每根杆件的长度是否在允许范围内来防止过长或过短的情况，避免对硬件造成损坏。如果检测到不正常的长度，系统会恢复到之前的状态并且发出警报。

（4）采用串口实现人机交互界面：

串口接收来自上位机的数据，并根据不同的命令来控制平台的角度和位移调整。对于每个接收到的指令，更新相应的角度值或平移值，计算新的支柱长度并用串口发送，实时显示平台各个电动缸的实际伸缩量。

（5）采用按键中断实现急停：

通过外部中断来检测按键操作，控制系统开启、急停和反转加减等功能。

## 1.3 嵌入式控制部分原理

六自由度电动平台采用平台STM32 芯片实现嵌入式控制，要想实现题目要求的测控系统，需要一个单片机最小系统和一系列外设。

单片机的最小系统就是让单片机能正常工作并发挥其功能时所必须的组成部分，也可理解为单片机正常运行的最小环境；构成最小系统主要有4 部分，分别为：

（1）单片机芯片—运行任务程序及执行相应的控制动作

（2）时钟电路—为单片机运行程序提供时钟源

（3）复位电路—使单片机内部各个模块处于确定的初始状态

（4）系统电源—提供工作电源

要使STM32 单片机正常运行，需具备以上四个电路。STM32 单片机内部已集成有时钟电路，所以STM32 单片机只需带有复位电路，晶振电路提供时钟源并且提供工作电源，便可正常运行，具体的电路设计将在第四章进行分析。

## 1.4 测控系统具体要求

（1）位置控制：当系统发出位置控制指令时，平台的六个电动缸能够根据接受到的信号在限定范围内进行伸缩运动，并最终实现上平台能够得到期望的位姿。

（2）伺服报警：当六自由度平台出现超载、电池警报、电机堵转等问题时，系统应当自动发出报警信号并自动关断电源进入保护状态。

（3）急停装置：若六自由度电动平台出现故障报警时，系统却不能马上停止运动，此时应通过急停按钮，立马关掉系统，防止运动平台出现进一步的严重事故。

（4）人机交互界面：一个完整的控制系统应当提供用户使用的交互界面，能够实时显示平台各个电动缸的实际伸缩量。

（5）寻零：每次平台使用结束后，平台应该能够自动完成回归零点位置。

## 1.5 设计实践要求

（1）采用stm32 单片机控制，基于Altium Designer 设计控制系统的硬件电路图、PCB 板；

（2）基于STM32CUBEMX 设计控制程序；

（3）基于SOLIDWORKS 设计实物；

（4）撰写设计报告。

# 第二章 六自由度平台运动学分析

运动学分析是研究六自由度平台的基础，它研究的是平台的位置、速度、加速度和时间的相对关系。

## 2.1 六自由度平台三维空间坐标系平移变换

如图2.1.1所示，可用RPY角描述六自由度平台的姿态。[1]将其行驶方向取为Z轴，则绕Z轴的旋转(角)称为滚转(Roll)；把绕Y轴的旋转(角)称为俯仰(Pitch)； 把竖直方向取为X轴，将绕X轴的旋转(角)称为偏航(Yaw)。

#### 

#### 图2.1.1 Stewart平台结构简图

设有固定坐标系为及活动坐标系，在初始状态时，两坐标系完全重合。当活动系的原点平移到绝对坐标（a，b，c)，及绕固定坐标系Z、Y、X轴分别旋转*α、β、γ*角时，根据空间坐标系变换原理，有以下的齐次坐标变换矩阵[2]：

(2-1)

其中cos简写为C，sin简写为S。

设点固连于活动系，用齐次坐标向量表示为，进行坐标变换后， *P’* 相对固定系移至了*P*点，*OP* 用齐次坐标向量表示为，则有以下空间坐标 变换公式：

(2-2)

如果坐标变换矩阵结构为3×3，则有：

(2-3)

上式称为欧拉变换矩阵，则空间坐标变换公式也可以表示为：

(2-4)

## 2.2 六自由度平台运动学求解

六自由度运动平台的位移和姿态大闭环控制需要用到运动学反解和正解。

### 2.2.1 反解计算

若已知上平台的位姿，而各杆件的长度未知，则各杆件长度的求解就是六自由度平 台的逆解计算。[3]通过给定的姿态数据计算出六个伸缩杆的位置长度，描述一个刚体在空间旋转的姿态，最常使用的方法是定义三个欧拉角来表达，当刚体旋转至某一姿态下，此三个欧拉角即组成唯一的旋转矩阵，并借由旋转矩阵作坐标转换，便可求得刚体的绝对位置。

如图2.2.1所示，[4]六自由度平台的结构参数主要有5个，分别是：上铰点分布圆半径 、上铰点分布角 、下铰点分布圆半径、下铰点分布角和平台处于初始位置时上下铰点的高度差*H*。

#### 

#### 图2.2.1 Stewart平台结构简图

为简化为见，以6个下铰点中心所在的平面为下平台，以6个上铰点中心所在的 平面设为上平台，将固定坐标系建立在下平台外接圆中心，将活动坐标系建立在上平台外接圆中心，如图2.2.1所示，平台初始零位时，坐标轴， ，与重合，当上平台位姿变化时，上平台即活动系位姿可用、 、、、、这6个参数来表达，其中为活动系原点的绝对坐标，、 、为上平台的姿态角，即分别为上平台法向量相对于下平台三个固定轴*Z、Y、 X*的转角。

活动系中任一点坐标与该点的绝对坐标的关系可按式 2-2计算。对于上平台各铰点，其坐标变换公式为：

(2-5)

当确定后，根据式2-1可求出转换矩阵H，则式2-5可求出点坐标，设各杆件的杆长初始值为，()为点的初始坐标， ()为点的坐标，可知*L0*为：

(2-6)

设各杆件伸缩后，其长度为*Li*，则可得：

(2-7)

式2-7即为六自由度平台的逆解公式，由其推导过程可知其求解步骤如下：

1. 根据上平台的位姿值，按式2-1求得变换矩阵H；
2. 根据上平台的几何尺寸，按图2.2.1求得上平台各铰点在活动系的相对坐标，根据式2-5求得上平台各铰点在固定系的绝对坐标；
3. 根据下平台的几何尺寸，求得下平台各铰点的坐标()；
4. 根据式2-7求得杆长值*Li*。

### 2.2.2 正解计算

若已知各杆件的长度，而上平台位姿未知，则位姿值的求解就是六自由度平台的解计算。

式 2-7 可化为：

(2-8)

令*-*

则式 2-8 可以转化为：

(2-9)

为易于理解，令*,*则式 2-9 可转化为：

(2-10)

式2-10即为六自由度平台的正解方程组。

按给定条件，*Li*及()是已知给定值，为未知值，由式2-2可 知，可由H计算确定，因而未知值转化为，由此可知，是非常复杂的非线性方程组，如要用解析法求解会很困难，迄今各国学者还没有很好的解决方案。随着计算机技术的发展，数值法已成为解决该问题的有效途径。

在这里可选择牛顿迭代法进行求解[5][6][7]。

对于非线性方程，用泰勒级数展开线性化表示，可得：

则上述非线性方程组可简化表示为：*-*,化简得：

则其牛顿迭代格式为： ,也可以化为

(2-11)

式 2-11 即为非线性方程组的牛顿迭代公式。

如前所述，设置迭代精度以及迭代误差，进行牛顿迭代法，便可进行正解。

# 第三章 控制系统机械设计及实物制造

## 3.1 机械部分三维建模

### 3.1.1 机械模型设计目标

我们本次设计的六自由度平台旨在实现高精度的运动与测控功能，其主要目标是满足多自由度运动的需求，包括三个平移自由度，即X、Y、Z方向，以及三个旋转自由度，即俯仰、横滚、偏航方向。平台可广泛应用于机械控制实验、虚拟仿真测试、精密制造及其他多自由度运动研究领域。设计过程中，我们对机械性能、结构稳定性及制造成本等各项方面进行了综合的考虑。

六自由度平台的核心在于实现精准的多自由度运动。在实现平移运动时，平台需要在X、Y、Z三个方向完成独立且线性平移运动，每个方向的最大位移需要达到设计指标。在实现旋转运动时，平台需在俯仰、横滚和偏航三个方向完成角度调整，角度范围我们可以根据不同的应用场景进行设置。在实际运动中，平台各自由度的运动也需尽量解耦，减少多自由度运动之间的干扰，保证精确控制。

高精度和快速响应是六自由度平台的关键要求。每个平移方向的位移误差和旋转自由度的角度误差需控制在一定范围内以内确保满足精密测控的需求。平台需在运动过程中快速达到设定位置，同时在负载变化时保持性能稳定。为此，我们需优化控制算法和响应时间，使系统具备良好的动态跟随性。在多次运行后，平台的返回精度需保持在设定范围内，保证长期使用的可靠性。

同时平台的结构需优化设计，以均匀分布载荷，减少局部应力集中导致的结构变形。平台的长期稳定运行和可靠性同样至关重要。在运动过程中可能会产生振动和噪声，我们可以通过优化平台设计，如增加阻尼、选用高刚性材料，减少振动对测控结果的干扰。为便于后期维护与升级，采用模块化设计思路，使得平台的主要部件可以单独替换和维修。

在满足功能需求的前提下，我们平台的制造成本也需得到合理控制。我们优先选用轻质高强度材料，结合3D打印技术制作复杂结构件，进一步降低加工难度和成本。部分小型零件采用3D打印技术，以实现轻量化和快速成型。在成本方面，我们通过结构简化、标准件选用等方式减少材料浪费和制造复杂性，保证整体设计在性能与成本间达到平衡。

### 3.1.2 三维建模soildworks软件简介

在本次六自由度平台的设计中，我们选择了SolidWorks作为主要的三维建模工具。SolidWorks是一款功能强大、用户友好的三维CAD软件，尤其在零件设计和装配功能方面表现出卓越的优势，为复杂机械系统的建模与验证提供了全方位的支持。

#### 

#### 图3.1 Soildworks软件

SolidWorks在零件设计功能上的先进性极大提高了建模效率和设计精度。SolidWorks支持装配体的动态仿真功能，可以直观地模拟机械系统的运动过程，验证装配设计的合理性以及运动范围是否符合预期。在装配过程中，SolidWorks可以自动检测零件之间的干涉或碰撞问题，帮助设计者及时发现并解决潜在的设计错误。我们还可以通过质量分布、重心计算等工具，装配模块还能协助优化系统设计，使整体结构更加稳定和合理。

### 3.1.3 三维建模基本思路

在六自由度平台的三维建模过程中，我们遵循了模块化和系统化的设计原则，结合SolidWorks的强大功能，以逐步细化的方式完成了整个模型的搭建。建模的基本思路围绕平台的结构特点展开，主要包括上下平台、电动推杆、万向节连接机构、丝杆滑台等核心部分。

#### 

#### 图3.2 SW三维模型

首先，我们将整个平台划分为多个模块化单元，分别建模并逐步组合。这种方法便于独立优化每个模块的设计，同时提高了装配精度。设计的起点是上下平台部分，两个平台的设计重点在于尺寸的合理性与结构的稳定性。我们使用SolidWorks进行参数化建模，通过约束定义了平台的几何形状和安装孔位置，为后续部件的装配提供了参考基准。

其次是电动推杆部分的建模。电动推杆是实现六自由度运动的核心组件，由丝杆滑台、光杆导向件、上下万向节以及3D打印支撑件组成。我们首先为推杆的主要运动部件建模，通过SolidWorks的阵列工具生成丝杆和导轨的重复结构，保证推杆的平行度与精度。同时利用约束功能模拟推杆的线性运动和旋转自由度，为后续的装配仿真提供了动态验证条件。

在连接机构方面，上下万向节通过精确的球面和旋转关节连接上下平台与推杆，确保系统在运动中具备灵活性和可靠性。我们在建模时充分考虑了运动范围与材料强度的匹配，并通过装配仿真验证了各万向节的旋转角度是否满足设计需求，避免出现因关节过限而导致的运动干涉问题。

最后是整体装配的建模与验证。我们将各模块导入到装配环境中，利用SolidWorks的装配功能对零件之间施加几何约束，逐步完成六自由度平台的结构搭建。在此过程中，通过运动分析和干涉检查，确保了模型的合理性与可行性。同时，为便于后续实物制造，我们生成了详细的装配图和爆炸图，将平台的整体结构关系和各零部件的安装顺序清晰呈现。

## 3.2 丝杆滑台机构

### 3.2.1 丝杆滑台的组成部分

丝杆滑台是六自由度平台中实现精确线性运动的关键组件，其结构复杂且功能多样，主要由以下几个部分组成，如下图所示。

#### 

#### 图3.3 丝杆滑台构成

（1）光杆

光杆是丝杆滑台的重要导向部件，通过与滑台电机套及滑块延伸件的连接，形成了完整的推杆结构。光杆的主要功能是为滑块提供运动方向上的约束，确保推杆在运行过程中具有高精度的线性导向能力，同时减少横向摆动，保证整体运动的稳定性。

（2）滑块延伸件

滑块延伸件作为滑块与万向节之间的连接部件，结构分为上下两段。下段与滑块牢固连接，用以传递滑块的运动；上段通过与万向节的结合，将滑块的线性运动转换为推杆整体的伸缩动作。滑块延伸件的设计需具备较高的强度和稳定性，以承受推杆在工作时的力传递。

（3）滑台电机套

滑台电机套是整个丝杆滑台的安装和驱动部件，固定在微型步进电机的滑台框架上。电机套为滑块和光杆提供了支撑和固定功能，同时也是步进电机驱动力输出的关键载体。

（4）微型步进电机丝杆滑台

微型步进电机丝杆滑台是整个组件的核心部分，由二向四线步进电机、丝杆和滑块组成。步进电机通过驱动丝杆的旋转，带动滑块沿光杆实现精确的线性移动。丝杆的螺纹结构决定了滑块的行程与电机转动的角度转换关系，步进电机的二向四线结构则保证了其控制的高精度与响应性。

