手势机器人设计与制作—控制程序设计

摘　　要

随着社会的进步和科技的发展，工业生产方式也得到了很大的提升。手势识别技术使得人们的双手免去了投入危险的作业环境之中的风险。提升了工业生产作业环境的安全性以及工业生产效率。本文主要研究设计了手势机器人的控制系统。它分别是由手势识别控制系统和机器人运动控制系统组成的。手势识别的控制主要通过人体佩戴数据手套来进行手势信息的识别采集，再由 STTM32 读取和解算，并利用定时器 TIM 生成 PWM 控制信号控制舵机驱动实现；机器人运动控制则由舵机驱动来实现。

关键词：手势识别、STM32、机器人

Gesture robot design and manufacture - control program design

Abstract

With the progress of society and the development of science and technology, the mode of industrial production has also been greatly improved. Gesture recognition technology allows people to keep their hands out of dangerous situations. It improves the safety and efficiency of industrial production environment. This paper mainly studies and designs the control system of gesture robot. It is composed of gesture recognition control system and robot motion control system respectively. Gesture recognition is mainly controlled by the human body wearing data gloves to identify and collect gesture information, which is then read and calculated by STTM32, and achieved by using timer TIM to generate PWM control signal to control the steering gear drive. Robot motion control is driven by the steering gear to achieve.

Keywords:STM32、Sensor、timer、gesture control

目录

1 绪论

1.1数据手套历史和现状

数据手套是实现虚拟与现实成功交互的硬件设施。其最先是由手套式的压力传感系统发展而成的。它最早是由 R.Sayre 等人开发的被称为 SayreGlove 的产品，旨在应用于系统的三维控制场合。在八十年代初期，得益于 T.Zimmerman 等人关于光弯曲传感手套的发展专利，数据手套领域取得了重大的发展。而数据手套的发展真正得到人们重视是在 1987 年之后。因为在这一年，“虚拟现实”这一专业术语才被 LannierJ 提出，并且将数据手套定义为实现人机交互功能的主要工具。

现如今，数据手套的应用非常广泛。其涉及到医疗修复、机械生产、手语翻译系统等。且在虚拟现实技术领域发挥了巨大的作用。涉及到的领域包括了模拟技术、技术训练、娱乐等、目前，市面上已经存在许多独立的数据手套产品。例如 5DT、CyberGlove 等等。图 1-1 为南非 5DT 公司研发的数据手套。

图 1-1 5DT 数据手套

1.2机械手的历史与现状

20 世纪中期，有关于机械手的研究在基于早期出现的古代机器人上悄然升起。并且在日后工业生产环境中，得以不断地发展。

1958 年，第一台机械手由美国联合控制公司研制出。1970 年，被后人熟知的斯坦福机械手第一次在斯坦福大学用计算机控制其运作。而将这一演示实现的是机器人学界中早期的革命家之一 Vivtor Schenman。70 年代之后，机械手在工业生产上得到了广泛的应用。人们通过机械手于其它生产设备的结合，大大地提高了生产率、产品质量、安全作业等。

随着计算机技术、微电子技术的飞速发展，机械手在工业生产上的发展也在不断地优化，追求更高精度、更加可靠和安全性。同时，机械手的应用也扩展到了生活中的方方面面。比如社火服务、海洋研究、石油开采、航天技术等等。图 1-2 为投入到生产线上的机械手。

图 1-2 scara 机械手

1.3手势识别技术的历史与现状

手势是人类除了语言表达之外，另一种表达方式。它能够在没有语言环境下进行交流。因此手势识别技术就有了其研究发展的意义。手势识别技术大致可以分为早期的接触式传感器技术和随着计算机技术发展的非接触式传感器。

基于接触式传感器的手势识别主要是通过使用多个传感器组合设计的数据手套。这类手势识别技术对人体的手部进行直接的检测，准确率高且虚拟技术体验感更加强烈。而基于非接触式传感器的手势识别则通常利用光学传感、雷达探测等技术。这些技术的工作流程可以分文以下三个步骤，如图 1-3 所示。

图 1-3 工作步骤

其中，手势的检测和相应图像的分割则是该系统地首要任务。而其检测的方法也有很多种。2010 年，清华大学沙亮等人在研究基于无标记全手势视觉的人机交互技术中提出了使用摄像头的车载手势视觉交互系统地解决方法。随后，微软公司公布了该摄像头可以借助红外线来识别手势运动。