### 3.2.2 丝杆滑台的运动原理

丝杆滑台的运动原理基于步进电机驱动丝杆旋转，通过螺纹传动实现滑块沿光杆的直线运动，进而带动推杆完成伸缩。

首先我们驱动步进电机，丝杆滑台通过微型步进电机作为动力源，步进电机在接收到控制信号后，按照设定的角度逐步旋转。其二向四线结构允许精确控制旋转角度，从而实现滑块的微小位移调整。

步进电机的旋转通过电机轴传递至丝杆，丝杆因其螺纹结构将旋转运动转化为滑块的直线运动。螺纹的导程决定了滑块的位移量与丝杆旋转角度之间的线性关系，从而确保了滑块运动的高精度。

滑块在运动过程中，由光杆提供精确的直线导向。光杆限制了滑块的横向和旋转运动，确保其能够沿预定方向平稳移动，避免由于横向力而产生偏移。

滑块延伸键连接滑块与推杆的万向节，当滑块在光杆上移动时，通过滑块延伸件将直线运动传递至推杆，实现推杆的精确伸缩运动。这种设计能够将滑块的线性运动完整地转化为推杆的整体伸缩动作，为六自由度平台的运动提供必要的驱动力。

## 3.3 机械部分参数指标

机械部分参数指标如下表所示，由于平台连接座和万向节的缘故，该参数指标与六自由度平台的数学模型有所区别。

#### 表3.1 机械部分参数指标

|  |  |  |
| --- | --- | --- |
| 序号 | 参数指标 | 实物模型的数值 |
| 1 | 上平台半径 | 53.5mm |
| 2 | 下平台半径 | 100mm |
| 3 | 上平台铰点最短距离 | 67.15mm |
| 4 | 下平台铰点最短距离 | 38.14mm |
| 5 | 初始杆长 | 94.71mm |
| 6 | 初始高度 | 104.5mm |

## 3.4 实物制造与展示

实物展示如下图所示：

#### 

#### （a）六自由度平台实物展示1 （b）六自由度平台实物展示2

#### 图3.4实物展示

# 第四章 控制系统的硬件电路设计

## 4.1 硬件电路设计altiumdesigner 软件简介

Altium Designer 是原Protel 软件开发商Altium 公司推出的一体化的电子产品开发系统，主要运行在Windows 操作系统。这套软件通过把原理图设计、电路仿真、PCB 绘制编辑、拓扑逻辑自动布线、信号完整性分析和设计输出等技术的完美融合，为设计者提供了全新的设计解决方案，使设计者可以轻松进行设计，熟练使用这一软件使电路设计的质量和效率大大提高。

本系统采用上下双板，上板为MPU6050陀螺仪，下板为控制板，上板通过陀螺仪传感器进行反馈三轴角度，用SCL和SDA与下板进行通信。

## 4.2硬件电路详细设计

### 4.2.1 供电部分

单片机的最小系统包括电源电路，晶振（时钟）电路，复位电路，去耦电路。

鉴于本次设计需要的输入输出引脚数较多，采用stm32F103Z 系列芯片，其引脚数共144，完全可以满足本次设计的需求，具体的芯片型号为STM32F103ZET6。

1. 电源电路

采用LM2596S\_5.0/NOPBT对输入电压进行降压稳压。LM2596S\_5.0/NOPBT 是一款常用的 降压型稳压电源芯片，能够输出最大3A的驱动电流，同时具有很好的线性和负载调节特性。二极管D4则为一个防反接二极管，防止电流倒灌。同时使用XT30PW-M30.G.Y连接器，该连接器是一个适用于 30A 电流负载的小型、坚固的 DC 电源连接器，为电池和电机之间提供稳定、安全的电气连接。

#### 

#### 图4.2.1 5.0V固定电压和DC电源连接器原理图

由于STM32的工作电压（VDD）为2.0~3.6V,使用AMS1117芯片将5V电源电压转换为稳定的3.3V电压供STM32主电源VDD使用。采用一个 AMS1117 实现稳压的功能。AMS1117 是一个正向低压降稳压器，在1A电流下压降为1.7V，从而实现5V 到3.3V的转化。降压电路如图4.2.2所示。并且配置一个 LED 灯检测电源的输入情况，C41和C42 两个电容实现去耦功能。

#### 

#### 图4.2.2 稳压/降压电路原理图

### 4.2.2 晶振和复位部分

（1）晶振电路:

晶振是单片机的心脏，作为外部时钟源，向单片机发送时钟信号。STM32 有复杂的时钟源系统，所有引脚功能作用都需要时钟信号，而且不同的外设需要不同频率的时钟信号，时钟频率越高，单片机运行速度越快，功耗也就越大。为了避免浪费资源，节省功率，因此，有多种时钟源，分外部和内部，高速和低速。有HSE、HIS、LSI、LSE 四种时钟源.我们一般使用外部高速晶振电路HSE（High Speed Extra）.

不同型号单片机要求不同频率的晶振，以及不同容值的滤波电容.Stm32F103系列的芯片手册中对高速时钟源的要求如图4.2.3所示。

#### 

#### 图4.2.3 STM32F103ZET6 数据手册内容

基于手册中的要求，我们采用外部8MHz 晶振，和20pf 的滤波电容组成晶振电路，如图4.2.4 所示。

#### 

#### 图4.2.4 晶振电路原理图

(2)复位电路

使单片机从头执行代码称为复位，因为单片机程序不允许从程序中间开始。某次使用后断电，并不知道程序进行在哪个地方，故我们需要上电时复位，让程序从最开始运行。因此需要给RESET引脚输入复位信号（低电平）使它从代码第一行重新开始执行。

复位电路一般要满足上电复位和按键复位。上电复位是为了在上电启动时从头执行，按键复位是为了便于调试，避免重复开关供电造成损坏。复位引脚为RESET引脚，在引脚和按钮间加一个上拉电阻接VDD。当按键按下时，RESET和地连通，产生低电平，实现复位。电路图如图4.2.5 所示。

#### 

#### 图4.2.5 复位电路原理图

上电复位原理：在上电瞬间，由于电容来两端电压不能突变，RESET出现短暂低电平（逻辑电平为0），芯片自动复位，然后电容开始充电，从0 开始充电到3.3V，RESET端口电压也为3.3V（逻辑电平为1）。

充电时间由电阻和电容共同决定：

上电后约1ms左右系统完成复位，之后，单片机开始正常工作。

按键复位原理：当按下按钮时，同样因为电容两端电压不能突变，电容放电，RESET处放出拉电流，变为低电平状态，单片机进行复位。然后按键松开，电容充电，RESET处变成高电平。

上拉电阻与下拉电阻：上、下拉电阻本质还是电阻，只是因为位置和作用而特有的称呼，是一种用法。某信号网络接一个上拉电阻在接VCC，使其在空闲状态时电压为VCC（也就是图中的3.3V）。如复位电路图所示，如果没有上拉电阻，按下复位键时就相当于3.3V 直接接地，造成短路。某信号网络接一个下拉电阻再接GND，使其在空闲状态时电压为GND 电平。

(3)去耦电路

板子上有不同的负载（比如某个运放电路、MCU 的内核、MCU 的IO、ADC、时钟），每个负载都需要稳定供电，每个负载要正常工作，前提就是负载上的供电电压要稳，也就是尽可能干净的3.3V。但该负载内的器件们工作起来，都要动态地吸收电流，供电电压就会附加各种高频噪声。可以看成是在DC 3.3V 上“耦和”了由于器件工作带来的AC 噪声。

这样耦和了AC 的DC 供电电压不仅会影响本负载区域内的电路的工作，也会影响到其它连接在同一个VCC 上的其它负载的工作，可能导致负载的电路工作出现问题。

既然每个负载工作起来会导致其电源出现额外的波动，那就让波动在本地尽可能降低，且不影响到其它负载的工作。降低负载供应波动影响的方式就是加强能即时响应的供给，通过备用的供给平滑掉主供给快速反应方面的不足。电容的本性就是储能，用电容来做备用电能提供供给也就能平滑掉负载瞬间的需求带来的波动，保证该负载的电压尽可能稳定，也就是将有可能耦和到DC 上的AC 给去除掉，同时由于让本地的DC 稳定，降低了对其它负载的波及。

这样的电路消除掉了MCU 供电电压的高频噪声，故称之为去耦电路[9]，具体的电路图如图4.2.6 所示。

#### d8773f3a125427bc7b3ca23d5a3db99

#### 图4.2.6 去耦电路原理图

综上所述，MCU 部分的最小系统整体电路如图4.2.7所示。

#### 

#### 图4.2.7 最小系统电路原理图

### 4.2.3 外设部分

(1)调试和下载电路

下载电路就是单片机下载代码和调试的电路。一般有串口转换下载、JTAG和SWD 三种方式。串口转换下载需要额外的串口转换芯片，好处是可以很方便地通过电脑USB 连接下载，但是不能实时调试。

JTAG接口是STM32中常用的调试和编程接口，支持调试、编程、功能验证、硬件测试等多种关键任务。JTAG接口提供了对微控制器的低级别访问和强大的调试能力，帮助开发人员在开发、生产、测试、维护等阶段快速高效地完成工作。对于调试复杂程序和硬件故障定位，JTAG是不可或缺的工具。

STM32的JTAG接口包含几个重要的信号引脚，通常在芯片的外设引脚表中可以找到。以下是JTAG接口常用引脚：

（1）TDI（Test Data In）：用于输入数据到目标设备。

（2）TDO（Test Data Out）：用于从目标设备输出数据。

（3）TMS（Test Mode Select）：用于控制JTAG状态机。

（4）TCK（Test Clock）：用于时钟信号。

（5）TRST（Test Reset，部分芯片支持）：用于重置JTAG接口。

（6）RESET（系统复位，部分芯片支持）：用于系统复位（不是JTAG专用，但有时与JTAG一起使用）。

STM32的JTAG引脚一般在不同型号的MCU中有所不同。查阅TM32F103系列中的JTAG引脚定义如下：

（1）PA13 (TDI)（2）PA14 (TDO)

（3）PA15 (TMS)（4）PB3 (TCK)

（5）PB4 (TRST 通常可以通过配置来选择是否启用)

#### 

#### 图4.2.8 JTAG电路原理图

该种方式是最稳定的下载方式，速度可以很快，但是需要占用的体积较大。

(2)启动模式设置接口

其中BOOT0和BOOT1用于设置STM32的启动方式，其对应方式如图4.2.9所示：

#### 

#### 图4.2.9 BOOT对应启动模式

在实际设置中一般只用到用户闪存（Flash）启动，也就是主闪存flash 存储器被选为启动区域，所以令BOOT1 悬空。但本项目中BOOT1连接PB2，由单片机确定启动模式，如图4.2.10所示。

#### 

#### 图4.2.10 启动电路原理图

(3)USB转串口模块

使用USART1作为转换的串口，通过CH340C串口转换芯片，经行串口的转换。即这里USB插上电以后，就可以进行供电，还可以进行串口的调试（电脑要下载相应的串口驱动），其原理图如图4.2.11所示。

#### 

#### 图4.2.11 USB转串口原理图

(4)BEEP电路

蜂鸣器简单分为有源和无源两种，所谓的“源”不是电源，可以简单的理解为有无振荡源。我们使用的是有源蜂鸣器需要外加一定频率（2~5Khz）的驱动信号驱动。其原理图如图4.2.12所示。

#### 

#### 图4.2.12 BEEP电路原理图

R24：是一个限流电阻，防止输出电流过大损坏三极管。

R22：相当于基极的下拉电阻。在低电平时可以通过电阻将偏置电压拉低使截止完全可靠。

工作原理：BEEP输出低电平的时候偏置电压为0，三极管是截止状态，蜂鸣器不工作；BEEP输出高电平时，存在偏置电压，三极管导通，由于蜂鸣器正极加了电源，蜂鸣器正常工作，电流从集电极流向地。三极管在这里起到开关作用。

(5)LED电路

根据测控系统要求设计两个LED分别作为保护灯LED0和工作灯LED1。其电路原理图如图4.2.13所示。

#### 

#### 图4.2.13 LED电路原理图

(6)LED电路

根据测控系统要求设计三个按键分别为更换加减模式KEY0、开启按键KEY1、关闭按键KEY2，如图4.2.14所示。

#### 

#### 图4.2.14 KEY电路原理图

(7)排口电路

采用普通排针，并接出3.3V 和5V 的输出口用以驱动LED灯、蜂鸣器外设和步进电机，如图4.2.15所示。

#### 

#### 图4.2.15 3.3V和5V电路原理图

(8)二相四线步进电机微型驱动板

其输出电路采用双极型结构，上侧使用PNP晶体管，下侧使用NPN晶体管，尽管封装为微型封装，但仍能实现低饱和输出和低功耗特性。

#### 

#### 图4.2.17 驱动芯片方框图

这两款IC均可通过微控制器的信号直接控制电机。一款IC适用于采用4输入逻辑（IN1、IN2、IN3和IN4）的2相步进电机的1-2相激励驱动，而另一款IC则适用于采用3输入逻辑（ENA、IN1和IN2）的2相步进电机的2相激励驱动。驱动板与最小系统的电路连接如图4.2.18所示。

#### 

#### 图4.2.18 驱动板电路图

## 4.3 PCB 详细设计

### 4.3.1 核心板PCB设计概述

绘制完毕原理图之后整个控制板的逻辑十分清晰，下面要将各个元器件拟定封装，排布位置，连接线路，搭建接口，绘制PCB。鉴于控制板的原理较为简单，同时没有过多复杂的线路，所以采用两层板设计，如图4.3.1所示。

#### 

#### 图4.3.1 双层板原理示意图

为了保证PCB 板的整体性更好，布局时采用单面走线法，尽可能在PCB 顶面完成走线，尽可能保持底面完整性，并作为地平面。这样做可以尽可能减少过孔的数量，同时采用贴片型的元器件以确保整体性。除排针和晶振之外避免使用直插类型的元器件封装。同时需要设置布线规则，设置导线粗细以及安全距离，使PCB板焊接元件时可以避免短接或者干扰相邻元件。