1.4手势机械手的研究意义与目的

在日常生活中，人们总是徒手或者手持工具去抓取、移动以及其他复杂的任务。然而，在这其任务中，人们也将自己的双手置于危险之中，稍有不慎，就造成严重后果。

而仿生机械手的概念就是极力地模拟或者代替人体手部动作的机械手，来帮助人类安全有效地从事各种科研、生产活动等，满足人类的发展需求、保证人类活动安全。

2 手势机器人总体方案设计

2.1手势机器人的设计目标

根据人手利用数据手套控制机器人运动的需求，针对其特点和工作环境，设计出一个5自由度的手势机器人的控制系统。完善数据手套检测手势功能，机器人驱动软件和手势控制软件。从而实现人手利用数据手套控制手势机器人运动及实现物体的抓取动作。

2.2本设计应解决的问题

（1）数据手套的设计

（2）编写基于NET的上位机程序

（3） 编写基于单片机STM32控制舵机驱动的程序

2.3手势控制系统方案设计

手势控制系统设计的重要内容是数据手套的实现和机器人运动的实现。

数据手套实现的方式有很多种,按照其搭载的传感器数目可以分为5触点数据手套、14触点数据手套等。考虑到其设计成本、实现途径等因素，本次研究所用的数据手套是在5触点数据手套的基础上另外搭载一个姿态传感器组成的。当人手佩戴数据手套时，由电脑发送指令至单片机控制中心，软件开启手势信息采集工作。手套上的5个弯曲传感器检测人体手指的弯曲信息，单片机利用ADC功能对这些信息进行采集，利用HC05蓝牙模块将信息返回电脑；姿态传感器检测人体的手部运动信息，单片机利用IIC功能采集手部运动信息并利用HC05蓝牙模块将信息返回给电脑。

HC05蓝牙模块对比其它通信方式，其使用方法非常简单易懂。虽然其内部含有蓝牙通信协议，但是它在使用时并不需要去考虑这部分的内容，其方法与普通串口的使用方法并差别。图2-1为HC05蓝牙模块的实物。

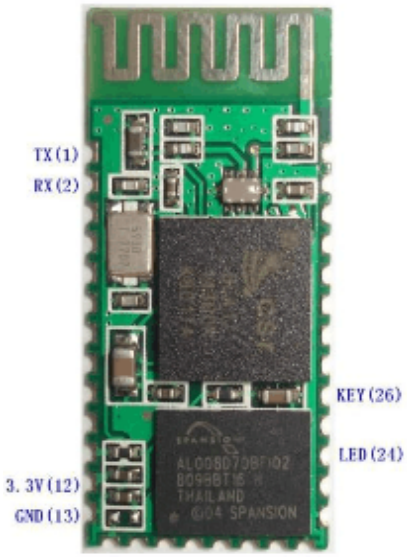

图 2-1 HC05 蓝牙模块

在选用弯曲传感器时，对比了各公司的传感器类型如表 2-1 所示，最终选定了由深圳麦格米特公司研发的柔性弯曲光纤传感器，实物如图 2-2 所示。

表 2-1 传感器类型性能比较

公司	传感器	体积	重量	价格	精度	安装	一致性	耐用性
VPL	柔性弯曲光纤传感器	小	轻	便宜	较低	简单	一般	一般
Vertex	电阻传感器	小	轻	便宜	较低	简单	一般	一般
Exos	摩尔传感器	较大	较重	偏贵	较高	较复杂	良好	一般
Megmeet	柔性弯曲光纤传感器	小	轻	便宜	较高	简单	良好	良好

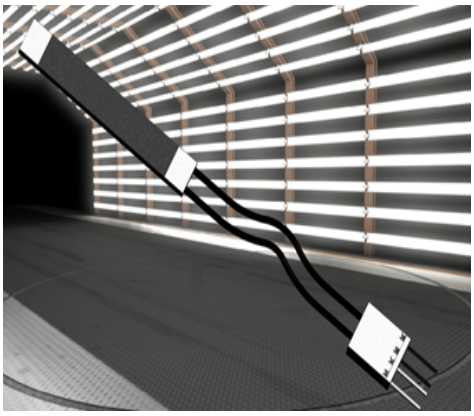

图 2-2 柔性弯曲光纤传感器

本文中，姿态传感器的选用的是 MPU6050 传感器。之所以选择这款传感器一是因为它是全球首款整合性 6 轴运动处理件；而是其与其他姿态传感器相比，它的检测结果免除了组合陀螺仪与加速器时之轴差的问题，同时它还减少了封装空间，更容易实现在手套上的配置。其传感器实物如图 2-3 所示，模块如图 2-4 所示。

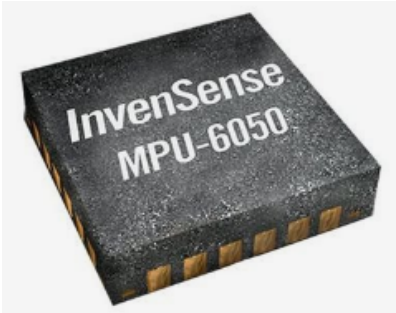

图 2-3 MPU6050 传感器

图 2-4 MPU6050 模块

机器人运动的实现分别由控制系统、驱动系统和机器人主体三部分组成。其控制关系如

图 2-5 所示。考虑到机器人手指和机器人手部的转动范围和受力大小，分别采用了 MG955 和 SG90 两种型号的舵机来驱动机器人手指运动和手部运动。其实物如图 2-6、图 2-7 所示。根据采集到的手势信息生成控制信号，控制舵机转动，从而实现控制机器人手指弯曲动作和机器人手部姿态动作。

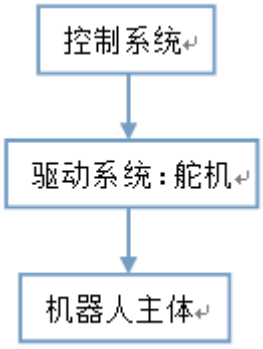

图 2-5 机器人运动控制关系

图 2-6 MG955 舵机

图 2-7 SG90 舵机

3 手势控制原理

手势控制实际上就是根据人体的手部信息来控制机器人运动的过程。要想实现这一过程，需要两部分内容，分别是：手势信息的获取以及机器人运动的控制。

3.1手势信息的获取

对于人体手指弯曲度信息的获取，本次研究是将表面涂有特殊电阻材料的弯曲传感器配置在数据手套上，这种特殊的电阻材料可以在人体手指运动时，使得传感器产生因应力变化而发生的弯曲变形。其表面的电阻大小也随着弯曲变形的大小成正比例关系变化。但由于弯曲传感器所采集到的数据为模拟信号，并非 STM32 单片机可读的数字信号。因此，需要设置 5 个 ADC 通道来将这 5 路模拟信号转换成 STM32 单片机可读的数字信号。通过 STM32 中文使用手册可知，STM32 单片机上的 ADC 为逐次逼近型的模拟转换器，需经过采用、保持、量化与编码四个步骤来将模拟信号转换成数字信号，其工作原理框图如图 3-1 所示。

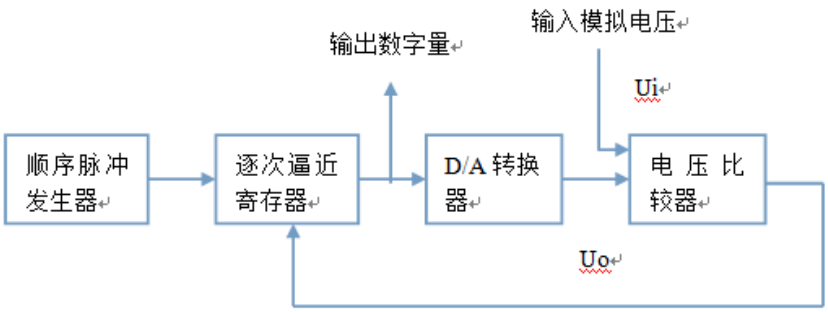

图 3-1 逐次逼近型模拟数据转换器工作原理框图

对于手部姿态信息的获取，本次研究通过在数据手套上配置姿态传感器 MPU6050 方式进行获取。表 3-1 为 MPU6050 传感器的内部资源及其功能介绍。

表 3-1 MPU6050 内部资源及其功能

MPU6050 内部资源	功能
三轴 MEMS 陀螺仪	检测物体转动的角速度
三轴 MEMS 加速计	检测物体运动的加速度
数据运动处理器 DMP	将检测的原始数据进行换算