### 4.3.2 供电部分PCB设计

按照供电部分的原理图，导入 PCB 当中。供电口采用 XT30 系列平铺插头，电阻统一采用0603封装，电容采用C1206 和C0603 封装（0603 的电容用在一些较小的位置），B340A 使用肖特基二极管。将电源部分放在整个板子靠上的部分，鉴于电源输入电压过大，有较大的烧毁元器件的风险，所以一是为了远离芯片防止烧坏芯片电路，二是为了便于接入24V外部供电。有关线路的布置，3.3V 线路布置在上层，总线路采用线宽0.6mm，一些直连元器件的线路采用线宽 0.25mm，所有 5V以及VCC的线路布置在底层，采用线宽为 0.8mm，以防止电流过大导致的局部温升。 5V以及VCC布置线路图如4.3.1所示：

#### 

**VCC**

**5V**

#### (a)全部图层视图

#### 

#### （b）底层VCC5以及VCC的PCB布线

#### 图4.3.1 VCC5以及VCC布线PCB

同时，由于电源部分需要承载的电流较大，所以在降压芯片附近额外铺铜，提升大电流的通过能力。即使如此也要注意防止 24V 电流直接全压启动防止电源部分冲入过大的浪涌电流烧毁元器件。具体的电源部分的线路布置如图4.3.2所示。

#### 

#### 图4.3.2 供电部分PCB

### 4.3.3 晶振和复位部分 PCB 设计

按照原理图，导入 PCB 当中。晶振采用 49S 的 8M 晶振。晶振电路属于高频电路，容易受到干扰，两条时钟线对称等长连接芯片引脚，且晶振工作区域周边禁止铺铜和走线。而且石英晶振由于内部结构问题比较脆，因此需要尽量放在板子内部靠近芯片，用外围电路围起远离板子边缘，以防碰撞破坏晶振。如图4.3.3所示。

#### 

#### (a)设置禁止区域 （b）晶振布线

#### 图4.3.3 晶振部分 PCB（黄色的部分均为丝印）

复位部分的按钮采用微动开关，布置在 3.3V 输出的附近，可以将之接入一整根 3.3V 的线路当中，避免线路过多或者过长，同时由于 NRST 引脚和晶振的20时钟线引脚距离非常近，所以复位的 NRST 线由过孔走底面接入芯片，如图 4.3.4所示。

#### 

#### 图4.3.4 RESET部分 PCB（黄色的部分均为丝印；蓝色为过孔）

### 4.3.4 PCB整体设计

首先先按照原理图分模块排列，利用布局传递，将原理图对应的元件传递到PCB中，按照元件模块化设计思想，单路布线以及差分对布线的方式，同时注意元件放置的安全距离，不同导线的间距，防止干扰。

#### 

#### 图4.3.5 整体 PCB（红色为顶层；蓝色为底层）

2D图以及3D图如下图所示：

#### 2D_STAWARED_XXB3D_STAWART_XXB

#### （a）2D图 （b）3D图

#### 图4.3.6 整体 PCB示意图

### 4.3.5 上板PCB整体设计

为了不干扰工作平台正常工作，将MPU6050设置为单独模块放置上平台，通过SCL与SDA与下板控制板进行连接。

#### 

#### （a）PCB布线图 （b）3D图

#### 图4.3.7 上板整体 PCB示意图

#### 

#### 图4.3.8 上板整体原理图

# 第五章 控制系统的程序设计和实现

## 5.1 程序设计软件简介及框架和逻辑结构

### 5.1.1程序设计软件简介

本系统的程序设计与开发基于 MDK-ARM 和 STM32CubeMX 两大开发工具，二者相辅相成，为嵌入式开发提供了高效、便捷的工作环境。

#### 

#### 图5.1 程序设计软件

MDK-ARM（Keil Microcontroller Development Kit for ARM）是一个集成开发环境（IDE），专为基于 ARM Cortex-M 核心的微控制器开发而设计。便于管理多文件、多模块的复杂项目，支持将项目划分为多个逻辑分组。内置 ARM 编译器，支持高效代码优化，生成占用空间小、运行速度快的二进制文件。提供强大的调试工具，如断点设置、变量监视、内存检查等，帮助开发者快速定位问题。支持多种实时操作系统（如 RTX RTOS），便于实现多任务管理和调度。支持通过 JTAG 或 SWD 接口与硬件目标板连接，进行在线调试和硬件仿真。

STM32CubeMX 是 STMicroelectronics 提供的一款图形化配置工具，用于简化 STM32 系列微控制器的初始化和外设配置。开发者可以通过直观的图形界面快速完成芯片的硬件配置，并生成初始化代码。提供芯片引脚视图，允许用户通过点击选择需要使用的引脚功能（如 GPIO、USART、ADC 等）。提供丰富的中间件支持，如 USB、TCP/IP 协议栈和文件系统，简化复杂功能的实现。提供可视化的时钟树设置工具，帮助用户直观地调整系统主频、外设时钟和定时器频率等参数。根据用户配置，自动生成 HAL（硬件抽象层）驱动程序代码，减少手动编写初始化代码的工作量。

STM32CubeMX 可以自动生成 MDK-ARM 项目文件，用户可以直接导入到 Keil 中，节省项目初始化时间。STM32CubeMX 的图形化配置界面有效避免了硬件资源冲突和配置错误，生成的 HAL 库代码稳定可靠，与 MDK-ARM 的开发流程无缝衔接。使用 STM32CubeMX 初始化的代码在 MDK-ARM 中可以方便地进行编译和调试，硬件配置与代码实现结合紧密，方便定位问题。

### 5.1.2 程序框架概述

在测控系统的程序框架以实现步进电机的实时位置控制为核心，结合姿态解算、目标位置计算、反馈控制和报警机制，同时支持人机交互功能。[10]下图是程序的具体实现和逻辑说明。

#### 

#### 图5.2.1 测控系统程序框架

程序的初始化部分位于main()函数的开头，主要完成硬件资源的配置。在程序启动时，通过调用HAL\_Init()完成HAL库的初始化，随后配置系统时钟SystemClock\_Config()，初始化GPIO端口MX\_GPIO\_Init()、串口MX\_USART1\_UART\_Init()和定时器MX\_TIM2\_Init()。例如，在MX\_GPIO\_Init()中，设置输入引脚用于读取传感器信号，设置输出引脚驱动LED报警灯或控制步进电机的方向。此外，姿态传感器的初始化通过调用特定的传感器驱动库函数Sensor\_Init()实现，使得传感器进入工作状态并准备好提供数据。初始化完成后，程序会启动串口接收中断，通过HAL\_UART\_Receive\_IT()持续接收用户指令，同时开启定时器HAL\_TIM\_Base\_Start\_IT()生成周期性中断，为系统提供精准的时间基准。

位置控制模块是系统的核心，通过函数PositionControl()实现。在该函数中，首先调用Sensor\_ReadData()从传感器中获取原始的加速度和角速度数据。这些数据通过CalculateEulerAngles()函数被解算为欧拉角，表示当前系统的姿态状态。然后，调用TargetPositionCalc()函数，计算目标位置和当前姿态之间的误差，生成控制信号作为步进电机的驱动指令。

最终，通过StepMotor\_Drive(control\_signal)将控制信号传递至步进电机，完成系统位置的精确调整。为了提高实时性，PositionControl()被频繁调用，尤其在主循环中，配合中断进行实时更新。

反馈与解算是程序中不可或缺的部分，用于实时更新系统状态。在Sensor\_ReadData()函数中，程序读取传感器采样的原始数据，包括三轴加速度和三轴角速度。随后，调用EulerAngleFusion()函数对这些数据进行滤波和融合，得到稳定且准确的欧拉角。这些欧拉角会通过全局变量实时更新系统的当前状态，用于后续的目标计算。目标位置解算逻辑封装在TargetPositionCalc()中，该函数通过计算当前姿态与目标姿态之间的偏差，得出需要调整的电机步数和运动方向，并通过更新target\_position和current\_position来完成反馈控制。

伺服报警与急停装置模块通过GPIO引脚的状态监控系统运行情况。在主循环中，调用CheckSystemStatus()函数，实时判断系统是否出现异常。当检测到传感器故障、电机过载或其他错误时，函数会设置标志位alarm\_flag为1，同时通过LED\_Alarm()函数点亮或闪烁报警灯，并调用EmergencyStop()立即停止所有运行中的电机。为确保报警信息可视化，程序通过HAL\_UART\_Transmit()将报警内容发送至上位机，以便操作员及时采取行动。

人机交互模块的核心功能是接收用户指令并反馈系统信息。程序通过配置串口中断，使HAL\_UART\_RxCpltCallback()函数在接收到用户数据后自动触发。在回调函数中，通过调用ProcessCommand()解析用户输入的指令。例如，当用户通过上位机发送指令时，ProcessCommand()会提取目标位置参数并更新target\_position。解析完成后，程序调用HAL\_UART\_Transmit()反馈成功消息，同时将新目标位置写入系统状态中。

欧拉角归零检测和目标位置调整功能是实现动态位置变化的关键。程序通过在主循环中定期调用UpdateTargetPosition()函数，检查当前欧拉角是否为零。如果检测到系统达到平衡状态，即欧拉角为零，则调用TargetPositionCalc()更新目标位置。例如，每当满足条件时，target\_position增加固定的步进值STEP\_INCREMENT，然后进入下一次位置调整流程。

### 5.1.3代码文件概述

本系统主要封装6个子系统文件夹，其中包含：Application/MDK-ARM、Application/User/Core、Drivers/STMF1xx\_HAL\_Driver、Drivers/CMSIS、MyCode、HARDWARE。如下图所示：

#### 

#### 图5.2.2 代码文件脉络

1. Application/MDK-ARM、Drivers/STMF1xx\_HAL\_Driver、Drivers/CMSIS为STM32CubeMX自动生成的代码，封装所需要的函数库。Application/User/Core中main.c文件集成整个测控系统的代码，包括调用各个功能以及函数，进行整个功能的实现。
2. MyCode文件夹中封装了IIC.c、inv\_mpu.c、inv\_mpu\_dmp\_motion\_driver.c、mpu6050.c。其中，MPU6050的通信方式为IIC，IIC.c中封装了定义IIC总线连接的GPIO端口,可任意改变SCL和SDA的引脚。inv\_mpu.c、inv\_mpu\_dmp\_motion\_driver.c进行DMP的移植读取三轴角度以及温度；mpu6050.c集成调用三轴角度、温度以及加速度等函数，需要读取数据直接再主函数调用即可。
3. HARDWARE文件夹中包含fytpi\_math.c，其封装了解算位姿需要的数学表达式包括三角函数、绝对值、平方根、限幅、斜坡等基础函数。light\_matrix.c封装了建立、求解矩阵的函数。step\_motor.c封装了步进电机驱动、伸长、缩短、暂停等函数。StewartMovement.c封装了逆解六自由度平台以及目标平台逆解的函数。

## 5.2 STM32CubeMX配置

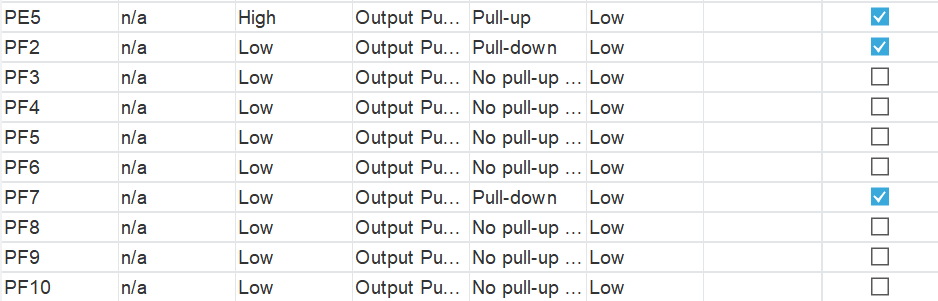
（1）MCU引脚配置布局

#### 

#### 图5.2.1 Pinout view

1. GPIO配置

#### 



#### 图5.2.2 GPIO配置

#### 表5.1 引脚配置表

|  |  |  |
| --- | --- | --- |
| 引脚配置以及功能 | | |
| PB5 | GPIO\_OUTPUT | PROTECT\_LED保护灯提示 |
| PE5 | GPIO\_OUTPUT | 工作灯提示 |
| PB6 | MPU6050\_SCL | 陀螺仪SCL口 |
| PB7 | MPU6050\_SDA | 陀螺仪SDA口 |
| PB8 | TIM4\_CH3 | 有源蜂鸣器 |
| PA9/PA10 | USART1 | 串口1 |
| PD0 | EXIT | K1 开启系统 |
| PD1 | EXIT | K2 急停系统 |
| PD2 | EXIT | K3 反转加减 |
| PA5;PC0;PC1;PC4;PF2;PF7 | GPIO\_OUTPUT(Pull-down) | 步进电机使能引脚 |
| PA7;PC5;PC2;PF8;PF5;PF3 | GPIO\_OUTPUT | 步进电机IN1 |
| PA6;PB0;PF10;PF9;PF6;PF4 | GPIO\_OUTPUT | 步进电机IN2 |

（3）定时器设置

TIM3作为定时器中断，。每10ms触发一次中断，用于控制两相四线步进电机时序，驱动电机旋转。

#### 

#### 图5.2.3 TIM3配置

TIM4\_CH3的PWM频率驱动蜂鸣器。

#### 

#### 图5.2.4 TIM4配置

（4）NVIC设置

EXTI line2 interrupt是急停键中断，优先级最高，设置为0。TIM3优先级最低，设置为2，其余优先级设置为1。

#### 

#### 图5.2.5 NVIC配置

## 5.3 系统库函数

### 5.3.1 MyCode文件

MPU-6000为全球首例整合性6轴运动处理组件，相较于多组件方案，免除了组合陀螺仪与加速器时之轴间差的问题，减少了大量的包装空间。MPU-6000整合了3轴陀螺仪、3轴加速器，并含可藉由第二个I2C端口连接其他厂牌之加速器、磁力传感器、或其他传感器的数位运动处理(DMP:Digital MotionProcessor)硬件加速引擎，由主要I2C端口以单一数据流的形式，向应用端输出完整的9轴融合演算技术。