普通的三轴陀螺仪的实物如图 3-2 所示。

如果在其三轴陀螺仪工作原理基础上，结合科里奥利力的应用，就可以得出本文所提及的三轴 MEMS 陀螺仪。所谓的科里奥利力，其实可以把它看作为就角速度，也正因此，三轴 MEMS 陀螺仪的角度可以通过其电容的变化计算。其关系如图 3-3 所示。

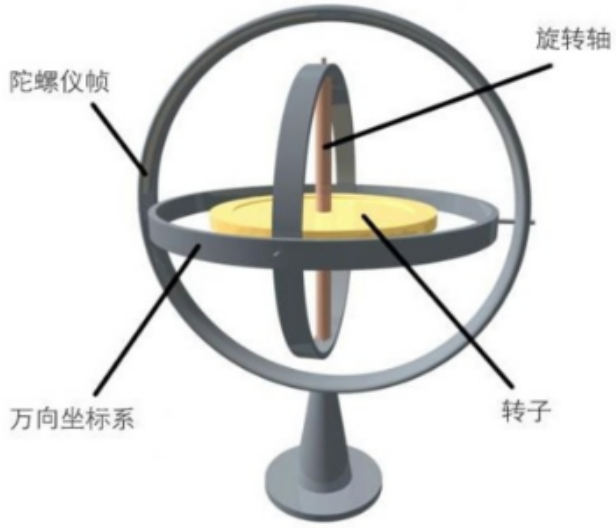

图 3-2 三轴陀螺仪

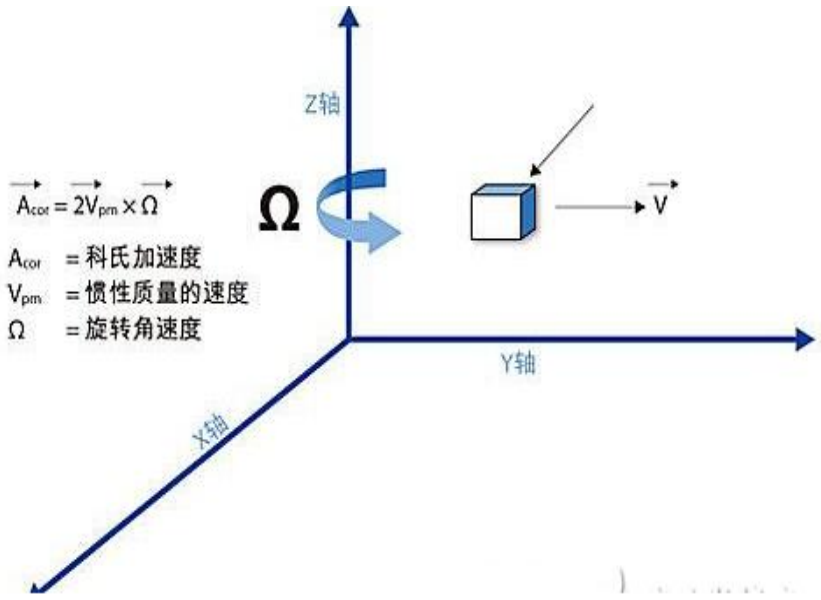

图 3-3 三轴角度与旋转速率的关系

而三轴 MEMS 加速度计就是在加速过程中，通过对人体手部所受到的惯性力的测量，利用牛顿第二定律获得加速度值。在静止状态下，加速度计三轴受力的合力与重力大小、

方向存在一定的关系，具体如图 3-4 所示。

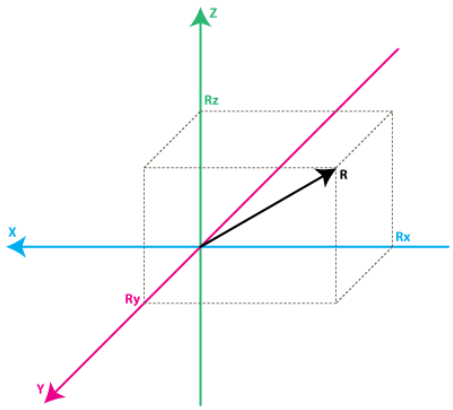

图 3-4 加速度计三轴合力

在测量到角速度和加速度后，利用 MPU 自带的数据处理器 DMP 对这些原始数据进行转换，得到所需的姿态数字信号。这时，STM32 单片机再利用 IIC 通信对此姿态数字信号进行读取。

至此，手势信息的获取工作已全部完成。

3.2机器人运动的控制

本次研究中，机器人的运动是通过驱动舵机实现的。因此，可以把机器人运动的控制理解成舵机的控制。

舵机实际上是一种角度（位置）伺服的驱动器，其内部组成如图 3-5 所示。它适用于角度不断变化并且可以保持的控制系统。本次研究中，利用 STM32 单片机内部的定时器 TIM1 和 TIM2，根据上述采集到的手势数字信息生成 PWM 信号。PWM 控制信号其实就是一个高低电平持续时间不断发生变化的电路，当这路进入到舵机时，它与舵机自身的基准电路的电压进行比较，并获得其电压输出差，从而实现对电机正反转的控制。其工作原理如图 3-6 所示。

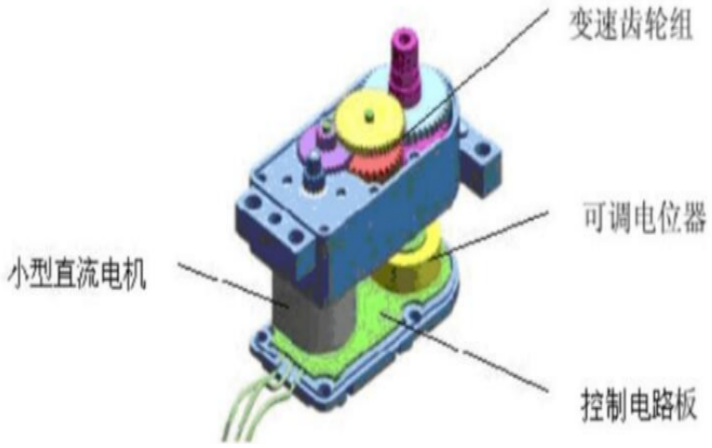

图 3-5 舵机的组成

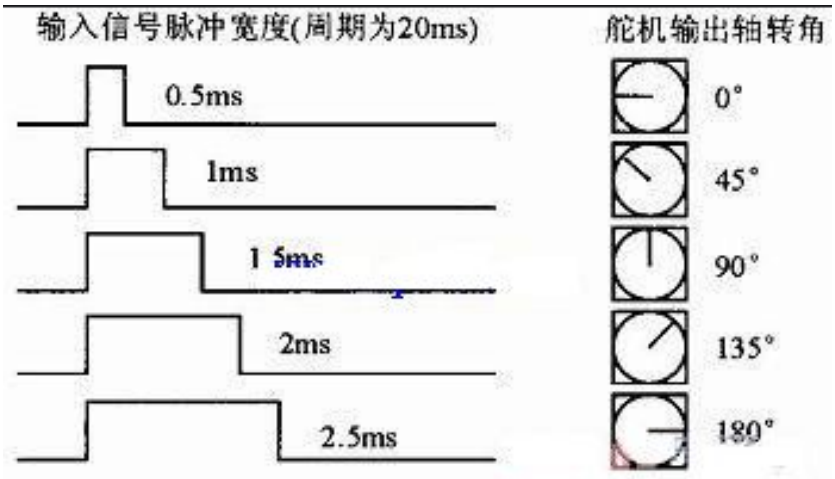

图 3-6 舵机工作原理图

4 系统软件设计准备工作

4.1 Keil uVision5 MDK

2006 年，ARM 公司收购了德国知名软件公司 Keil，并推出了基于 ARM7、ARM9、CORTEX-M 微处理器的开发软件 MDK。目前，MDK 也是开发 ARM 内核单片机的主流开发工具。它提供了包括 C 编译器、宏汇编、连接器、库管理和一个功能强大的仿真调试器在内的完整开发方案，通过一个集成开发环境（uVision）将这些功能组合在一起。此外，MDK 还适合不同层次的开发者使用，包括了刚入门级别的。因此，我们选择了 Keil uVision4MDK 作为我们 stm32 的软件开发平台。其初始界面如图 4-1 所示。