MPU-6000的角速度全格感测范围为±250、 主500、±1000与±2000°/sec (dps)，可准确追緃快速与慢速动作,并且，用户可程式控制的加速器全格感测范围为±2g、±4g± 8g与± 16g。产品传输可透过最高至400kHz的I2C或最高达20MHz的SPI。

MPU6050是通过IIC来与MCU通信的，它有两个IIC接口，第一个是主IIC，通过SCL和SDA两条线与MCU通信；第二个辅助IIC通道，通过AUX\_CL和AUX\_DA连接外部从设备，比如磁传感器，这样就可以组成一个九轴传感器。VLOGIC 是 IO 口电压，该引脚最低可以到 1.8V，我们一般直接接 VDD 即可。AD0 是从 IIC 接口（接 MCU）的地址控制引脚，该引脚控制IIC 地址的最低位。如果接 GND，则 MPU6050 的 IIC 地址是：0X68，如果接 VDD，则是0X69。

1. IIC.c

定义IIC总线连接的GPIO端口, 只需要修改下面4行代码如图5.3.1，即可任意改变SCL和SDA的引脚;定义读写SCL和SDA的宏，已增加代码的可移植性和可阅读性。当SCL高电平时，SDA出现一个下跳沿表示IIC总线启动信号；当SCL高电平时，SDA出现一个上跳沿表示IIC总线停止信号。检测IIC总线设备，CPU向发送设备地址，然后读取设备应答来判断该设备是否存在，返回值 0 表示正确， 返回1表示未探测到。

#### 

#### 图5.3.1 IIC总线定义图

1. inv\_mpu\_dmp\_motion\_driver.c

DMP 是“数字运动处理器”（Digital Motion Processor）的缩写，通常与 MPU6050 陀螺仪和加速度计等传感器相关。DMP 的主要功能是处理和计算传感器数据，以提供更准确的运动和姿态信息。DMP 的主要特点：

数据融合：DMP 可以将来自多个传感器的数据（如加速度计和陀螺仪）进行融合，提供更稳定和准确的输出。实时处理：DMP 能够在硬件级别上实时处理数据，减轻主处理器的负担，提高系统的响应速度。姿态估计：通过 DMP，用户可以获得设备的姿态角（如滚转、俯仰和偏航角），这对于本系统十分重要。

1. inv\_mpu.c

驱动程序为传感器驱动程序层，此函数通过 I2C 与传感器通信的硬件驱动程序。基于 I2C 的对陀螺仪驱动，封装了陀螺仪姿态角解算、DMP初始化等功能。同时可以在头文件设置int mpu\_dmp\_get\_data(float \*pitch,float \*roll,float \*yaw)函数对三轴角度进行获取；MPU6050 陀螺仪的 DMP 初始化，使姿态角归零，因此设置uint8\_t mpu\_dmp\_init(void)函数对DMP进行初始化。

1. mpu6050.c

整体封装了初始化IIC通信、复位唤醒等功能，得到具体MPU陀螺仪三轴角度数值以及温度数值。通过下面函数进行初始化以及pitch，roll，yaw轴三轴欧拉角度获取。

uint8\_t mpu\_dmp\_init(void);

int mpu\_dmp\_get\_data(float \*pitch,float \*roll,float \*yaw);

#### 屏幕截图 2025-01-04 104138

#### 图5.3.2 MPU6050实时初始化图

### 5.3.2 HARDWARE文件

（1）fytpi\_math.c

包括三角函数、绝对值、平方根、限幅、斜坡等基础函数。快速计算三角函数，由第二章六自由度平台反解可知，六自由度平台属于一种并联机器人，因此需要运用DH参数，解算矩阵，根据三角函数周期性和对称性，只需取90个已知值就能计算出所有角度值。对于目标变量，最小值，最大值进行限幅，保证代码不进入死循环。同时，集成快速开平方根的函数。

1. light\_matrix.c

该文件封装矩阵运算基础所需要的函数，包括创建矩阵、矩阵加减法、矩阵求逆等函数，便于逆解六自由度平台时进行调用以及矩阵求解。

1. step\_motor.c

封装了步进电机驱动、伸长、缩短、暂停等函数。先对步进电机EN1、IN1、IN2进行引脚定义，构造步进电机结构体以便于直接指定引脚，对步进电机伸长、缩短函数进行定义。同时定义motor->foreward，用IFforeward传递参数，若IFforeward=1，则步进电机伸长，否则缩短。可直接在主函数中调用函数，确定步进电机伸长还是缩短。

#### 

#### 图5.3.2 step\_motor.h定义图

1. StewartMovement.c

由第二章可知，六自由度平台的结构参数主要有5个，分别是：上铰点分布圆半径 、上铰点分布角 、下铰点分布圆半径、下铰点分布角和平台处于初始位置时上下铰点的高度差H。

#### 表5.2 六自由度平台参数

|  |  |  |
| --- | --- | --- |
| 序号 | 参数指标 | 实物模型的数值 |
| 1 | 上铰点分布圆半径 | 53.5mm |
| 2 | 下铰点分布圆半径 | 100mm |
| 3 | 上铰点分布角 |  |
| 4 | 下铰点分布角 |  |
| 5 | 初始高度 | 104.5mm |

六自由度平台解算的基本思想是：平台初始零位时，坐标轴， ，与重合，当上平台位姿变化时，上平台即活动系位姿可用、 、、、、这6个参数来表达，其中为活动系原点的绝对坐标，、 、为上平台的姿态角，即分别为上平台法向量相对于下平台三个固定轴*Z、Y、 X*的转角。

#### 

#### 图5.3.3 六自由度平台坐标系图

初始状态时，六自由度平台上平台水平，所以其上平台活动位姿为：

（， ，，，，）=（0，0，0，，0，0）

活动系中任一点坐标与该点的绝对坐标的关系可按式 2-2计算。因此电动缸长度即为相对应上平台P点以及下平台B点的欧氏距离。当确定后，根据式2-1可求出转换矩阵H，则式2-5可求出点坐标，设各杆件的杆长初始值为，()为点的初始坐标， ()为点的坐标，可知*L0*为：

（5-1）

因此以下为部分代码：

#### 

#### 图5.3.4 六自由度平台H矩阵建立图

对于各个电动缸的长度，先计算下平台B点的绝对位置，由于规定下平台：

（， ，，）=（0，0，0，）

即只需通过下铰点分布角，运用三角函数知识即可解。上平台P点相对坐标可求，通过相对坐标以及B点绝对坐标可以求解P点绝对坐标，再运用式5-1即可求解出电动缸长度。

位置控制时，通过：void TR\_stewart\_movement(float alpha, float beta, float gama, float a, float b, float c, float \*L1, float \*L2, float \*L3, float \*L4, float \*L5, float \*L6)函数，

设置需要达到的目标位姿，归零时只需要让目标位姿初始化，即可寻零。

下图为StewartMovement.h规定的参数文件：

#### 

#### 图5.3.5 六自由度平台参数图

## 5.4 程序设计与解析

### 5.4.1原始变量与初始化

在测控系统的程序中，变量与初始化部分起到了奠定整个系统运行基础的作用。初始化是程序的第一步，其目的是将硬件和软件资源配置到工作状态，并对所有需要的全局变量赋予初始值。这部分内容将逐一展开，从变量定义到初始化过程进行详细说明，同时结合代码解析其功能和意义。

变量定义是程序设计的重要环节。根据程序的功能需求，我们将测控系统的变量可以分为以下几大类：

#### 

#### 图5.4.1 原始变量定义

（1）姿态相关变量

姿态解算是测控系统的核心任务之一，涉及到传感器数据的采集和处理。以下是与姿态相关的变量：

• gyro\_data：存储陀螺仪采样的三轴角速度数据，通常以浮点数表示，单位为°/s。这些数据通过传感器采集，用于计算系统的旋转角速度。

• accel\_data：存储加速度传感器采样的三轴加速度数据，单位为g，用于计算系统的倾斜角度。

• current\_euler\_angle：存储当前系统姿态的欧拉角信息，俯仰角Pitch、滚转角Roll和偏航角Yaw。该变量是通过解算角速度和加速度数据得到的，最终用于控制目标的姿态调整。

（2）位置控制相关变量

位置控制是通过步进电机实现的，需要定义多个变量进行状态跟踪和指令传递：

• current\_position：步进电机当前的位置，通常以步数表示。每个步数对应一个固定的电机转角。

• target\_position：目标位置，用于设定电机需要运动到的具体位置，通常通过用户输入或程序逻辑动态更新。

• control\_signal：控制信号，用于驱动电机的方向和步数。这是位置控制逻辑的输出结果，通过设置电机驱动器的信号引脚来实现控制。

• eul\_angle：用于寻零以及初始化，位姿控制时希望从工作初始位置进行目标位置拟合，只需要在初始位置上通过对角度、三轴进行各个变量加减，即可改变目标位置，达到姿态控制。因此，对结构体eul\_angle进行封装：定义：

{float target\_alpha, target\_beta, target\_gama, target\_a, target\_b, target\_c; }

（3）报警与状态标志

系统运行过程中可能出现异常情况，例如姿态超出安全范围、目标位置不可达等，这些情况需要通过报警标志记录：

• Begin：标志系统是否处于报警状态，1表示正常，0表示急停。

• mpu：标志系统是否完成初始化。此标志位将初始状态为1，只进行初始化一次，初始化完成后自动置为0。

（4）串口通信相关变量

串口通信是人机交互的重要手段，用户可以通过串口发送指令或接收状态反馈信息。相关变量包括：

• uart\_buffer：存储接收到的串口数据，用于解析用户指令。

• UART\_BUFFER\_SIZE：定义串口缓冲区的大小，避免因接收数据过多而导致缓冲区溢出。

初始化部分主要包括硬件资源初始化、全局变量赋值和外设配置。在STM32的嵌入式开发中，初始化通过调用相关的HAL库函数来实现，确保所有资源都处于正确的工作状态。系统启动时，首先调用HAL\_Init()函数初始化HAL库，该函数会配置系统中断优先级、初始化全局时钟以及复位相关外设状态。接着调用SystemClock\_Config()函数配置系统时钟，使芯片运行在预期的频率下。GPIO端口是连接外部设备的重要接口。在MX\_GPIO\_Init()函数中，根据系统需求设置了LED报警灯引脚为输出模式，用于指示系统状态；步进电机的控制引脚也被配置为输出模式；同时，传感器引脚被配置为输入模式，以接收数据信号。除了GPIO外，系统的串口和定时器也是关键外设。通过调用MX\_USART1\_UART\_Init()和MX\_TIM3\_Init()、MX\_TIM4\_Init()分别完成串口和定时器的初始化。

自定义初始化部分：需要对电机、串口中断、定位时基、MPU6050以及DMP进行初始化。

#### 

#### 图5.4.2 初始化

### 5.4.2主循环逻辑与代码

主循环是整个测控系统程序的核心部分，其主要作用是协调数据采集、姿态解算、位置控制、状态反馈以及外部指令处理等模块的执行，确保系统按照预期逻辑连续稳定地运行。它通过定时器中断的触发来标志各个模块的执行时机，并在主循环中完成具体的任务逻辑，以避免阻塞中断和提高系统的实时性。

具体步骤如下：

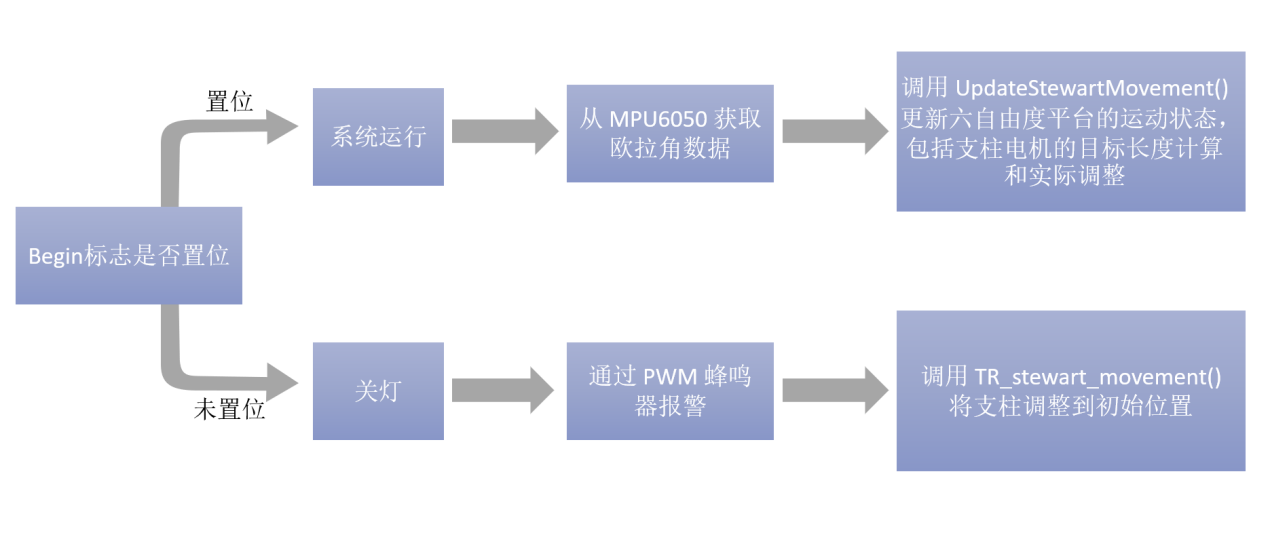
Step1: 开机，初始化系统，反馈六自由度平台所得三轴角度，将当前的三轴角度赋值初始角度，此后目标位置基于此加减；