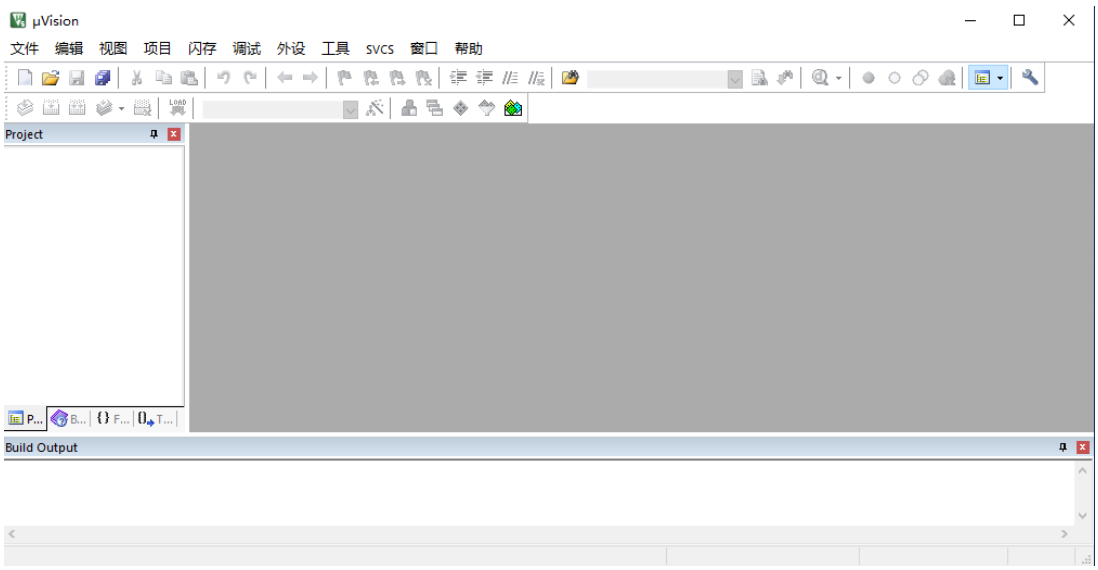

图 4-1 Keil uVision5 MDK 初始界面

4.2 Visual Studio

由美国微软公司推出的 Visual Studio 应用软件，简称为 VS，实际上是一个开发集成系统即常说的 IDE。这款软件中，涵盖了一个基本完整的开发工具集，它包括了如 UML 工具、代码管控工具、集成开发环境等整个软件生命周期中所需要的大部分工具。该软件适用的开发语言非常多，表 4-1 中列举出了其支持的部分开发语言。 图 4-2 为 VS 初始界面。

表 4-1 VS 适用的开发语言

C 语言
C++
C#
JAVA

图 4-2 VS 初始界面

4.3 MPU6050 传感器软件设计准备

MPU6050 传感器工作原理如图 4-3 所示，通过软件设置，可以直接得到由其内部三轴陀螺仪测量人手在空间中的角度原始数据和加速度计测量的加速度原始数据。得到原始数据后，还需进行解算，才能得出研究所需的姿态数据即欧拉角。原始数据的解算有两种方法：一种是调用 MPU 自身配置的 DMP 即数字运动处理器，对原始数据直接进行四元数转换，再对其利用公式计算得出所需的欧拉角；另一种则是对其检测到的原始数据进行一些姿态融合解算（卡尔曼、积分运算等），以求出当前的欧拉角。对比两种方法，明显发现第二种比第一种复杂，所以选择第一种方来获取欧拉角。

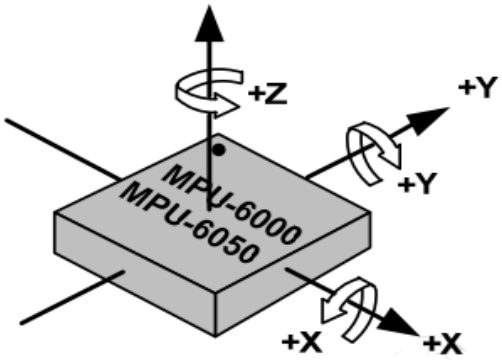

图 4-3 加速度计和陀螺仪的原理

4.3.1 DMP 数字运动处理器

DMP 的意义在于读取 MPU 加速度计和陀螺仪检测的原始数据并进行四元组数据转换。转换后的四元组数据的浮点数是放大了 2 的 30 次方倍数的，即采用了 q20 格式。为了获得正确的浮点数，方便之后计算欧拉角，将得到的四元组数据除以 2 的 20 次方。公式如下所示。

q0=quat[0]/q30 (式 4-1)

q1=quat[1]/q30 (式 4-2)

q2=quat[2]/q30 (式 4-3)

q3=quat[3]/q30 (式 4-4)

其中 quant[0]~quat[3]为四元组数。在求出浮点数之后，就可以根据以下公式得出我们需要的欧拉角：俯仰角、横滚角、航向角

俯仰角 Pitch=asin(-2*q3*q2+2*q0*q2)*57.3 (式 4-5)

横滚角 roll=atan2(2*q2*q3+2*q0*q1,-2*q1*q1-2*q2*q2+1)*57.3 (式 4-6)

航向角 yaw=atan2(2*(q1*q2 + q0*q3),q0*q0+q1*q1-q2*q2-q3*q3)*57.3 (式 4-7)

公式中的 57.3 表示弧度转换为角度，即 180/π 。

4.3.2 STM32 移植 DMP

要想使用 DMP 来获取欧拉角，我们还需要将 DMP 固件库移植到控制芯片 STM32 上。首先，我们需要在官网上下载 DMP 库和 IIC 文件。在 MDK5 上创建 STM32F103 工程文件，选中如图 4-4 所示文件添加至 HARDWARE 中。

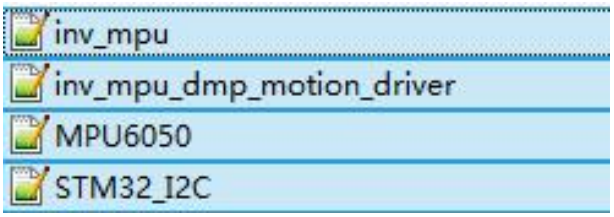

图 4-4 需要添加文件

5 系统的软件设计

5.1手势识别软件设计

本文中，手势识别是依靠于数据手套实现的。因此，手势识别软件设计也可以称为是数据手套软件设计，其可分为柔性弯曲度光纤传感器软件设计和 MPU6050 姿态传感器软件设计。

5.2柔性弯曲度纤维传感器软件设计

这部分的设计需要用到硬件设备有：柔性弯曲度光纤传感器、控制芯片 STM32、HC05 蓝牙模块、USB 数据线；需要使用的 STM32 内部资源有：模拟数据转换器 ADC、直接内容存取 DMA、定时器 TIM。图 5-1 为程序控制流程图。

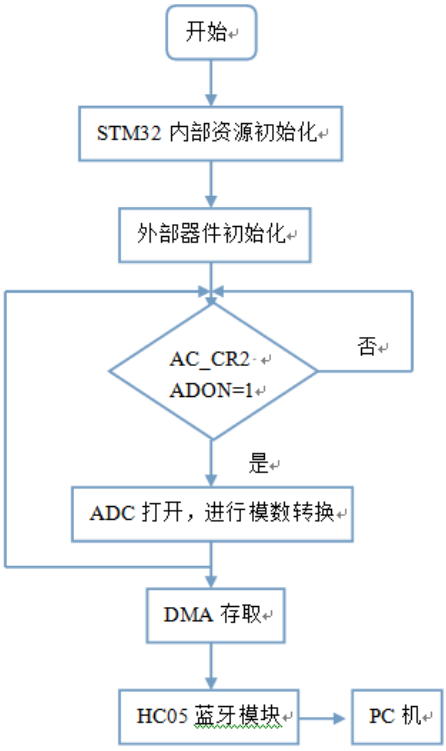

图 5-1 弯曲传感器程序控制流程图

此部分的程序设计核心程序是采集手指模拟信息并对其进行模数转换。采集手指模拟信息需利用 ADC 的信号输入通道，本文中将通道 10-14 都设置为模拟信号输入的引脚，则相对的程序为：

```
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin= GPIO_Pin_0;//PC0 作为输入通道
```

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //模拟输入引脚
GPIO_Init(GPIOC, &GPIO_InitStructure);GPIOC 初始化

GPIO_InitStructure.GPIO_Pin= GPIO_Pin_1;//PC1 作为输入通道
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //模拟输入引脚
GPIO_Init(GPIOC, &GPIO_InitStructure);GPIOC 初始化

GPIO_InitStructure.GPIO_Pin= GPIO_Pin_2;//PC2 作为输入通道
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //模拟输入引脚
GPIO_Init(GPIOC, &GPIO_InitStructure);GPIOC 初始化

GPIO_InitStructure.GPIO_Pin= GPIO_Pin_3;//PC3 作为输入通道
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //模拟输入引脚
GPIO_Init(GPIOC, &GPIO_InitStructure);GPIOC 初始化

GPIO_InitStructure.GPIO_Pin= GPIO_Pin_4;//PC4 作为输入通道
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //模拟输入引脚
GPIO_Init(GPIOC, &GPIO_InitStructure);GPIOC 初始化
```

完成模拟数据引脚的设置后，需配置相应的时钟，即为：

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,  ENABLE);// 用 到 USART1, 开 启 USARTA 时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);//因为复用 USART1,需开启
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);//用到 PC0-4 管脚，所以需要开启 GPIOC 时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1,   ENABLE);//用到 ADC1,需要开启 ADC1 时钟
RCC_ADCCLKConfig(RCC_PCLK2_Div6);// 配 置 ADC1 模 拟 时 钟 PCLK2 的 6 分 频 ，72/6=12MHZ
```

接着就是对 ADC 输出结构体的配置，其程序如下：

```
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;//设置 ADC 为独立工作模式
ADC_InitStructure.ADC_ScanConvMode = ENABLE;//设置 ADC 为扫描模式
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;//设置 ADC 为连续转换模式
ADC_InitStructure.ADC_ExternalTrigConv=ADC_ExternalTrigConv_None;//关闭外部
```

触发

```
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;//转换结果为右对齐
ADC_InitStructure.ADC_NbrOfChannel = 5;//转换通道的数目
ADC_Init(ADC1, &ADC_InitStructure);//初始化ADC1
```

准备工作完成后，需要设置其通道的转换时间。参考STM32中文使用手册，ABP2总线是挂载了ADC、DMA等外设时钟的。因此，我们设置其时钟为：

```
RCC_ADCCLKConfig(RCC_PCLK2_Div6);
```

由以上的设置，可以知道72/6=12MHZ为ADC的时钟频率。接下来，就是对各输入通道的采样顺序和时间进行程序设计。其程序代码如下：

```
ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_239Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 2, ADC_SampleTime_239Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 3, ADC_SampleTime_239Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_13, 4, ADC_SampleTime_239Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 5, ADC_SampleTime_239Cycles5);
```

对于转换结果的存储，这里使用到了DMA功能，所以需要在ADC功能初始化程序里开启DMA功能，并使能，即：

```
ADC_DMACmd(ADC1, ENABLE);
ADC_Cmd(ADC1, ENABLE);
```

完成以上内容后，就可以开启转换。其代码如下：

```
ADC_ResetCalibration(ADC1);
while(ADC_GetResetCalibrationStatus(ADC1));
ADC_StartCalibration(ADC1);
while(ADC_GetCalibrationStatus(ADC1));
```

DMA功能，通ADC功能一样都时属于单片机的外设资源，在对其进行调用是，同样需要进行初始化。其初始化程序如下：

```
void DMA_init(void)
{
   DMA_InitTypeDef DMA_InitStructure;
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);//用到DMA1，需要开启DMA1时钟
DMA_DeInit(DMA1_Channel1);//指定DMA通道
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&(ADC1->DR);//DMA外设ADC地址
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&ADC_ConvertedValue;//DMA内存地址，用来存储转换结果
```

```
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;//数据传输方向，外设到内存
DMA_InitStructure.DMA_BufferSize=10*5;//传输内容的大小，每通道采集次数乘上通道数目
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;//外设地址寄存器不变
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;//存储器地址递增
DMA_InitStructure.DMA_PeripheralDataSize=DMA_PeripheralDataSize_HalfWord ;//数据宽度为 16 位
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord ;//数据宽度为 16 位
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;//DMA 设置为循环传输模式
DMA_InitStructure.DMA_Priority = DMA_Priority_High;//DMA 通道拥有高优先级
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;//本次设计为外设到到内存的传输模式，所以需禁止内存到内存的传输模式
DMA_Init(DMA1_Channel1, &DMA_InitStructure);//初始化 DMA

DMA_Cmd(DMA1_Channel1, ENABLE);//使能 DMA 通道

}
```

完成以上设置后，就可以在主函数中输出相应的数值，其程序如下：

```
int main(void)
{
    int sum;
    u8 i,j;
    float ADC_Value[5];//用来保存经过转换得到的电压值
    ADC_Init();
    DMA_Init();

    ADC_SoftwareStartConvCmd(ADC1, ENABLE);//开始采集

    while(1)
    {
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);//等待传输完成否则第一位数据容易丢失
```