Step2: 按下开启键，开机灯点亮，更新当前姿态；

Step3: 串口中断进行位置控制，同时按键可控制加减转换，同时串口实现人机交互；

Step4: 电动缸长度超程之后会进行伺服报警，打开保护灯；

Step5: 按下急停键，开机灯关闭，蜂鸣器报警。



#### 图5.4.3 主循环逻辑

（1）程序框架使用了一个无限循环 while(1)：在循环中，各种功能模块通过标志位的触发来有序运行。数据采集模块会通过mpu 标志位，当主循环检测到该标志位为1时，便会立即调用传感器读取函数，从陀螺仪和加速度计中采集pitch、roll、yaw三轴角度数据和温度数据。同样，姿态解算模块、位置控制模块、状态反馈模块和外部指令处理模块也遵循类似的逻辑，通过标志位的设置与清除来实现模块的协调工作。这种基于标志位的调度机制确保了各个功能模块能够按照一定的频率独立运行，同时避免了模块之间的资源竞争和冲突。

（2）在数据采集过程中，系统需要从陀螺仪两个传感器中获取实时数据，作为姿态解算的基础。由于数据采集是一个较为快速的过程，主循环通过调用函数：

mpu\_dmp\_get\_data(&pitch, &roll, &yaw)；

temp=MPU\_Get\_Temperature();

完成三轴欧拉角数据的采集以及工作系统温度数据采集。此函数利用 MPU6050 驱动库中的接口函数与传感器进行通信，读取到的三轴角度数据会存储到全局变量pitch、roll、yaw中，供后续的姿态解算使用。

姿态解算是主循环的另一个重要模块，负责将陀螺仪原始数据融合成欧拉角（Roll、Pitch 和 Yaw）。采用了互补滤波算法，利用陀螺仪的高频响应性和加速度计的低频稳定性，计算得到更准确的姿态角度。解算逻辑被封装在 CalculateEulerAngles 函数中，其中融合算法的实现需要对传感器数据进行矩阵运算和滤波，因此对计算性能有一定的要求。解算完成后，结果会存储到 current\_euler\_angle 中，供位置控制和状态反馈模块使用。

由于获取初始三轴角度，作为平台初始角度，stewart\_angle.target\_alpha=pitch;

stewart\_angle.target\_beta=roll;stewart\_angle.target\_gama=yaw，将陀螺仪获得的数据赋值到stewart\_angle中。

（3）位置控制模块通过比较目标位置与当前位置信息，生成步进电机的控制信号，从而驱动步进电机完成相应的位移调整。位置控制算法实现较为简单，采用了比例控制方式，根据当前位置与目标位置的差值生成正负信号，驱动步进电机向目标方向移动。该逻辑通过 PositionControlAlgorithm 函数实现，并通过 ExecuteMotorControl 函数执行具体的电机控制命令。在控制过程中，主循环不断监控当前位置与目标位置的偏差，当偏差小于0.1这一阈值时，位置控制逻辑会停止工作，步进电机已到达目标位置。

（4）状态反馈模块通过设置两个 LED 和串口向用户提供系统运行状态的信息。在发生报警或故障时，系统会开启蜂鸣器提示用户注意，同时关闭LED灯；在正常运行时，LED 则保持开启状态。此外，设置1个LED灯兼有保护灯以及加减模式转换的功能。系统会通过串口发送当前的位置信息和姿态角度，方便用户实时了解系统运行状态或进一步调整目标位置。反馈逻辑被封装在按键中断以及串口中断函数中，其中 LED 的控制通过 GPIO 操作完成，而串口反馈则基于 HAL 库的 UART 接口函数实现。

（5）最后，外部指令处理模块允许用户通过串口发送控制指令，例如设置新的目标位置或调整系统参数。主循环通过解析用户发送的指令字符串，提取其中的有效信息，并据此更新系统的运行参数。调用

stewart\_movement(&pitch, &roll, &yaw, stewart\_angle.target\_a, stewart\_angle.target\_b, stewart\_angle.target\_c, &L1, &L2, &L3, &L4, &L5, &L6)

更新参数，通过以上的循环逻辑，测控系统实现了各模块之间的协调工作，能够实时完成数据采集、姿态解算、位置控制、状态反馈以及外部指令处理的任务。

### 5.3.3中断函数逻辑与代码

在嵌入式测控系统中，中断函数是实现实时性的重要手段，能确保系统按既定频率执行任务并快速响应突发事件。中断函数可以脱离主程序的顺序执行流程，在特定硬件事件，如定时器溢出、外部信号触发或串口数据接收发生时独立执行。

（1）外部中断

外部中断的主要作用是对外部事件，如按键、传感器信号等进行快速响应。在本系统中，外部中断用来处理紧急事件和用户交互，例如复位系统、启动报警或其他紧急操作。外部中断通过 GPIO 端口的电平变化（上升沿或下降沿）触发，执行以下逻辑：

①检测触发的具体引脚，以区分不同的外部事件。

②根据引脚功能执行对应任务，如复位系统、记录报警状态或点亮指示灯。

③清除中断标志位，避免重复触发。

#### 

#### 图5.4.5 外部中断

如图所示，第一部分为开启系统 (KEY\_Begin\_Pin)：发送 “Begin” 到上位机，置位 Begin=1。第二部分为急停系统 (KEY\_Stop\_Pin)：发送 “Emergency”，将 Begin=0。第三部分为切换加减模式 (KEY\_UP\_Pin)：反转方向增减，点亮提示灯。

（2）定时器中断

定时器中断是系统的核心调度机制，负责触发各功能模块的执行。通过设置定时器的溢出周期，系统可以以精准的时间间隔执行特定任务。在本系统中，定时器设置为每 1 毫秒触发一次中断，通过累积时间计数器 system\_tick，可以实现多种不同周期任务的并行调度。

每次触发定时器中断时，程序会执行以下步骤：

① 增加系统时间计数器 system\_tick，以记录程序运行时间。

② 使用计数器的余数运算判断是否到达特定时间点（ 10ms），并设置对应的标志位。标志位的设定允许主程序在适当时机执行相应的功能模块，驱动step\_motor.c中电机旋转，同时用step\_tran反馈其正转还是反转。通过定时器3定时，驱动电机旋转：

motor\_move(&step\_motor\_1, step\_tran);

③ 定时器中断完成后自动返回主循环，保持主程序的正常执行。

#### 

#### 图5.4.6 定时器中断

使用核心运动控制函数void UpdateStewartMovement(void)，核心功能是根据当前欧拉角（pitch、roll、yaw）计算各支柱的目标长度，通过:

absFloat(TR\_L1 - L1) > Allowable\_error

比较当前长度 *TR\_Lx* 和目标长度 *Lx*，Allowable\_error设置为0.1，决定每个步进电机的运行方向，接着调用 motor\_change() 调整方向，motor\_stop() 停止电机。

#### 

#### 图5.4.7 核心运动控制函数

（3）串口中断

串口中断用于接收外部设备发送的控制指令，并通过缓存机制避免数据丢失。串口中断触发后会执行以下任务：

① 从接收寄存器中读取数据字节，并存入缓冲区 uart\_buffer。

② 检查接收的数据是否来源于串口1，并进行判断，对应参数增加或者减少。

③ 调用TR\_stewart\_movement函数，对目标值运动学解算，串口打印对应参数。

④ 对解算得到的电动缸长度与保护长度进行比较，同时将结算得到的长度复制保存，若超程则进行保护措施以及串口“Error”警告，LED保护灯开启。

#### 表5.3 串口中断发送对应意义

|  |  |
| --- | --- |
| 发送字符及对应含义 | |
| A |  |
| B |  |
| G |  |
| X |  |
| Y |  |
| Z |  |

该图为串口调试界面，我们可以清楚地看到程序解算的过程，结果如图。

#### 

#### 图5.4.8 串口调试界面

同时进行解算各个电机的长度，将长度与最长长度以及最短长度对比，如果超过最长长度或者低于最短长度，则开启保护LED灯，将上一值保留，不做加减。

# 参考文献

1. 刘晓昕.Stewart平台的MATLAB集成工具箱设计[D].哈尔滨工业大学,2007.
2. 颜旗权. 6自由度飞行模拟器的研究[D].哈尔滨工程大学,2008.
3. 高来福,江涛.六自由度并联操作平台系统设计[J].机械管理开发,2024,39(07):83-86+90.2024.07.030.
4. 黄晓敏.基于Matlab的六自由度平台研究[D].华南理工大学,2013.
5. 张尚盈,赵慧,韩俊伟.六自由度运动平台实时控制的正/反解算法[J].机床与液压,2003,03:133-135+123.
6. 刘延斌,韩秀英.一种Stewart平台位置正解的快速解法[J].矿山机械,2007,02:119-121.
7. Nguyen, C.C.; Zhou, Z-L; Antrazi, S.S.; Campbell, C.E., Jr., "Efficient computation of forward kinematics and Jacobian matrix of a Stewart platform-based manipulator," Southeastcon '91., IEEE Proceedings of , vol., no., pp.869,874 vol.2, 7-10 Apr 1991
8. 赵慧,张尚盈.Stewart平台雅可比矩阵分析[J].重庆科技学院学报(自然科学版),2007,03:142-144.
9. 周海波,徐蓓蓓.开关电路中去耦电容的选取与应用[J].第二届检验检疫电磁兼容及实验室建设学术研讨会论文集.2012.2012.03.27.
10. 胡雄,汪超,唐刚.基于STM32的六自由度平台的姿态测量系统设计[J].东华大学学报(自然科学版),2016,42(04):597-603.

# 结语

本次课程实践，我们进行了六自由度平台测控系统的设计与加工，我们从理论分析、方案设计到系统调试和实物展示，完成了一个系统性且具有挑战性的工程项目。

本项目以Stewart并联机器人的逆解理论为基础，结合运动学分析和牛顿迭代法，解决了平台运动精度与解算效率的难题；基于STM32的硬件电路设计与程序开发，我们实现了模块化封装、伺服报警、人机交互和急停保护等功能，为系统的稳定运行提供了强有力的保障；此外，通过SolidWorks对平台结构进行优化建模，并在实际制造中融入丝杆滑台机构，有效提升了系统的工艺性与经济性。

通过机械设计制造、硬件设计到PCB设计、再到算法功能以及实现，完全从0开始，不断优化迭代，解决工程难题，具备工程知识，考验自学能力。

从一个人装配到凌晨，再到独立学习硬件，从原理图开始到独立绘制PCB，从乱七八糟的布线，到学习掌握5V、3.3V以及GND线与其他线路粗细不同，模块化设计，如何铺铜，如何摆放元件位置，从分散的元件排布到如今还算紧凑的规划位置，三天内迭代三版PCB，收获颇丰。回想刚开始借用HI226却未从F407成功移植到F103中，还烧坏了HI226陀螺仪，到不放弃学会MPU6050开源如何移植到F103ZET6中，从看不懂的代码不断go to the definition看懂，再到自己成功设置读取三轴角度的函数，并且完成六轴的解算。从中期的三轴问题，到不断看文献理解六轴解算原理，再构建代码，完善系统设计。独自面对六个电机以及电驱的凌乱电路，不断出现的调试问题，锻炼我独当一面的能力。

感谢这次课设让我学会了勇于面对困难，不断解决问题的能力，更挖掘我的工程实践能力以及迅速自学潜质。工程是模块化的，但组合在一起仍然会出现各种问题，因此需要我们了解多方面知识，不能仅仅局限于本学科，实现真正学科交叉。通过此次实践，加强我对嵌入式设计、嵌入式系统的认识，也让我真正实现全栈开发设计，在日后工程问题解决时，更加理解如何设计、为何设计，以及好的设计如何为全系统锦上添花。

在设计过程中，我们深刻认识到理论与实践相结合的重要性。从初期的模型建立，到硬件电路的原理设计，再到最终的程序调试，每一步都需要理论分析和工程实践的紧密结合。特别是在系统调试阶段，我们多次遇到算法不稳定、电机运行异常等问题，但通过不断优化硬件布局、调整控制参数，以及改进程序逻辑，最终使系统达到了预期的设计要求。

该项目的完成不仅展示了六自由度平台的核心设计技术，也为该技术在虚拟仿真、精密制造、飞行模拟等领域的广泛应用提供了参考。同时，在整个设计过程中，我们全面提升了团队协作能力、工程实践能力和问题解决能力。本次课程设计为未来深入研究并联机器人和高精度测控系统奠定了扎实基础。

当然，项目也存在着一些不足。例如，在平台运行速度与负载能力的平衡上，还有进一步优化的空间；在程序设计方面，对于实时性的提升以及算法优化也有改进余地。这些都为后续研究工作提供了方向。

总体而言，本次六自由度平台测控系统的设计充分展示了跨学科、多领域融合的重要性，为我们提供了宝贵的实践经验和创新思路。未来，我们希望继续优化该系统，运用相关的经验和知识，将其拓展到更多应用场景，实现更大的工程价值和社会效益。

# 附录

## 附录A：MPU6050欧拉角解算代码MyCode

(1)mpu6050.c

#include "stm32f1xx\_hal.h"

#include "mpu6050.h"

//初始化MPU6050

//返回值:0,成功

uint8\_t MPU\_Init(void)

{

uint8\_t res;

MPU\_IIC\_Init();//初始化IIC总线

MPU\_Write\_Byte(MPU\_PWR\_MGMT1\_REG,0X80);

//复位MPU6050

delay\_ms(100);

MPU\_Write\_Byte(MPU\_PWR\_MGMT1\_REG,0X00);

//唤醒MPU6050

MPU\_Set\_Gyro\_Fsr(3); //陀螺仪传感器,±2000dps

MPU\_Set\_Accel\_Fsr(0); //加速度传感器,±2g

MPU\_Set\_Rate(50); //设置采样率50Hz

MPU\_Write\_Byte(MPU\_INT\_EN\_REG,0X00);//关闭所有中断

MPU\_Write\_Byte(MPU\_USER\_CTRL\_REG,0X00);

//I2C主模式关闭

MPU\_Write\_Byte(MPU\_FIFO\_EN\_REG,0X00); //关闭FIFO

MPU\_Write\_Byte(MPU\_INTBP\_CFG\_REG,0X80);

//INT引脚低电平有效

res=MPU\_Read\_Byte(MPU\_DEVICE\_ID\_REG);

if(res==MPU\_ADDR)//器件ID正确

{

MPU\_Write\_Byte(MPU\_PWR\_MGMT1\_REG,0X01)

MPU\_Write\_Byte(MPU\_PWR\_MGMT2\_REG,0X00);

MPU\_Set\_Rate(50); //设置采样率为50Hz

} else return 1;

return 0;

}

//设置MPU6050陀螺仪传感器满量程范围

//fsr:0,±250dps;1,±500dps;2,±1000dps;3,±2000dps

//返回值:0,设置成功

uint8\_t MPU\_Set\_Gyro\_Fsr(uint8\_t fsr)

{

return MPU\_Write\_Byte(MPU\_GYRO\_CFG\_REG,fsr<<3);

}

//设置MPU6050加速度传感器满量程范围

//fsr:0,±2g;1,±4g;2,±8g;3,±16g

//返回值:0,设置成功

uint8\_t MPU\_Set\_Accel\_Fsr(uint8\_t fsr)

{

return MPU\_Write\_Byte(MPU\_ACCEL\_CFG\_REG,fsr<<3);

}

//设置MPU6050的数字低通滤波器

//lpf:数字低通滤波频率(Hz)

//返回值:0,设置成功

uint8\_t MPU\_Set\_LPF(uint16\_t lpf)

{

uint8\_t data=0;

if(lpf>=188)data=1;

else if(lpf>=98)data=2;

else if(lpf>=42)data=3;

else if(lpf>=20)data=4;

else if(lpf>=10)data=5;

else data=6;

return MPU\_Write\_Byte(MPU\_CFG\_REG,data);

}

//设置MPU6050的采样率(假定Fs=1KHz)

//rate:4~1000(Hz)

//返回值:0,设置成功

// 其他,设置失败

uint8\_t MPU\_Set\_Rate(uint16\_t rate)

{

uint8\_t data;

if(rate>1000)rate=1000;

if(rate<4)rate=4;

data=1000/rate-1;

data=MPU\_Write\_Byte(MPU\_SAMPLE\_RATE\_REG,data);

return MPU\_Set\_LPF(rate/2); //自动设置LPF为采样率的一半

}

//得到温度值

//返回值:温度值(扩大了100倍)

short MPU\_Get\_Temperature(void)

{

uint8\_t buf[2];

short raw;

float temp;

MPU\_Read\_Len(MPU\_ADDR,MPU\_TEMP\_OUTH\_REG,2,buf);

raw=((uint16\_t)buf[0]<<8)|buf[1];

temp=36.53+((double)raw)/340;

return temp\*100;;

}

//得到陀螺仪值(原始值)

//gx,gy,gz:陀螺仪x,y,z轴的原始读数(带符号)

//返回值:0,成功

uint8\_t MPU\_Get\_Gyroscope(short \*gx,short \*gy,short \*gz)

{

uint8\_t buf[6],res;

res=MPU\_Read\_Len(MPU\_ADDR,MPU\_GYRO\_XOUTH\_REG,6,buf);

if(res==0)

{

\*gx=((uint16\_t)buf[0]<<8)|buf[1];

\*gy=((uint16\_t)buf[2]<<8)|buf[3];

\*gz=((uint16\_t)buf[4]<<8)|buf[5];

}

return res;;

}

//得到加速度值(原始值)

//gx,gy,gz:陀螺仪x,y,z轴的原始读数(带符号)

uint8\_t MPU\_Get\_Accelerometer(short \*ax,short \*ay,short \*az)

{

uint8\_t buf[6],res;

res=MPU\_Read\_Len(MPU\_ADDR,MPU\_ACCEL\_XOUTH\_REG,6,buf);

if(res==0)

{

\*ax=((uint16\_t)buf[0]<<8)|buf[1];

\*ay=((uint16\_t)buf[2]<<8)|buf[3];

\*az=((uint16\_t)buf[4]<<8)|buf[5];

}

return res;;

}

//IIC连续写 addr:器件地址 reg:寄存器地址 len:写入长度 buf:数据区

uint8\_t MPU\_Write\_Len(uint8\_t addr,uint8\_t reg,uint8\_t len,uint8\_t \*buf)

{

uint8\_t i;

MPU\_IIC\_Start();

MPU\_IIC\_Send\_Byte((addr<<1)|0);//发送器件地址+写命令

if(MPU\_IIC\_Wait\_Ack()) //等待应答

{

MPU\_IIC\_Stop();

return 1;

}

MPU\_IIC\_Send\_Byte(reg); //写寄存器地址

MPU\_IIC\_Wait\_Ack(); //等待应答

for(i=0; i<len; i++)

{

MPU\_IIC\_Send\_Byte(buf[i]);//发送数据

if(MPU\_IIC\_Wait\_Ack()) //等待ACK

{

MPU\_IIC\_Stop();

return 1;

}

}

MPU\_IIC\_Stop();

return 0;

}

//IIC连续读 addr:器件地址 reg:要读取的寄存器地址 len:要读取的长度 buf:读取到的数据存储区

//返回值:0,正常

uint8\_t MPU\_Read\_Len(uint8\_t addr,uint8\_t reg,uint8\_t len,uint8\_t \*buf)

{

MPU\_IIC\_Start();

MPU\_IIC\_Send\_Byte((addr<<1)|0);//发送器件地址+写命令

if(MPU\_IIC\_Wait\_Ack()) //等待应答

{

MPU\_IIC\_Stop();

return 1;

}

MPU\_IIC\_Send\_Byte(reg); //写寄存器地址

MPU\_IIC\_Wait\_Ack(); //等待应答

MPU\_IIC\_Start();

MPU\_IIC\_Send\_Byte((addr<<1)|1);//发送器件地址+读命令

MPU\_IIC\_Wait\_Ack(); //等待应答

while(len)

{

if(len==1)\*buf=MPU\_IIC\_Read\_Byte(0);//读数据,发送nACK

else \*buf=MPU\_IIC\_Read\_Byte(1); //读数据,发送ACK

len--;

buf++;

}

MPU\_IIC\_Stop(); //产生一个停止条件

return 0;

}

//IIC写一个字节

//reg:寄存器地址

//data:数据

//返回值:0,正常

uint8\_t MPU\_Write\_Byte(uint8\_t reg,uint8\_t data)

{

MPU\_IIC\_Start();

MPU\_IIC\_Send\_Byte((MPU\_ADDR<<1)|0);//发器件地址+写命令

if(MPU\_IIC\_Wait\_Ack()) //等待应答

{

MPU\_IIC\_Stop();

return 1;

}

MPU\_IIC\_Send\_Byte(reg); //写寄存器地址

MPU\_IIC\_Wait\_Ack(); //等待应答

MPU\_IIC\_Send\_Byte(data);//发送数据

if(MPU\_IIC\_Wait\_Ack()) //等待ACK

{

MPU\_IIC\_Stop();

return 1;

}

MPU\_IIC\_Stop();

return 0;

}

//IIC读一个字节

//reg:寄存器地址

//返回值:读到的数据

uint8\_t MPU\_Read\_Byte(uint8\_t reg)

{

uint8\_t res;

MPU\_IIC\_Start();

MPU\_IIC\_Send\_Byte((MPU\_ADDR<<1)|0);//发器件地址+写命令

MPU\_IIC\_Wait\_Ack(); //等待应答

MPU\_IIC\_Send\_Byte(reg); //写寄存器地址

MPU\_IIC\_Wait\_Ack(); //等待应答

MPU\_IIC\_Start();

MPU\_IIC\_Send\_Byte((MPU\_ADDR<<1)|1);//发器件地址+读命令

MPU\_IIC\_Wait\_Ack(); //等待应答

res=MPU\_IIC\_Read\_Byte(0);//读取数据,发送nACK

MPU\_IIC\_Stop(); //产生一个停止条件

return res;

}

(2)inv\_mpu.c

#include <stdio.h>

#include <stdint.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "inv\_mpu.h"

#include "inv\_mpu\_dmp\_motion\_driver.h"

#include "mpu6050.h"

#include "usart.h"

#define MPU6050//定义我们使用的传感器为MPU6050

#define MOTION\_DRIVER\_TARGET\_MSP430

//定义驱动部分,采用MSP430的驱动(移植到STM32F1)

#if defined MOTION\_DRIVER\_TARGET\_MSP430

#define i2c\_write MPU\_Write\_Len

#define i2c\_read MPU\_Read\_Len

#define get\_ms mget\_ms

#define log\_i printf //打印信息

#define log\_e printf //打印信息

#define fabs fabsf

#define min(a,b) ((a<b)?a:b)

#elif defined EMPL\_TARGET\_MSP430

#include "msp430.h"

#include "msp430\_i2c.h"

#include "msp430\_clock.h"

#include "msp430\_interrupt.h"

#include "log.h"

#define i2c\_write msp430\_i2c\_write

#define i2c\_read msp430\_i2c\_read

#define delay\_ms msp430\_delay\_ms

#define get\_ms msp430\_get\_clock\_ms

#ifdef MPU6500

data[0] = 0;

if (i2c\_write(st.hw->addr, st.reg->accel\_intel, 1, data))

goto lp\_int\_restore;

#endif

//q30格式,long转float时的除数.

#define q30 1073741824.0f

//陀螺仪方向设置

static signed char gyro\_orientation[9] =

{ 1, 0, 0,

0, 1, 0,

0, 0, 1

};

//MPU6050自测试

uint8\_t run\_self\_test(void)

{

int result;

//char test\_packet[4] = {0};

long gyro[3], accel[3];

result = mpu\_run\_self\_test(gyro, accel);

if (result == 0x3)

{

float sens;

unsigned short accel\_sens;

mpu\_get\_gyro\_sens(&sens);

gyro[0] = (long)(gyro[0] \* sens);

gyro[1] = (long)(gyro[1] \* sens);

gyro[2] = (long)(gyro[2] \* sens);

dmp\_set\_gyro\_bias(gyro);

mpu\_get\_accel\_sens(&accel\_sens);

accel[0] \*= accel\_sens;

accel[1] \*= accel\_sens;

accel[2] \*= accel\_sens;

dmp\_set\_accel\_bias(accel);

return 0;

} else return 1;

}

//陀螺仪方向控制

unsigned short inv\_orientation\_matrix\_to\_scalar(

const signed char \*mtx)

{

unsigned short scalar;

scalar = inv\_row\_2\_scale(mtx);

scalar |= inv\_row\_2\_scale(mtx + 3) << 3;

scalar |= inv\_row\_2\_scale(mtx + 6) << 6;

return scalar;

}

//方向转换

unsigned short inv\_row\_2\_scale(const signed char \*row)

{

unsigned short b;

if (row[0] > 0)

b = 0;

else if (row[0] < 0)

b = 4;

else if (row[1] > 0)

b = 1;

else if (row[1] < 0)

b = 5;

else if (row[2] > 0)

b = 2;

else if (row[2] < 0)

b = 6;

else

b = 7; // error

return b;

}

//mpu6050,dmp初始化

uint8\_t mpu\_dmp\_init(void)

{

uint8\_t res=0;

MPU\_IIC\_Init(); //初始化IIC总线

if(mpu\_init()==0) //初始化MPU6050

{

res=mpu\_set\_sensors(INV\_XYZ\_GYRO|INV\_XYZ\_ACCEL); if(res)return 1;

res=mpu\_configure\_fifo(INV\_XYZ\_GYRO|INV\_XYZ\_ACCEL); if(res)return 2;

res=mpu\_set\_sample\_rate(DEFAULT\_MPU\_HZ);//设置采样率

if(res)return 3;

res=dmp\_load\_motion\_driver\_firmware(); //加载dmp固件

if(res)return 4; res=dmp\_set\_orientation(inv\_orientation\_matrix\_to\_scalar(gyro\_orientation));//设置陀螺仪方向

if(res)return 5; res=dmp\_enable\_feature(DMP\_FEATURE\_6X\_LP\_QUAT|DMP\_FEATURE\_TAP| //设置dmp功能 DMP\_FEATURE\_ANDROID\_ORIENT|DMP\_FEATURE\_SEND\_RAW\_ACCEL|DMP\_FEATURE\_SEND\_CAL\_GYRO|

DMP\_FEATURE\_GYRO\_CAL);

if(res)return 6;

res=dmp\_set\_fifo\_rate(DEFAULT\_MPU\_HZ); if(res)return 7;

res=run\_self\_test(); //自检

if(res)return 8;

res=mpu\_set\_dmp\_state(1); //使能DMP

if(res)return 9;

} else return 10;

return 0;

}

//得到dmp处理后的数据(本函数需要比较多堆栈,局部变量有点多)

//pitch:俯仰角 精度:0.1° 范围:-90.0° <---> +90.0°

//roll:横滚角 精度:0.1° 范围:-180.0°<---> +180.0°

//yaw:航向角 精度:0.1° 范围:-180.0°<---> +180.0°

float mpu\_pitch=0.0f;

float mpu\_roll=0.0f;

float mpu\_yaw=0.0f;

int mpu\_dmp\_get\_data(float \*pitch,float \*roll,float \*yaw)

{

float q0=1.0f,q1=0.0f,q2=0.0f,q3=0.0f;

unsigned long sensor\_timestamp;

short gyro[3], accel[3], sensors;

unsigned char more;

long quat[4];

if(dmp\_read\_fifo(gyro, accel, quat, &sensor\_timestamp, &sensors,&more))return 1;

if(sensors&INV\_WXYZ\_QUAT)

{

q0 = quat[0] / q30; //q30格式转换为浮点数

q1 = quat[1] / q30;

q2 = quat[2] / q30;

q3 = quat[3] / q30;

//计算得到俯仰角/横滚角/航向角

mpu\_pitch = asin(-2 \* q1 \* q3 + 2 \* q0\* q2)\* 57.3; // pitch

mpu\_roll = atan2(2 \* q2 \* q3 + 2 \* q0 \* q1, -2 \* q1 \* q1 - 2 \* q2\* q2 + 1)\* 57.3; // roll

mpu\_yaw = atan2(2\*(q1\*q2 + q0\*q3),q0\*q0+q1\*q1-q2\*q2-q3\*q3) \* 57.3; //yaw

// 将数据传出

\*pitch = mpu\_pitch;

\*roll = mpu\_roll;

\*yaw = mpu\_yaw;

} else return 2;

return 0;}

## 附录B：平台位姿解算以及电机驱动代码HARDWARE

1. step\_motor.c

#include "step\_motor.h"

Motor step\_motor\_1 = {

GPIOA,

GPIO\_PIN\_5,

GPIOA,

GPIO\_PIN\_7,

GPIOA,

GPIO\_PIN\_6,

1};

Motor step\_motor\_2 = {

GPIOC,

GPIO\_PIN\_4,

GPIOC,

GPIO\_PIN\_5,

GPIOB,

GPIO\_PIN\_0,

1};

Motor step\_motor\_3 = {

GPIOC,

GPIO\_PIN\_1,

GPIOC,

GPIO\_PIN\_2,

GPIOF,

GPIO\_PIN\_10,

1};

Motor step\_motor\_4 = {

GPIOC,

GPIO\_PIN\_0,

GPIOF,

GPIO\_PIN\_8,

GPIOF,

GPIO\_PIN\_9,

1};

Motor step\_motor\_5 = {

GPIOF,

GPIO\_PIN\_7,

GPIOF,

GPIO\_PIN\_5,

GPIOF,

GPIO\_PIN\_6,

1};

Motor step\_motor\_6 = {

GPIOF,

GPIO\_PIN\_2,

GPIOF,

GPIO\_PIN\_3,

GPIOF,

GPIO\_PIN\_4,

1};

/\* 初始设置步进电机引脚电平 \*/

void motor\_Init(Motor\* motor)

{

HAL\_GPIO\_WritePin(motor->EnableGPIO, motor->EnablePin, GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(motor->EN1GPIO, motor->EN1Pin, GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(motor->EN2GPIO, motor->EN2Pin, GPIO\_PIN\_RESET);

}

/\* 改变电机状态 \*/

void motor\_change(Motor\* motor, int IFforeward)

{

HAL\_GPIO\_WritePin(motor->EnableGPIO, motor->EnablePin, GPIO\_PIN\_SET); // 打开电机使能引脚

motor->foreward = IFforeward;

}

/\* 驱动步进电机正反转函数 \*/

void motor\_move(Motor\* motor, uint8\_t step\_trasns)

{

if (motor->foreward == 1)// 伸长

{

if (step\_trasns == 1)

{

HAL\_GPIO\_TogglePin(motor->EN1GPIO, motor->EN1Pin);

}

else

{

HAL\_GPIO\_TogglePin(motor->EN2GPIO, motor->EN2Pin);

}

}

else// 缩短

{

if (step\_trasns == 1)

{

HAL\_GPIO\_TogglePin(motor->EN2GPIO, motor->EN2Pin);

}

else

{

HAL\_GPIO\_TogglePin(motor->EN1GPIO, motor->EN1Pin);

}

}

}

/\* 电机停止函数 \*/

void motor\_stop(Motor\* motor)

{

HAL\_GPIO\_WritePin(motor->EnableGPIO, motor->EnablePin, GPIO\_PIN\_RESET); // 关闭电机使能引脚

}

1. step\_motor.h

#ifndef \_\_STEP\_MOTOR\_\_

#define \_\_STEP\_MOTOR\_\_

#include <stdint.h>

#include <stm32f1xx\_hal.h>

extern uint16\_t step\_trasns;

typedef struct {

GPIO\_TypeDef\* EnableGPIO;

uint16\_t EnablePin;

GPIO\_TypeDef\* EN1GPIO;

uint16\_t EN1Pin;

GPIO\_TypeDef\* EN2GPIO;

uint16\_t EN2Pin;

int foreward;

}Motor;

extern Motor step\_motor\_1, step\_motor\_2, step\_motor\_3, step\_motor\_4, step\_motor\_5, step\_motor\_6;

void motor\_Init(Motor\* motor);

void motor\_change(Motor\* motor, int IFforeward);

void motor\_move(Motor\* motor, uint8\_t step\_trasns);

void motor\_stop(Motor\* motor);

#endif

1. StewartMovement.c

#include "StewartMovement.h"

#include "printf.h"

#include "light\_matrix.h"

#include "math.h"

#include "fytpi\_math.h"

#include "inv\_mpu.h"

#include "inv\_mpu\_dmp\_motion\_driver.h"

#include "mpu6050.h"

Mat H;

Mat B; // 下平台B点绝对位置

Mat p1; // 上平台p点相对位置

Mat p2; // 计算p点绝对位置

float valB[4], valp1[4], valp2[4];

float alpha , beta, gama;

//float a = 104.5, b = 0 , c = 0;

float a , b , c ;

float L1, L2, L3, L4, L5, L6;

float TR\_L1, TR\_L2, TR\_L3, TR\_L4, TR\_L5, TR\_L6;

//int StewartCount = 0;

// stewart平台运动解算

void stewart\_movement(float \*pitch, float \*roll, float \*yaw, float a, float b, float c, float \*L1, float \*L2, float \*L3, float \*L4, float \*L5, float \*L6)

{

alpha = \*pitch, beta = \*roll, gama = \*yaw;

a = 104.5, b = 0 , c = 0;

printf(" alpha: %.4f\t", alpha);

printf(" beta: %.4f\t ", beta);

printf(" gama: %.4f\t\r\n", gama);

float valH[] =

{

cos(alpha)\*cos(beta), cos(alpha)\*sin(beta)\*sin(gama)-sin(alpha)\*cos(gama), cos(alpha)\*sin(beta)\*cos(gama)+sin(alpha)\*sin(gama), a,

sin(alpha)\*cos(beta), sin(alpha)\*sin(beta)\*sin(gama)+cos(alpha)\*cos(gama), sin(alpha)\*sin(beta)\*cos(gama)-cos(alpha)\*sin(gama), b,

-sin(beta), cos(beta)\*sin(gama), cos(beta)\*cos(gama), c,

0, 0, 0, 1

};

MatCreate(&H, 4, 4);

MatSetVal(&H, valH);

/\*分别计算六杆长度\*/

// L1

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(ThetaB);

valB[2] = Rb \* cos(ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(60 - ThetaP);

valp1[2] = Rp \* cos(60 - ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

valp1[0] = 1;

valp1[1] = 1;

valp1[2] = 1;

valp1[3] = 1;

MatCreate(&p2, 4, 1);

MatSetVal(&p2, valp2);

MatMul(&H, &p1, &p2);

\*L1 = MatNorm(&B, &p2);

printf(" L1: %.4f\t", \*L1);

// L2

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(120 - ThetaB);

valB[2] = Rb \* cos(120 - ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(60 + ThetaP);

valp1[2] = Rp \* cos(60 + ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*L2 = MatNorm(&B, &p2);

printf(" L2: %.4f\t", \*L2);

// L3

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(120 + ThetaB);

valB[2] = Rb \* cos(120 + ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(180 - ThetaP);

valp1[2] = Rp \* cos(180 - ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*L3 = MatNorm(&B, &p2);

printf(" L3: %.4f\t", \*L3);

// L4

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(240 - ThetaB);

valB[2] = Rb \* cos(240 - ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(180 + ThetaP);

valp1[2] = Rp \* cos(180 + ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*L4 = MatNorm(&B, &p2);

printf(" L4: %.4f\t", \*L4);

// L5

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(240 + ThetaB);

valB[2] = Rb \* cos(240 + ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(300 - ThetaP);

valp1[2] = Rp \* cos(300 - ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*L5 = MatNorm(&B, &p2);

printf(" L5: %.4f\t", \*L5);

// L6

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(360 - ThetaB);

valB[2] = Rb \* cos(360 - ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(300 + ThetaP);

valp1[2] = Rp \* cos(300 + ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*L6 = MatNorm(&B, &p2);

printf(" L6: %.4f\t\r\n", \*L6);

}

/\* stewart平台目标位置运动解算 \*/

void TR\_stewart\_movement(float alpha, float beta, float gama, float a, float b, float c, float \*TR\_L1, float \*TR\_L2, float \*TR\_L3, float \*TR\_L4, float \*TR\_L5, float \*TR\_L6)

{

printf("TR\_alpha: %.4f\t", alpha);

printf("TR\_beta: %.4f\t ", beta);

printf("TR\_gama: %.4f\t\r\n", gama);

float valH[] =

{

cos(alpha)\*cos(beta), cos(alpha)\*sin(beta)\*sin(gama)-sin(alpha)\*cos(gama), cos(alpha)\*sin(beta)\*cos(gama)+sin(alpha)\*sin(gama), a,

sin(alpha)\*cos(beta), sin(alpha)\*sin(beta)\*sin(gama)+cos(alpha)\*cos(gama), sin(alpha)\*sin(beta)\*cos(gama)-cos(alpha)\*sin(gama), b,

-sin(beta), cos(beta)\*sin(gama), cos(beta)\*cos(gama), c,

0, 0, 0, 1

};

MatCreate(&H, 4, 4);

MatSetVal(&H, valH);

// 分别计算六杆长度

// L1

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(ThetaB);

valB[2] = Rb \* cos(ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(60 - ThetaP);

valp1[2] = Rp \* cos(60 - ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

valp1[0] = 1;

valp1[1] = 1;

valp1[2] = 1;

valp1[3] = 1;

MatCreate(&p2, 4, 1);

MatSetVal(&p2, valp2);

MatMul(&H, &p1, &p2);

\*TR\_L1 = MatNorm(&B, &p2);

printf("TR\_L1: %.4f\t", \*TR\_L1);

// L2

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(120 - ThetaB);

valB[2] = Rb \* cos(120 - ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(60 + ThetaP);

valp1[2] = Rp \* cos(60 + ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*TR\_L2 = MatNorm(&B, &p2);

printf("TR\_L2: %.4f\t", \*TR\_L2);

// L3

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(120 + ThetaB);

valB[2] = Rb \* cos(120 + ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(180 - ThetaP);

valp1[2] = Rp \* cos(180 - ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*TR\_L3 = MatNorm(&B, &p2);

printf("TR\_L3: %.4f\t", \*TR\_L3);

// L4

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(240 - ThetaB);

valB[2] = Rb \* cos(240 - ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(180 + ThetaP);

valp1[2] = Rp \* cos(180 + ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*TR\_L4 = MatNorm(&B, &p2);

printf("TR\_L4: %.4f\t", \*TR\_L4);

// L5

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(240 + ThetaB);

valB[2] = Rb \* cos(240 + ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(300 - ThetaP);

valp1[2] = Rp \* cos(300 - ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*TR\_L5 = MatNorm(&B, &p2);

printf("TR\_L5: %.4f\t", \*TR\_L5);

// L6

// 下平台B点绝对位置

valB[0] = 0;

valB[1] = Rb \* sin(360 - ThetaB);

valB[2] = Rb \* cos(360 - ThetaB);

valB[3] = 1;

MatCreate(&B, 4, 1);

MatSetVal(&B, valB);

// 上平台p点相对位置

valp1[0] = 0;

valp1[1] = Rp \* sin(300 + ThetaP);

valp1[2] = Rp \* cos(300 + ThetaP);

valp1[3] = 1;

MatCreate(&p1, 4, 1);

MatSetVal(&p1, valp1);

// 计算p点绝对位置

MatMul(&H, &p1, &p2);

\*TR\_L6 = MatNorm(&B, &p2);

printf("TR\_L6: %.4f\t\r\n", \*TR\_L6);

}

1. StewartMovement.h

#ifndef \_\_STEWARTMOVEMENT\_\_

#define \_\_STEWARTMOVEMENT\_\_

#include <inv\_mpu.h>

// 平台参数

static float Rp = 107 / 2;

static float Rb = 200 / 2;

static float ThetaP = 35;

static float ThetaB = 18;

// 保护长度

static float Protect\_More\_L = 200, Protect\_Less\_L = 100;

// 允许出现误差范围

static float Allowable\_error = 0.1;

// 平台状态参数

extern float alpha , beta , gama , a , b , c ;

// 初始化六支柱长度

extern float L1, L2, L3, L4, L5, L6;

// 初始化六支柱目标长度

extern float TR\_L1, TR\_L2, TR\_L3, TR\_L4, TR\_L5, TR\_L6;

void stewart\_movement(float \*pitch,float \*roll,float \*yaw, float a, float b, float c, float \*L1, float \*L2, float \*L3, float \*L4, float \*L5, float \*L6);

void TR\_stewart\_movement(float alpha, float beta, float gama, float a, float b, float c, float \*L1, float \*L2, float \*L3, float \*L4, float \*L5, float \*L6);

#endif

## 附录C：六自由度平台控制系统主函数

1. main.h

#ifndef \_\_MAIN\_H

#define \_\_MAIN\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

/\* Includes ------------------------------------------------------------------\*/

#include "stm32f1xx\_hal.h"

/\* USER CODE BEGIN ET \*/

typedef struct {

float target\_alpha, target\_beta, target\_gama, target\_a, target\_b, target\_c;

}eul\_angle;

/\* USER CODE END ET \*/

/\* Exported constants --------------------------------------------------------\*/

/\* USER CODE BEGIN EC \*/

/\* USER CODE END EC \*/

/\* Exported macro ------------------------------------------------------------\*/

/\* USER CODE BEGIN EM \*/

/\* USER CODE END EM \*/

/\* Exported functions prototypes ---------------------------------------------\*/

void Error\_Handler(void);

/\* USER CODE BEGIN EFP \*/

/\* USER CODE END EFP \*/

/\* Private defines -----------------------------------------------------------\*/

#define K3\_Pin GPIO\_PIN\_0

#define K3\_GPIO\_Port GPIOD

#define K3\_EXTI\_IRQn EXTI0\_IRQn

#define K2\_Pin GPIO\_PIN\_1

#define K2\_GPIO\_Port GPIOD

#define K1\_Pin GPIO\_PIN\_2

#define K1\_GPIO\_Port GPIOD

#define PROTECT\_LED\_Pin GPIO\_PIN\_5

#define PROTECT\_LED\_GPIO\_Port GPIOB

#define MPU6050\_SCL\_Pin GPIO\_PIN\_6

#define MPU6050\_SCL\_GPIO\_Port GPIOB

#define MPU6050\_SDA\_Pin GPIO\_PIN\_7

#define MPU6050\_SDA\_GPIO\_Port GPIOB

/\* USER CODE BEGIN Private defines \*/

extern eul\_angle stewart\_angle;

/\* USER CODE END Private defines \*/

#ifdef \_\_cplusplus

}

#endif

#endif /\* \_\_MAIN\_H \*/

1. main.c

#include "main.h"

#include "tim.h"

#include "usart.h"

#include "gpio.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

#include "stdio.h"

#include "IIC.h"

#include "inv\_mpu.h"

#include "inv\_mpu\_dmp\_motion\_driver.h"

#include "mpu6050.h"

#include "step\_motor.h"

#include "StewartMovement.h"

#include "printf.h"

#include "fytpi\_math.h"

/\* USER CODE BEGIN PV \*/

eul\_angle stewart\_angle = {0, 0, 0, 104.5, 0, 0};

float COPY\_L1, COPY\_L2, COPY\_L3, COPY\_L4, COPY\_L5, COPY\_L6;

int copy\_NUM = 0, add\_trans = 1;

uint8\_t step\_tran=1;

float pitch,roll,yaw; //欧拉角

short aacx,aacy,aacz; //加速度传感器原始数据

short gyrox,gyroy,gyroz; //陀螺仪原始数据

float temp;//温度

uint8\_t RxBuffer[50];

uint8\_t TxBuffer[50];

//uint8\_t BTx\_Buffer[] ="Begin";

//uint8\_t STx\_Buffer[] ="Emergency";

uint32\_t pwmValue = 300; // 按下按键实现蜂鸣器占空比30%

uint8\_t Begin = 2;

uint8\_t Replace = 0;

uint8\_t mpu=1;

uint8\_t StewartCount = 0;

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

void UpdateStewartMovement(void);

/\* USER CODE END 0 \*/

int main(void)

{

HAL\_Init();

SystemClock\_Config();

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_TIM4\_Init();

MX\_USART1\_UART\_Init();

MX\_TIM3\_Init();

/\* USER CODE BEGIN 2 \*/

MPU\_Init(); //MPU6050初始化

mpu\_dmp\_init(); //dmp初始化

/\* 电机初始化 \*/

motor\_Init(&step\_motor\_1);

motor\_Init(&step\_motor\_2);

motor\_Init(&step\_motor\_3);

motor\_Init(&step\_motor\_4);

motor\_Init(&step\_motor\_5);

motor\_Init(&step\_motor\_6);

motor\_change(&step\_motor\_1, 1);

motor\_change(&step\_motor\_2, 1);

motor\_change(&step\_motor\_3, 1);

motor\_change(&step\_motor\_4, 1);

motor\_change(&step\_motor\_5, 1);

motor\_change(&step\_motor\_6, 1);

HAL\_UART\_Receive\_IT(&huart1,(uint8\_t \*)&RxBuffer,1);

//开串口中断

HAL\_TIM\_PWM\_Start(&htim4, TIM\_CHANNEL\_3); //开PWM

HAL\_TIM\_Base\_Start\_IT(&htim3);

// 启动定时器并允许中断

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

if(mpu==1)//MPU初始化（只初始化一次）

{

mpu=0;

while(mpu\_dmp\_get\_data(&pitch, &roll, &yaw)); //必须要用while等待，才能读取成功

printf("Initialization successful\r\n");

temp=MPU\_Get\_Temperature(); //得到温度信息

printf("X:%.4f\t Y:%.4f\t Z:%.4f\t Operating temperature:%.2f\r\n",roll,pitch,yaw,temp/100);//串口1输出采集信息

HAL\_Delay(1000);

printf("---------------------------------------------------------------------------------------------------------------\r\n\r\n");

printf("Device ID: %-8d\r\n", StewartCount);

StewartCount++;

//在平台解算得出的初始值上加减

stewart\_angle.target\_alpha=pitch;

stewart\_angle.target\_beta=roll;

stewart\_angle.target\_gama=yaw;

TR\_stewart\_movement(stewart\_angle.target\_alpha, stewart\_angle.target\_beta, stewart\_angle.target\_gama, stewart\_angle.target\_a, stewart\_angle.target\_b, stewart\_angle.target\_c, &TR\_L1, &TR\_L2, &TR\_L3, &TR\_L4, &TR\_L5, &TR\_L6);

printf("---------------------------------------------------------------------------------------------------------------\r\n\r\n");

}

/\* 串口发开机命令 \*/

if(Begin==1)

{

printf("---------------------------------------------------------------------------------------------------------------\r\n");

printf("Begin\r\n");

HAL\_GPIO\_WritePin(GPIOE, GPIO\_PIN\_5, GPIO\_PIN\_RESET);//开启时开灯

HAL\_Delay(1000);

UpdateStewartMovement();

Begin=0;

}

else if(Begin==0)

{

printf("---------------------------------------------------------------------------------------------------------------\r\n");

printf("Emergency\r\n");

HAL\_GPIO\_WritePin(GPIOE, GPIO\_PIN\_5, GPIO\_PIN\_SET);//关灯

\_\_HAL\_TIM\_SET\_COMPARE(&htim4, TIM\_CHANNEL\_3, 3000);//打开蜂鸣器

HAL\_Delay(2000);

\_\_HAL\_TIM\_SET\_COMPARE(&htim4, TIM\_CHANNEL\_3, 0);//关闭蜂鸣器

/\* 寻零 \*/

eul\_angle stewart\_angle = {0, 0, 0, 104.5, 0, 0};

TR\_stewart\_movement(stewart\_angle.target\_alpha, stewart\_angle.target\_beta, stewart\_angle.target\_gama, stewart\_angle.target\_a, stewart\_angle.target\_b, stewart\_angle.target\_c, &TR\_L1, &TR\_L2, &TR\_L3, &TR\_L4, &TR\_L5, &TR\_L6);

Begin=2;

}

/\* 反转加减提示灯 \*/

if(Replace==1)

{

HAL\_GPIO\_TogglePin(GPIOB, GPIO\_PIN\_5);

HAL\_Delay(1000);

HAL\_GPIO\_TogglePin(GPIOB, GPIO\_PIN\_5);

Replace=0;

}

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

RCC\_OscInitStruct.OscillatorType

RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.HSEPredivValue = RCC\_HSE\_PREDIV\_DIV1;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL9;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

}

/\* USER CODE BEGIN 4 \*/

/\* 按键中断 \*/

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)

{

if(GPIO\_Pin==K1\_Pin)//开启系统

{

Begin=1;

}

if(GPIO\_Pin==K2\_Pin)//急停系统

{

Begin=0;

}

if(GPIO\_Pin==K3\_Pin)//反转加减

{

Replace = 1;//加减提示灯

if (copy\_NUM != 0)

{

if (add\_trans)

add\_trans = 0;

else

add\_trans = 1;

copy\_NUM = 0;

}

}

}

/\* 串口中断 \*/

void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart)

{

if (huart->Instance == USART1) // 检测串口 USART1

{

// 检查发送字母

if (RxBuffer[0] == 'A') //如果上位机发送A，则alpha角

{

if (add\_trans==1)

stewart\_angle.target\_alpha += 0.50;

else

stewart\_angle.target\_alpha -= 0.50;

copy\_NUM = 1;

}

else if (RxBuffer[0] == 'B') // 如果上位机发送B,则beta角

{

if (add\_trans==1)

stewart\_angle.target\_beta += 0.50;

else

stewart\_angle.target\_beta -= 0.50;

copy\_NUM = 2;

}

else if (RxBuffer[0] == 'G') // 如果上位机发送G,则gama角

{

if (add\_trans==1)

stewart\_angle.target\_gama += 0.50;

else

stewart\_angle.target\_gama -= 0.50;

copy\_NUM = 3;

}

else if (RxBuffer[0] == 'X') // 如果上位机发送X,则x轴平移

{

if (add\_trans==1)

stewart\_angle.target\_a += 1.50;

else

stewart\_angle.target\_a -= 1.50;

copy\_NUM = 4;

}

else if (RxBuffer[0] == 'Y') // 如果上位机发送Y,则y轴平移

{

if (add\_trans==1)

stewart\_angle.target\_b += 1.50;

else

stewart\_angle.target\_b -= 1.50;

copy\_NUM = 5;

}

else if (RxBuffer[0] == 'Z') // 如果上位机发送Z,则z轴平移

{

if (add\_trans==1)

stewart\_angle.target\_c += 1.50;

else

stewart\_angle.target\_c -= 1.50;

copy\_NUM = 6;

}

/\* 备份数据 \*/

COPY\_L1 = TR\_L1;

COPY\_L2 = TR\_L2;

COPY\_L3 = TR\_L3;

COPY\_L4 = TR\_L4;

COPY\_L5 = TR\_L5;

COPY\_L6 = TR\_L6;

printf("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\r\n\r\n");

// 目标长度解算

TR\_stewart\_movement(stewart\_angle.target\_alpha, stewart\_angle.target\_beta, stewart\_angle.target\_gama, stewart\_angle.target\_a, stewart\_angle.target\_b, stewart\_angle.target\_c, &TR\_L1, &TR\_L2, &TR\_L3, &TR\_L4, &TR\_L5, &TR\_L6);

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_5, GPIO\_PIN\_SET); // 关闭过长or短保护提示灯

// 过长or短保护

if (Protect\_More\_L < TR\_L1 || Protect\_More\_L < TR\_L2 || Protect\_More\_L < TR\_L3 || Protect\_More\_L < TR\_L4 || Protect\_More\_L < TR\_L5 || Protect\_More\_L < TR\_L6

|| Protect\_Less\_L > TR\_L1 || Protect\_Less\_L > TR\_L2 || Protect\_Less\_L > TR\_L3 || Protect\_Less\_L > TR\_L4 || Protect\_Less\_L > TR\_L5 || Protect\_Less\_L > TR\_L6)

{

printf("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\r\n");

printf("Error\r\n");

Begin=0;

TR\_L1 = COPY\_L1; // 恢复数据

TR\_L2 = COPY\_L2;

TR\_L3 = COPY\_L3;

TR\_L4 = COPY\_L4;

TR\_L5 = COPY\_L5;

TR\_L6 = COPY\_L6;

if (copy\_NUM == 1) // 恢复设定值

stewart\_angle.target\_alpha -= 0.50;

else if (copy\_NUM == 2)

stewart\_angle.target\_beta -= 0.50;

else if (copy\_NUM == 3)

stewart\_angle.target\_gama -= 0.50;

else if (copy\_NUM == 4)

stewart\_angle.target\_a -= 1.5;

else if (copy\_NUM == 5)

stewart\_angle.target\_b -= 1.5;

else if (copy\_NUM == 6)

stewart\_angle.target\_c -= 1.5;

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_5, GPIO\_PIN\_RESET); // 打开过长or短保护提示灯

}

// 继续接收

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t \*)&RxBuffer, 1);

}

}

/\* 定时器中断 \*/

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

if(htim->Instance == TIM3) //定时器3,定时时间10ms

{

motor\_move(&step\_motor\_1, step\_tran); // 驱动电机旋转

motor\_move(&step\_motor\_2, step\_tran);

motor\_move(&step\_motor\_3, step\_tran);

motor\_move(&step\_motor\_4, step\_tran);

motor\_move(&step\_motor\_5, step\_tran);

motor\_move(&step\_motor\_6, step\_tran);

if (step\_tran)

step\_tran = 0;

else

step\_tran = 1;

}

/\* USER CODE END Callback 0 \*/

}

/\* 更新六自由度平台各电机参数 \*/

void UpdateStewartMovement(void)

{

printf("---------------------------------------------------------------------------------------------------------------\r\n\r\n");

printf("Device ID: %-8d\r\n", StewartCount);

// 运动解算得到六支柱长度

stewart\_movement(&pitch, &roll, &yaw, a, b, c, &L1, &L2, &L3, &L4, &L5, &L6);

StewartCount++;

// 更新每根支柱的电机状态

// L1

if (absFloat(TR\_L1 - L1) > Allowable\_error)

{

if (TR\_L1 > L1)

motor\_change(&step\_motor\_1, 1);

else

motor\_change(&step\_motor\_1, 0);

}

else

motor\_stop(&step\_motor\_1);

// L2

if (absFloat(TR\_L2 - L2) > Allowable\_error)

{

if (TR\_L2 > L2)

motor\_change(&step\_motor\_2, 1);

else

motor\_change(&step\_motor\_2, 0);

}

else

motor\_stop(&step\_motor\_2);

// L3

if (absFloat(TR\_L3 - L3) > Allowable\_error)

{

if (TR\_L3 > L3)

motor\_change(&step\_motor\_3, 1);

else

motor\_change(&step\_motor\_3, 0);

}

else

motor\_stop(&step\_motor\_3);

// L4

if (absFloat(TR\_L4 - L4) > Allowable\_error)

{

if (TR\_L4 > L4)

motor\_change(&step\_motor\_4, 1);

else

motor\_change(&step\_motor\_4, 0);

}

else

motor\_stop(&step\_motor\_4);

// L5

if (absFloat(TR\_L5 - L5) > Allowable\_error)

{

if (TR\_L5 > L5)

motor\_change(&step\_motor\_5, 1);

else

motor\_change(&step\_motor\_5, 0);

}

else

motor\_stop(&step\_motor\_5);

// L6

if (absFloat(TR\_L6 - L6) > Allowable\_error)

{

if (TR\_L6 > L6)

motor\_change(&step\_motor\_6, 1);

else

motor\_change(&step\_motor\_6, 0);

}

else

motor\_stop(&step\_motor\_6);

}

/\* USER CODE END 4 \*/

void Error\_Handler(void)

{

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif