湖南工程学院

毕 业 论 文

题　　目：两轮平衡车的设计与制作

学院：电气信息学院

专业：电子信息工程 班级：1301 学号：201301030106

学生姓名：张巧龙

导师姓名：李延平

完成日期：2017-05-16

两轮平衡车的设计与制作

摘要：以 C/C++等高级语言编程能够帮助开发者更加有效的设计两轮平衡车的控制系统。本文设计的平衡车系统是以 C 语言作为开发语言来设计软件程序的，选用意法半导体公司的 STM32F103 系列的微控制器作为本控制系统的主控芯片，使用惯性传感器 MPU6050 实时采集得到加速度的值及角速度的值，经过卡尔曼滤波算法融合加速度和角度计算出两轮平衡车系统的倾角，并通过 PID 控制器计算出控制 PWM，通过 PWM 信号输出到电机驱动上，从而控制电机的速度和方向，从而控制平衡车的平衡和简单行走。

关键词：STM32F103；MPU6050；卡尔曼滤波；PWM；PID 控制器

Design and manufacture of two wheel balancing vehicle

Abstract： To C / C + + and other high-level language programming to help developers more effective design of two-wheeled vehicle control system. In this paper, the design of the balance system is based on the C language as a development language to design software programs, the use of STMicroelectronics STM32F103 series of microcontrollers as the control system of the master chip, the use of inertial sensor MPU6050 real-time acquisition of the value of acceleration And the angular velocity, the Kalman filter algorithm is used to calculate the inclination of the two-wheeled vehicle system by the acceleration and angle of the Kalman filter algorithm. The PWM is calculated by the PID controller and output to the motor drive through the PWM signal to control the motor speed and direction , Thus controlling the balance of the balance and simple walking.

Keywords: STM32F103; MPU6050; Calman filter; PWM;PID controller

第1章 绪论

1.1 课题研究背景及其意义

上个世纪 80 年代来，伴随着科学技术迅猛发展和经济的高速发展，信息技术、计算机技术及其它相关科学的相互渗透，这些都极大的促进了控制科学与工程技术研究的不断深入，控制系统向智能化控制的发展已经成为一种趋势。早在十九世纪末期，汽车诞生后，人类在交通运输方面走上了新的台阶，而交通工具的技术性能方面有了很大的突破，汽车开始渐渐的成为人们生活的主导，也因此极大地加速了社会进步的进程。而现在，随着汽车的数量在不停的刷新，同时电动车、电动汽车近年来也成为了人们的交通王具的新宠，人们对于车辆的关注程度达到了历史的高峰，人们关注问题的不再只是价格或便捷那么简单，车辆的安全性和舒适性也是很重要的关注焦点。

本世纪来各种关于系统智能化的东西被陆续提出来，如人工智能、智能工厂等相关概念，而汽车作为人民生活的一部分，车辆智能化也是很重要的一大智能课题。如今不仅是各种传感器技术和无线技术已经实现在车体上了，还有各种远程检测控制及无人驾驶等等高新科技将在不远的将来实现在车辆工程上。智能车将作为一口综合性地学科登上舞台，它是电子、物理、数学、计算机、机械、化学等多学科技术的融合，目前已成为了各国科研的热点。

1.2 国内外研究现状

1.2.1 国外研究现状

最早关于双轮平衡机器人的研究是在上个世纪 80 年代，日本 Electro-Commnicat-Ions 大学的 KazuoYamfuji 教授萌生了制作出一种能自动站立的机器人，在 1987 年申请的一项“平行双轮机器人”专利中使用了该技术。以下介绍两轮平衡车的历史发展动态。

1995 年美国发明家安迪·卡门开始研制 Segway 平衡车如图 1.2.1。直到 2001 年 12 月这项高保密的发明公诸于众，他将两轮平衡车作为新型代步工具，并与 2002 年量产出第一辆 Segway HT i167。

2002 年，Leg 公司设计了两轮移动机器人 LegWay，如图 1.2.2 所示。他是第一款移动玩具机器人，可以对它进行远程操控。只有手掌大小、结构紧凑、控制灵活、模块化的结构设计，使得二次安装和拆卸都很方便。

2004 年，Homebrew 机器人俱乐部的 Ted Larson 和 Bob Allen 制作了两轮自平衡机器人 Bender，并在机器人上安装了一个摄像头使它也成为一款自主移动机器人。它由三层板构成，支架做得很高，使重心竖直靠上。但其平衡表现很出色，获得了第一届年度 Robolymipics'' Best of Show''类金牌。

图 2.1 Segway 平衡车

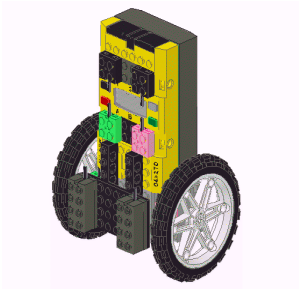

图 2.2 Legway 平衡车

1.2.2 国内研究现状

受国外两轮自平衡机器人研究热潮的影响，国内一些研究机构和平衡车爱好者们经过努力，也取得了诸多的成果。

（1）2004 年，台湾国立大学电机工程研究所利用模糊控制完成了一辆两轮自平衡车的研制。

（2）中国科技大学利用倒立摆控制原理研究出了自平衡两轮代步电动车，在车体内部嵌入式 CPU 的控制下，通过采集平衡传感器和速度、加速度传感器的数据，并建立系统 数学模型，和控制算法，计算输出 PWM 信号，控制两个伺服电机转动，保持车体平衡。在 车子运行中能够根据人体重心的偏移，自动前进、后退、及转向。

（3）哈尔滨工程大学也研制了类似的两轮自立平衡机器人，该系统使用了两块 C8051 单片机和人机交互式的上位机作为核心控制器，使用反射式的红外距离传感器以及双轴加速度计 ADXL202 来检测车体倾角。利用 PWM 动态控制两个直流电机的转速。上位机与机器人之间的数据通信使用的是超小型、超低功耗、高速无线收发 MODEM，人机交互界面采用了 240*128 图形液晶、方向摇杆和按键。基于这些齐全而可靠的硬件设计，使用了一套有效的、独特的软件算法，完成了该系统的平衡控。

1.3 平衡车的应用领域及发展前景

随着先进材料技术、自动化技术、数字信号处理技术、电子科学技术的发展，无人机的发展也越来越好，它可以代替人们进行一些我们很难完成或者很危险的事情以及给人们生活带来便捷。例如：在救险、城市交通、环境监控以及工业安全巡防等领域，平衡车在检测生化武器以及核辐射危险区域等任务中发挥重要作用。平衡车为特殊环境安全巡检提供了新的平台，它能够在工作人员的操控下进行工作，替代人工巡检对象实施接近检测，减少工人的工作强度。在生活中，平衡车可以在堵车的环境中、拥挤的商场中自由的行走，给人们的生活带来便捷。

1.4 本课题主要研究内容

平衡车控制系统的设计与实现是基于 STM32F103 系列的 MCU 为主控平台，以车架为机械载体的平衡车控制系统的设计，包括惯性导航系统的的数据采集、无线数据通信及 PID 控制等的设计与实现，所以本课题的研究主要是以下几个内容：

1.平衡车控制系统的控制理论基础。平衡车的控制系统是基于惯性导航系统实现的，那么在惯性导航系统中姿态是如何检测，如何表示、如何控制的是必须要研究的问题。

2. 平衡车系统的硬件电路设计，在自动控制系统中要有可靠的硬件系统来保证自动化程序的可靠运行，那么系统需要那些硬件电路，这些电路模块是怎么连接也是必须要研究的问题。

3. 平衡车控制系统的软件系统设计，真正自动化的控制还是需要软件来实现，该干什么，该怎么干这些都是软件说了算，这也是本次设计任务中的核心任务。

4. 平衡车的制作与调试，为了验证最终的研究效果就必须要有实际的作品出来，而自动化控制系统的控制效果也需要根据实际应用效果来调整相对应的参数才能保证控制系统能正常高效的工作，所以必须要有实际作品并完成调试工作。

第2章 平衡车控制系统设计基础

2.1 姿态的描述

要实现对平衡车姿态精确控制，首先就需要知道当前平衡车的姿态。平衡车的姿态，指的是平衡车在水平方向的倾角，一般可以用欧拉角来描述，包括横滚角（roll）、俯仰角（pitch）和偏航角（yaw）。根据传感器的安装位置不同，平衡车的姿态角在横滚角和俯仰角之间得到。

2.2 姿态的控制

当能够正确的得到当前的姿态的时候，就可以用其来输出控制了。通过控制平衡车系统的姿态角来控制平衡车的直立。基本的控制思路是：设定目标姿态，即小车处于平衡状态调整两个电机的转速，让测量出来的姿态到达目标姿态，为了避开繁重的动力学建模，选择使用 PID 控制器。

2.2.1 PID 控制器介绍

对于平衡车这种拥有复杂的数学模型的系统，使用 PID 控制器是一种合适且可行的选择。PID 控制器是基于偏差的比例（P）、积分（I）和微分（D）控制的系统，它属于闭环负反馈控制系统中的一种，通过整定 K_p 、K_i 和 K_d 这几个增益参数，能够让自动化控制系统输出的动态响应满足性能指标的要求。PID 控制器的控制原理简单，拥有鲁棒性好、适用范围广，参数易于调节的特点，广泛的应用在自动化控制领域中。

常见的 PID 控制器原理如图 2.2 所示

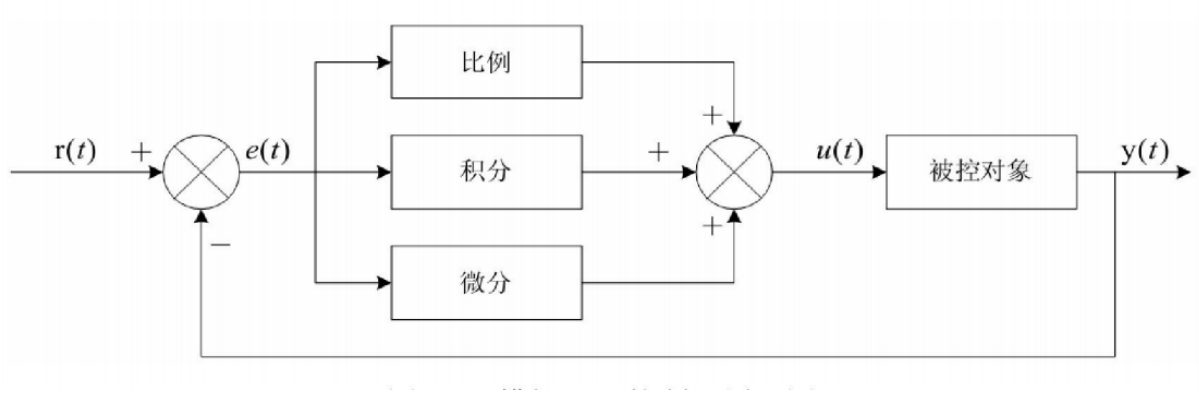

图 2.2 常规 PID 控制器原理图

图中 $r(t)$ 表示目标值，PID 控制器的输入量是 $e(t)$，也就是目标值与实际输出值之间

的控制偏差，PID 控制器的输出量用$u(t)$表示，同时也就是被控对象的输入量，那么被控对象的输出值就是$y(t)$。通过上面的输入输出的关系可以很容易的得到常规 PID 控制器的控制规律：

$$u(t)=K_p e(t)+K_i\int_0^t e(t)dt+K_d\frac{de(t)}{dt} \qquad (2\text{-}2)$$

式(2-2)中K_p、K_i和K_d分别是 PID 控制器中比例、积分和微分的增益参数。

2.3 自动跟随原理介绍

要实现平衡车自动跟随的功能，需要选用能获取当前小车的位置和人的位置的传感器。本文选用单发单收的超声波模块进行测距，在车上的两端各安装一个单收超声波，人手持一个单发的超声波模块。这样左侧与右侧离人的距离就构成了一个三角形。简单示意图如下图 2-3 所示。当小车正对着人时，距离 A=B，当人左拐时，A 必定小于 B，同理当人右拐时，A 大于 B。当人向前走时，A 和 B 的距离必定大于设定距离。由此可获得人与车的距离以及人的行走路线，来实现小车的自动跟随功能。

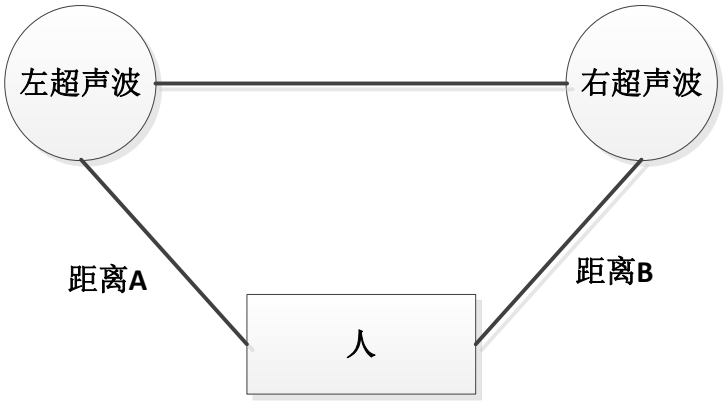

图 2.3 自动跟随原理简单示意图

2.4 系统总体设计方案

根据以上的姿态描述，可以轻松的制定出系统的总体设计方案，控制方案结构示意如图 2.4 所示：

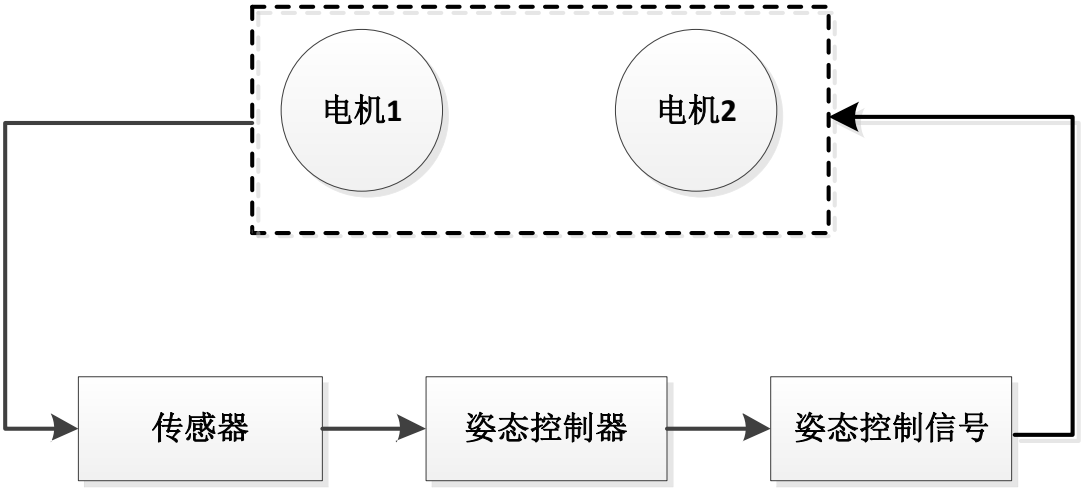

图 2.3 平衡车控制系统总体设计结构示意图

第3章 控制系统硬件设计及实现

3.1 控制系统硬件总体设计

本平衡车控制系统的设计以 STM32F103 系列的 MCU 作为主控，通过惯性传感器 MPU6050 采集姿态数据，实现实时姿态的测量，用蓝牙完成遥控器或者超声波到平衡车的无线通信，使用 TB6612FNG 的电机驱动 IC，然后再加入电源电路、按键控制电路以及 LED 指示灯电路组成最后的平衡车控制电路板。并最终做成实物，完成课题要求。

系统硬件结构如图 3.1 所示：

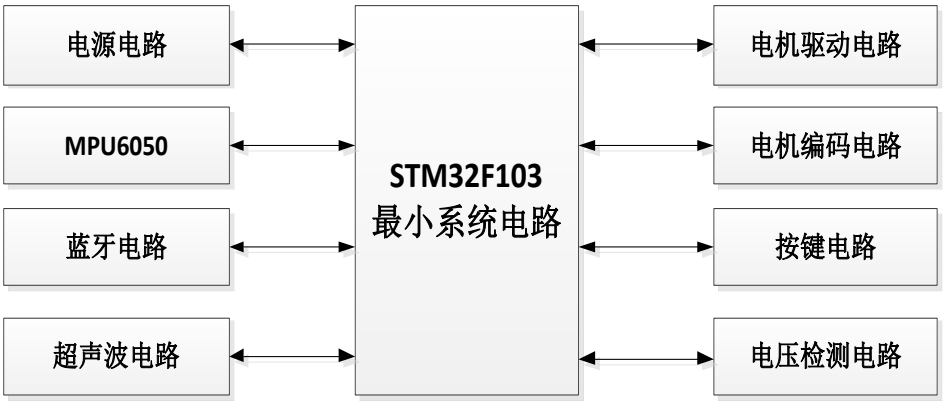

图 3.1 平衡车控制系统硬件结构示意图

3.2 主控 STM32F103C8T6 最小系统电路设计

平衡车控制系统的核心部分就是主控，它执行用户指令并协调其它各个模块的工作。其完成的主要功能就是从传感器获取数据并完成数字滤波、数据融合、姿态解算工作得到准确的姿态信息，控制蓝牙模块接收来自发射端的控制信号以及接受超声波的信号并对其进行处理然后做出相应的反映，最终得到准确的控制量并输出给电机控制模块，完成平衡车的姿态控制及稳定行走。

由于整个控制系统的运算量较大，对实时性要求比较高，需要一块拥有强大运算能力的主控的同时又不能浪费硬件资源，权衡各种性能要求及资源利用后，最终选择使用意法半导体公司的 ARM Cortex-M3 内核的 STM32F103C8T6 32bit 的 MCU 作为本平衡车控制系统的主控芯片，该芯片拥有 48 个引脚，硬件资源丰富，足够完成平衡车控制系统的设计而又不浪费资源。芯片脚位分布如图 3.2 所示：

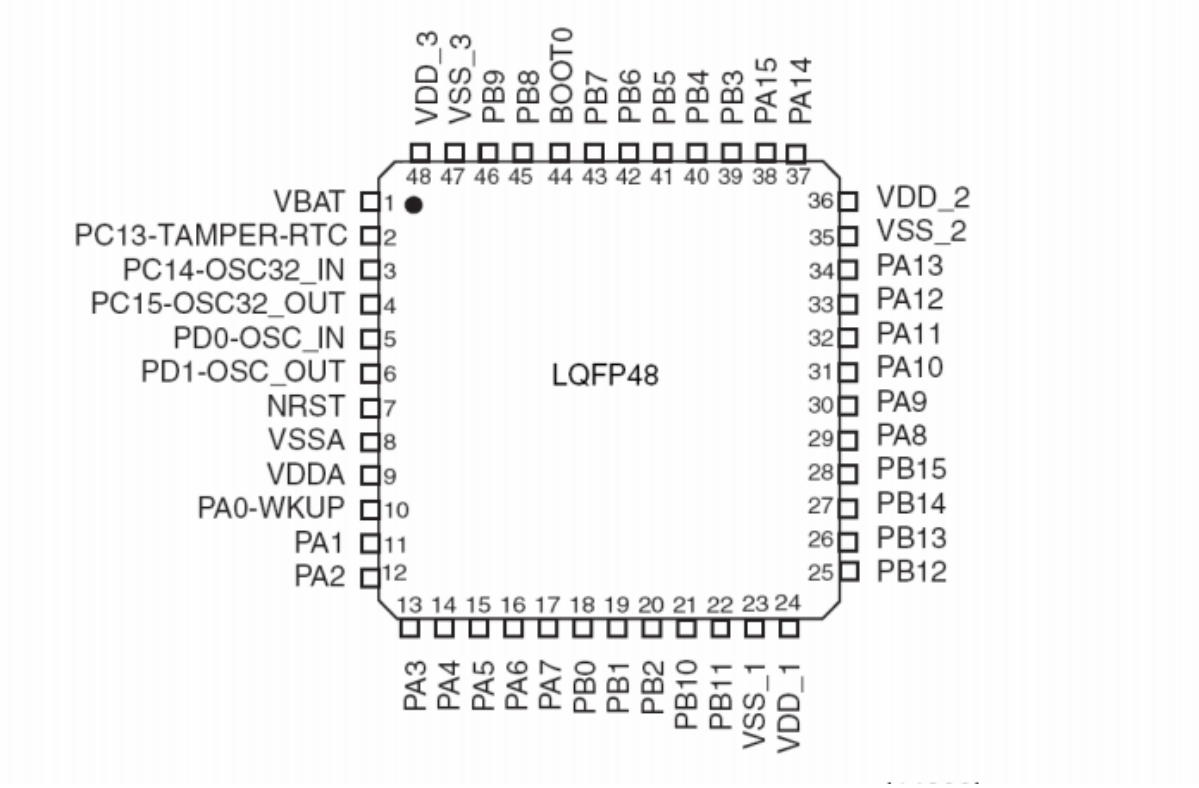

图 3.2　STM32F103C8T6 引脚分布图

芯片所拥有的硬件资源如表 3.1 所示：

外设		STM32F103Tx	STM32F103Cx		STM32F103Rx		STM32F103Vx	
闪存(K字节)		64	64	128	64	128	64	128
SRAM(K字节)		20	20	20	20		20	
定时器	通用	3个(TIM2、TIM3、TIM4)						
	高级控制	1个(TIM1)						
通信接口	SPI	1个(SPI1)	2个(SPI1、SPI2)					
	I^2C	1个(I^2C1)	2个(I^2C1、I^2C2)					
	USART	2个(USART1、USART2)	3个(USART1、USART2、USART3)					
	USB	1个(USB 2.0全速)						
	CAN	1个(2.0B 主动)						
GPIO端口		26	37		51		80	
12位ADC模块(通道数)		2(10)	2(10)		2(16)		2(16)	
CPU频率		72MHz						
工作电压		2.0~3.6V						
工作温度		环境温度：-40°C~+85°C/-40°C~+105°C(见表8) 结温度：-40°C~+125°C(见表8)						
封装形式		VFQFPN36	LQFP48		LQFP64 TFBGA64		LQFP100 LFBGA100	

表 3.1 STM32F103 硬件资源表

根据芯片的用户手册及平衡车控制系统的要求设计主控的最小系统如图 3.3 所示：

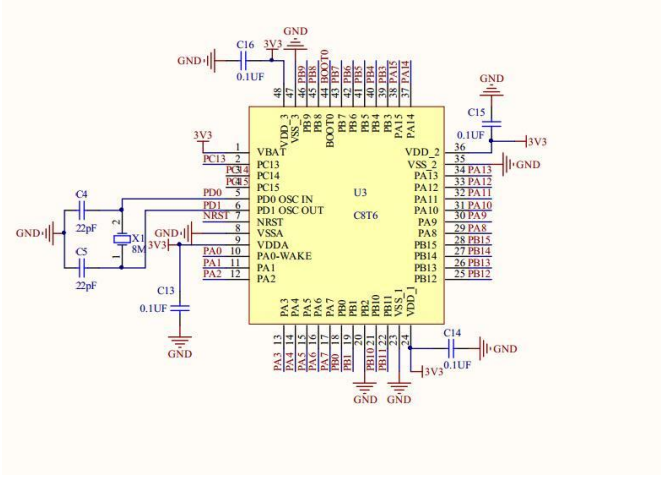

图 3.3 STM32F103C8T6 最小系统电路图

在最小系统中，具有满足 STM32F103C8T6 运行的最小环境，有供电系统、外部晶振电路，芯片复位电路。外加 SWD 的编程接口用于对主控编程及在线调试。

3.3 惯性传感器 MPU6050 电路设计

惯性传感器主要是用于测量平衡车的当前姿态的，较为常用的惯性传感器有加速度计、陀螺仪等，主控 STM32F103C8T6 通过自身的硬件 IIC 总线与传感器通信取得传感器的测量数据，再对数据进行数字滤波处理、数据融合处理后最终解算得到当前的姿态，为了满足控制系统的设计要求，设计选用目前市面上常见的 MPU6050 六轴惯性传感器（三轴加速度传感器与三轴陀螺仪的结合体）。

惯性传感器 MPU6050 电路设计如图 3.3 所示：

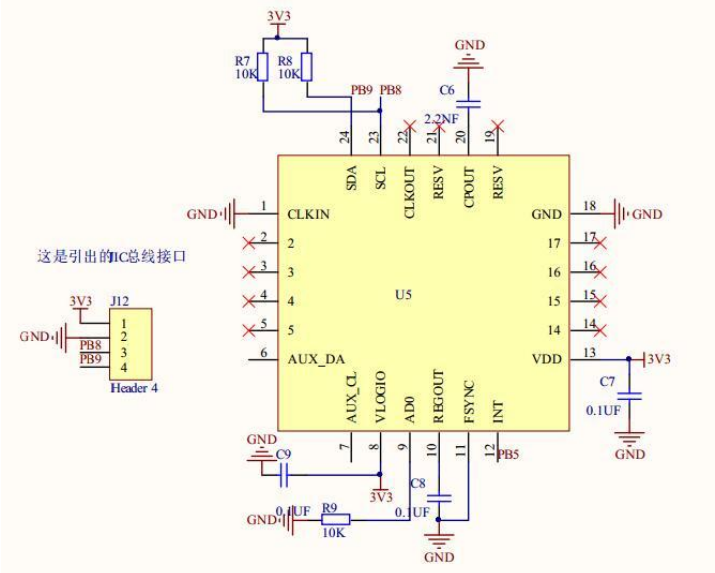

图 3.3 惯性传感器 MPU6050 电路图

3.4 超声波电路设计

本平衡车系统在运行时存在多种工作模式，第一种工作模式是遥控行走模式，第二种是自动跟随行走。自动跟随选用的传感器是单发单收的串口超声波。由本次使用的是成品模块的超声波。只需要简单的把模块引脚连接到主控芯片上就可以稳定工作了。

超声波电路设计如图 3.4 所示：

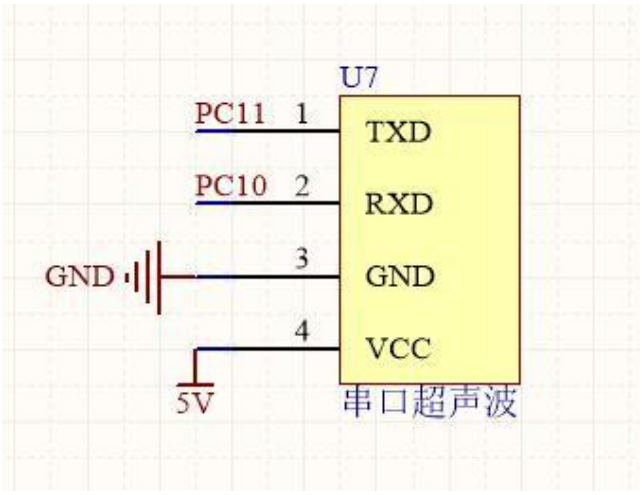

图 3.4 超声波电路设计

3.5 蓝牙电路设计

HC05 是一款主从一体的串口通信芯片，本设计通过此款芯片来完成发射端（遥控器）到接收端（平衡车）的数据通信，从而遥控平衡车行走。

蓝牙模块电路设计如图 3.5 所示：

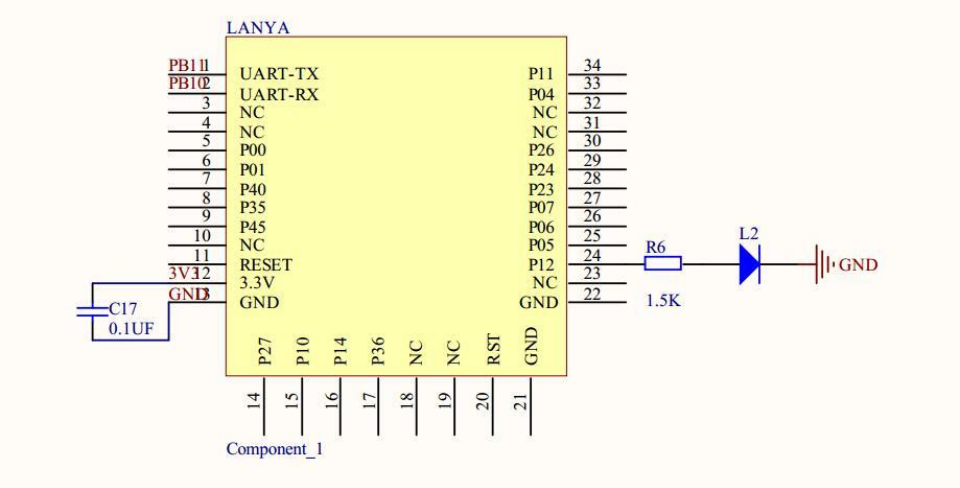

图 3.5 蓝牙电路图

3.6 直流电机驱动电路设计

为了对电机进行调速，必须要外加电机驱动。该模块相对于传统的 L298N 效率上提高很多,体积上也大幅度减少，在额定范围内，芯片基本不发热，并且外围所需要搭建的电路很简单，只需外接电源滤波电容就可以直接驱动电机，有利于减小系统尺寸。它具有大电流 MOSFET-H 桥结构，双通道电路输出，可同时驱动 2 个电机。

驱动电路设计如下图 3.6 所示：

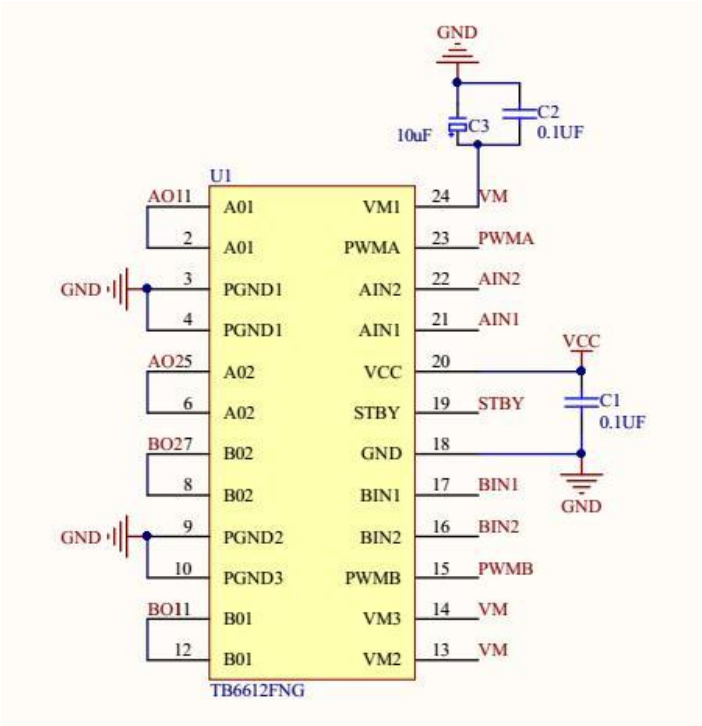

图 3.6 直流电机驱动电路图

3.7 速度测速电路设计

为了使平衡车可以稳定地直立行走，必须控制车速，使平衡车在急转弯时速度不至过快而倒下。通过控制驱动电机上的平均电压可以控制车速，但是如果开环控制电机转速，会受很多因素影响，例如电池电压、电机传动摩擦力、道路摩擦力和前轮转向角度等。这些因素会造成平衡车运行不稳定。通过检测速度，然后对平衡车速度进行闭环反馈控制，即可消除上述各种因素的影响，使得平衡车运行得更稳定。本次采用的测速模块是霍尔编码器，霍尔编码器是一种通过磁电转换讲输出轴上的机械几何位移量转换成脉或数字量的传感器。霍尔编码器是由霍尔码盘和霍尔元件组成，霍尔码盘是在一定直径的圆板上等分地布置有不同的磁极。霍尔码盘与电动机同轴，电动机旋转时，霍尔元

件检测出若干脉冲信号，为判断转向，一般输出两组存在一定相位的方波差。简单的示意图如图 3.7 所示。

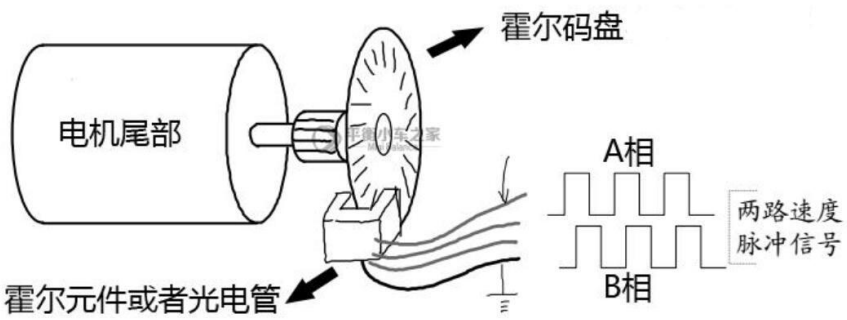

图 3.7 测速示意图

3.8 电池电压检测电路设计

本平衡车系统使用的动力能源是 12v 的航模锂电池锂，航模锂电池具有强大的放电能力，能够满足本平台对动力能源的要求，但是当锂电池过放时回影响锂电池的性能，所以为了不让锂电池过放，就需要让控制系统时刻知道当前的电池电压，当检测到电池电压过低的时候要做出相应的保护措施。所以对电池电压的采集很有必要。

电池电压检测电路设计如图 3.8 所示：

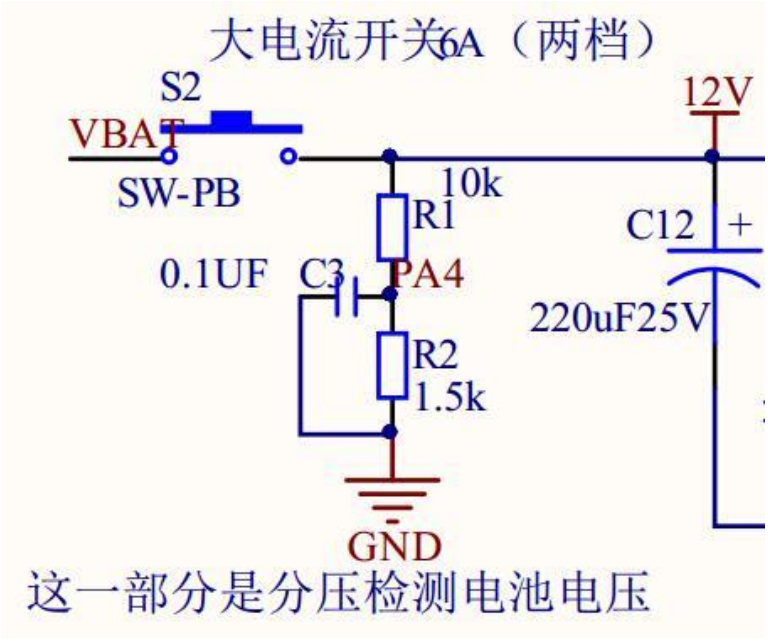

图 3.8 电池电压检测电路图

3.9 电源稳压电路设计

在每一个电路系统中，一个稳定的电源电压是系统能够稳定工作的基础，同时也能

确保传感器测量数据的准确性。所以必须给平衡车控制电路板加入稳定的电源稳压电路。由于该平衡车系统是由标称 12v 的锂电池供电，然而主控芯片 STM32F103C8T6、传感器、蓝牙模块都工作在 3.3v 或 5v 的电压下，所以选择直接从锂电池的输出电压直接稳压到 3.3v 和 5v 给控制系统供电，电机直接由锂电池供电。

选用电源稳压电路设计如图 3.9 所示：

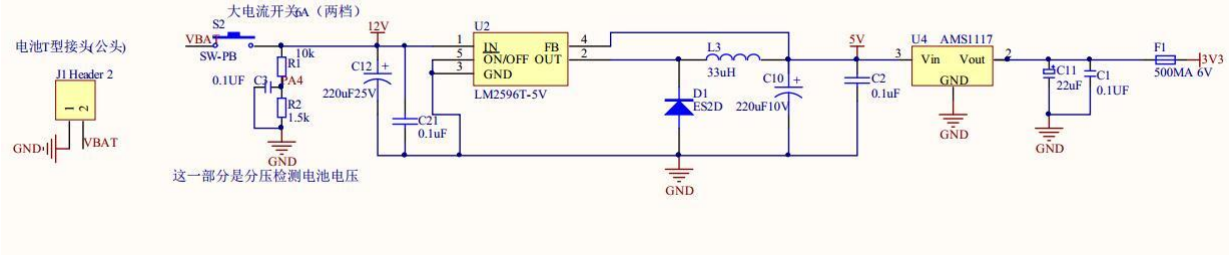

图 3.9 电源稳压电路图

第4章 控制系统软件设计及实现

4.1 控制系统软件总体设计

由于平衡车的机械结构简单，所以将大部分复杂的处理都通过软件来完成，导致软件设计的难度较大，所以平衡车控制系统的软件设计是此次设计最为核心的部分。软件设计主要分为传感器数据采集极其数据处理、姿态解算、蓝牙数据通信、超声波数据通信、PID 控制及 PWM 信号输出这个部分。由于模块化的设计，使得程序的思路非常清晰。程序主体流程如图 4.1 所示：

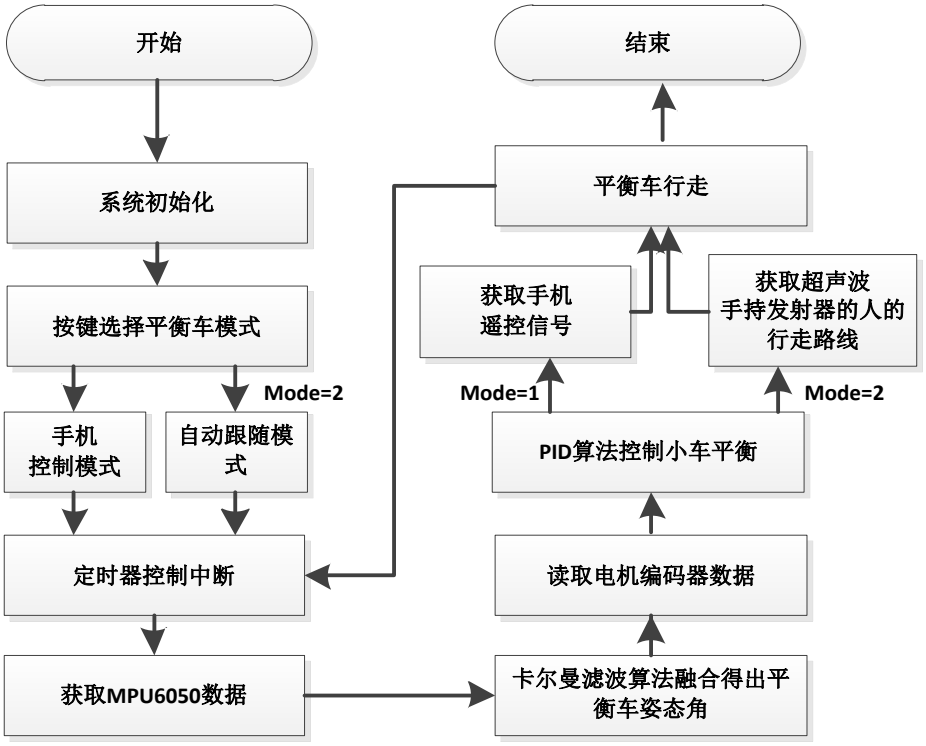

图 4.1 控制系统软件总体流程图

其中系统模块初始化需要完成的工作就是初始化好主控的各个 IO 口，完成 IIC、USART 及 ADC 的初始化，然后再完成传感器、无线模块、超声波模块的初始化配置。

之后便进入整个控制系统的大循环体中依次执行相关的任务。

检测按键就是通过按键设置平衡车的工作模式，第一种是通过遥控器通过蓝牙通信，获得操控着对平衡车的操控命令，比如说加减速，前进后退等。第二种模式，小车自动跟随手上持有超声波发射器的人。

采集传感器数据第一个是主控芯片通过 IIC 总线与 MPU6050 惯性传感器通信，得到传感器采集的原始数据，然后在做一些滤波处理，给后续的姿态解算提供一个可靠的数

据，第二个是主控芯片通过 USART 与蓝牙模块通信，为模式一提供数据。第三个是主控芯片通过 USART 与超声波模块通信，为模式二提供数据。

更新当前的姿态是主控根据传感器所采集到的有效数据计算出当前姿态的过程，这部分姿态解算程序的设计可以算得上是整个软件系统设计中最核心的部分，因为这部分的程序设计的好坏可以直接影响到最终平衡车行走的效果。

在解算出当前的姿态和得到操控者的控制命令后，就需要计算出相应的控制输出量，然后输出控制，以达到操控着的命令要求。

其实有上面这些步骤已经可以保证平衡车稳定运行，但是为了保证平衡车的安全性，还需要时刻检测电池电压的状态防止损坏小车以及电池过放。

出于对控制系统中对陀螺仪积分运算的要求，保证每次积分时间固定且可靠，要求程序主题循环每 10ms 执行一次，即积分时间为 0.01s，姿态、控制的刷新频率为 100Hz。

4.2 传感器 MPU6050 数据采集与处理

要实现对平衡车的控制就必须要先知道平衡车当前的姿态，而在平衡车系统中常用具有三轴加速度计和三轴陀螺仪的 MPU6050 集成型惯性传感器来采集平衡车的姿态信息。先由主控获取传感器的原始数据，然后对数据进行滤波处理以及姿态解算之后获取一个相对精确的姿态信息，然后再结合接受到的控制信息计算出输出控制量，完成对姿态的控制，形成这样一个闭环控制系统。

所以对于传感器的数据采集及数据处理显得十分重要。

4.2.1 传感器 MPU6050 数据采集

MPU6050 提供的数据通信方式的 IIC 总线，最快允许 400kbps 的通信速率，为了通信的稳定性，选择模拟 IIC 通信。

在采集到正确的传感器原始数据之前，需要先通过 IIC 通信对 MPU6050 完成配置。

配置程序如下：

```
u8 MPU_Init(void)
{
    u8 res;
    MPU_IIC_Init();//初始化 IIC 总线
    MPU_Write_Byte(MPU_PWR_MGMT1_REG,0X80);
   delay_ms(100);
    MPU_Write_Byte(MPU_PWR_MGMT1_REG,0X00);
    MPU_Set_Gyro_Fsr(3);
    MPU_Set_Accel_Fsr(0);
    MPU_Set_Rate(200);
    MPU_Write_Byte(MPU_INT_EN_REG,0X00);
    MPU_Write_Byte(MPU_USER_CTRL_REG,0X00);
    MPU_Write_Byte(MPU_FIFO_EN_REG,0X00);
```

```
        MPU_Write_Byte(MPU_INTBP_CFG_REG,0X80);
        res=MPU_Read_Byte(MPU_DEVICE_ID_REG);
        if(res==MPU_ADDR)
        {
            MPU_Write_Byte(MPU_PWR_MGMT1_REG,0X01);
            MPU_Write_Byte(MPU_PWR_MGMT2_REG,0X00);
            MPU_Set_Rate(50);
        }else return 1;
        return 0;
}
```

在配置完后就只需要等带等待传感器把数据转换完后，从传感器读取出数据就行，由于配置后的传感器采样率高于程序读取传感器数据周期，所以每次读取传感器的数值都能保证得到的数据是最新的数据。

原始数据的读取程序如下：

```
u8 MPU_Get_Gyroscope(short *gx,short *gy,short *gz)
{
    u8 buf[6],res;
     res=MPU_Read_Len(MPU_ADDR,MPU_GYRO_XOUTH_REG,6,buf);
     if(res==0)
     {
         *gx=((u16)buf[0]<<8)|buf[1];
         *gy=((u16)buf[2]<<8)|buf[3];
         *gz=((u16)buf[4]<<8)|buf[5];
     }
    return res;
}
u8 MPU_Get_Accelerometer(short *ax,short *ay,short *az)
{
    u8 buf[6],res;
     res=MPU_Read_Len(MPU_ADDR,MPU_ACCEL_XOUTH_REG,6,buf);
     if(res==0)
     {
         *ax=((u16)buf[0]<<8)|buf[1];
         *ay=((u16)buf[2]<<8)|buf[3];
         *az=((u16)buf[4]<<8)|buf[5];
     }
    return res;
}
```

4.2.2 传感器 MPU6050 数据处理

然而传感的原始数据并不是完全可靠的，还需要对其做一定的滤波处理和单位转换之后才能用于姿态的解算。

一般来说加速度计非常容易受到震动的影响，它采集的是沿各个轴向的加速度值，当机体发生震动时都会导致加速度计采集出来的加速度信号有大幅度的跳动，所以需要对加速度计采集出来的数据进行数字低通滤波处理，通常在这之前先做一次滑动平均滤波处理的话效果会更好。

陀螺仪采集的是绕各轴旋转的角速度值，虽然对震动并不敏感，但是陀螺仪却容易受到温度的影响，环境温度的变化会导致采集到的角速度值不一样，好在温度偏移是缓

慢的，并不会导致角速度有很大的变化，而且因为温度不变的话静态时检测到的角速度值虽不为 0，也是一个稳定的数值，所以只要对陀螺仪采集到的数据做好零偏处理和单位转化就可以用于姿态解算了。

4.3 蓝牙 HC-05 模块数据收发实现

4.3.1 蓝牙 HC-05 初始化配置

本次设计的平衡车有两种模式，模式一：遥控行走平衡车。为了能无线遥控平衡车，可在平衡车系统上进行了蓝牙与手机上位机的通信，从而控制小车的行走。蓝牙与主控的数据通信以及主控对蓝牙的控制都是通过 USART 实现的。同样为了能够正常使用蓝牙模块还需要相对其进行初始化处理，初始化函数如下：

```
void uart1_init(u32 bound)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1|RCC_APB2Periph_GPIOA, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    USART_Cmd(USART1, ENABLE);
}
```

4.3.2 蓝牙 HC-05 数据采集和处理

当蓝牙串口配置成功之后，直接在串口中断读取数据即可，在读取之后，可直接处理数据，然后将控制信号给单片机，单片机做出相应的处理，控制平衡车行走。数据采集和处理的函数如下：

```
void USART1_IRQHandler(void)
{
        u16 Res;
        if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
        {
            Res =USART_ReceiveData(USART1);
            if(Res==0x00)    Flag_Qian=0,Flag_Hou=0,Flag_Left=0,Flag_Right=0;
            else if(Res==0x01)      Flag_Qian=1,Flag_Hou=0,Flag_Left=0,Flag_Right=0;
```

```
                else if(Res==0x05)      Flag_Qian=0,Flag_Hou=1,Flag_Left=0,Flag_Right=0;
                else if(Res==0x02||Res==0x03||Res==0x04)
                Flag_Qian=0,Flag_Hou=0,Flag_Left=0,Flag_Right=1;
                else if(Res==0x06||Res==0x07||Res==0x08)
                Flag_Qian=0,Flag_Hou=0,Flag_Left=1,Flag_Right=0;
                else Flag_Qian=0,Flag_Hou=0,Flag_Left=0,Flag_Right=0;
            }
    }
```

4.4 蓝牙 HC-05 控制程序框图

控制者通过手机上位机给出控制信号，通过手机蓝牙与平衡车上的蓝牙进行通信，当平衡车得到手机上位机的控制信号之后，处理控制信号，然后通过平衡车的速度环以及转向环控制平衡车的简单行走。蓝牙控制程序框图如下图 4.4 所示。

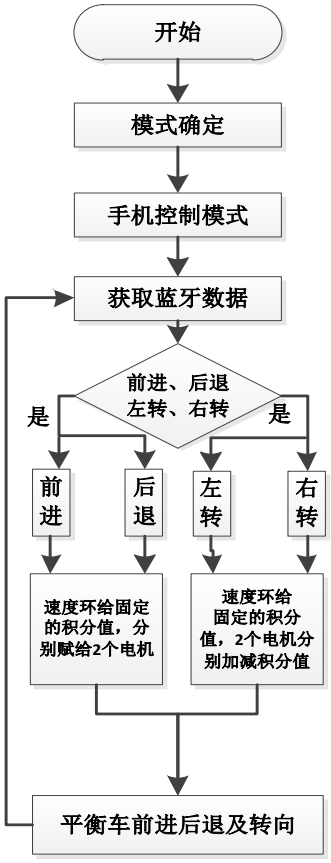

图 4. 4 蓝牙控制程序框图

4.5 超声波模块数据读取及处理

4.5.1 超声波模块初始化配置

本次设计的平衡车有两种模式，模式二：自动跟随。为了能平衡车自动跟着人行走

走，本平衡车系统选用的超声波是单发单收的超声波模块。平衡车上装载了两个超声波，人手上持有一个发射超声波。超声波与主控的数据通信是通过 USART 实现的。同样为了能够正常使用超声波模块还需要相对其进行初始化处理，初始化函数如下：

```
void uart3_init(u32 bound)
{
    GPIO_InitTypeDef    GPIO_InitStructure;
    USART_InitTypeDef    USART_InitStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    NVIC_InitStructure.NVIC_IRQChannel = USART3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=2 ;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART3, &USART_InitStructure);
    USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
    USART_Cmd(USART3, ENABLE);
}
```

4.5.2 超声波数据采集和处理

超声波串口配置成功之后，直接使用串口输出超声波数据即距离，输出频率是一秒发送 50 个数据包， 数据包格式是 0XA5 加两个字节的数据。数据的单位是毫米。所以在读取完数据之后需要做一些处理。数据采集和处理的函数如下：

```
void uart3_init(u32 bound)
{
    if(USART_GetITStatus(USART3, USART_IT_RXNE) != RESET)
    {
        dat1[num1]=USART_ReceiveData(USART3);
        if(dat1[0] != 0xa5) num1 = 0;
        else num1++;
        if(num1==3)
        {
            num1 = 0;
            distance1 = dat1[1]<<8 | dat1[2];
        }
    }
}
```

4.6 超声波模块控制程序框图

在模式二中，当采集到手持超声波发射器的人和平衡车系统的距离之后，将距离数据给单片机，单片机处理之后，然后将处理完之后的数据通过平衡车的速度环以及转向环给平衡车，从而完成平衡车的自动跟随。自动跟随程序框图如下图 4.4 所示。

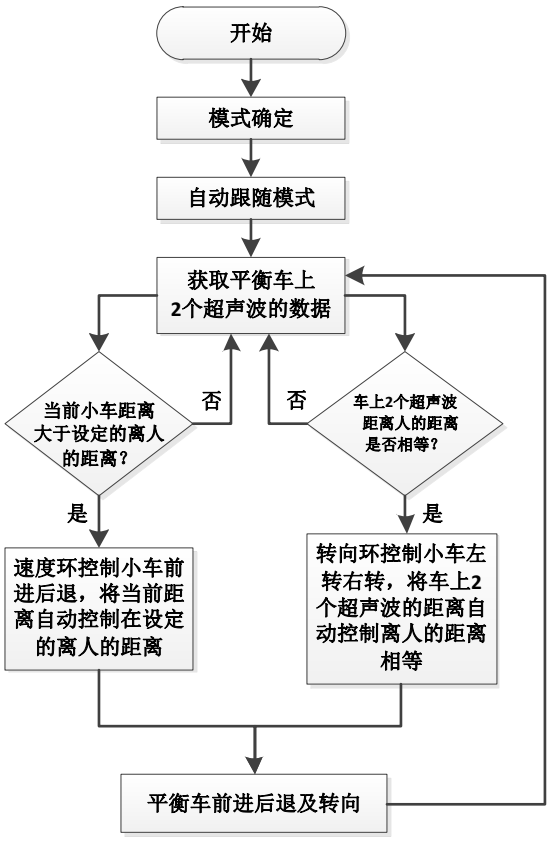

图 4. 6 超声波控制程序框图

4.7 PWM 信号输出

在直流电机的调速控制里面，常用的调速方式就是利用 PWM 控制信号来完成调速功能，在 STM32 的 MCU 中可以利用定时器产生 PWM，根据定时器的配置不同，PWM 的模式不同，当定时器设置的模式为 TIM_OCMode_PWM1 时，PWM 控制信号的占空比越高，电机的转速就越快，反之 PWM 控制信号的占空比越低，电机的转速就越慢，当定时器设置的模式为 TIM_OCMode_PWM2 时，PWM 控制信号的占空比越高，电机的转速就越慢，反之 PWM 控制信号的占空比越低，电机的转速就越快。

在本平衡车控制系统中，通过输出两路 PWM 控制信号来控制两个电机的转速。由上一章的电路设计可以知道系统选用的 STM32F103C8T6 拥有三个通用定时器和一个高

级定时器，其中高级定时器可以产生两路频率一样但占空比不一样的 PWM 控制信号，可以完成系统对两路 PWM 控制信号输出的要求。

在程序设计中，通过官方提供的驱动程序对高级定时器进行配置，使其输出两路占空比相互独立的 PWM 控制信号。在程序运行的过程中只需要改变 CCRX 寄存器中的值就可以改变对应路数的占空。

PWM 输出配置程序如下：

```
void TIM1_PWM_Init(u16 arr,u16 psc)
{
	GPIO_InitTypeDef GPIO_InitStructure;
	TIM_TimeBaseInitTypeDef	TIM_TimeBaseStructure;
	TIM_OCInitTypeDef	TIM_OCInitStructure;
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA , ENABLE);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_11;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &GPIO_InitStructure);
	TIM_TimeBaseStructure.TIM_Period = arr;
	TIM_TimeBaseStructure.TIM_Prescaler =psc;
	TIM_TimeBaseStructure.TIM_ClockDivision = 0;
	TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
	TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
	TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OCInitStructure.TIM_Pulse=0;
	TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
	TIM_OC1Init(TIM1, &TIM_OCInitStructure);
	TIM_OC4Init(TIM1, &TIM_OCInitStructure);
	TIM_CtrlPWMOutputs(TIM1,ENABLE);
	TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
	TIM_OC4PreloadConfig(TIM1, TIM_OCPreload_Enable);
	TIM_ARRPreloadConfig(TIM1, ENABLE);
	TIM_Cmd(TIM1, ENABLE);
}
```

4.8 姿态解算 C 语言实现

得到 MPU6050 可靠数据之后，还需要使用这些数据来解算出当前的平衡车的姿态。MPU6050 中陀螺仪可以得到当前平衡车系统 XYZ 三个方向的角速度，按理说，通过对角速度积分，便能得到当前系统的姿态角，但是陀螺仪本身极易受噪声干扰，产生漂移误差，误差会越来越大。目前常用于姿态解算算法有卡尔曼滤波算法以及互补滤波算法。

4.8.1 卡尔曼滤波算法

卡尔曼滤波算法是一种线性最小方差估计，是一种离散线性滤波的递推算法，姿态计算的精度很高，在平衡车系统中，卡尔曼滤波利用传统姿态结算方法建立滤波模型，

然后通过预测更新过程得到下一时刻的姿态角。

卡尔曼滤波算法的 C 语言程序实现如下：

```
float K1 =0.02;
float angle, angle_dot;
float Q_angle=0.001;
float Q_gyro=0.003;
float R_angle=0.5;
float dt=0.005;
char    C_0 = 1;
float Q_bias, Angle_err;
float PCt_0, PCt_1, E;
float K_0, K_1, t_0, t_1;
float Pdot[4] ={0,0,0,0};
float PP[2][2] = { { 1, 0 },{ 0, 1 } };
void Kalman_Filter(float Accel,float Gyro)
{
        angle+=(Gyro - Q_bias) * dt; //先验估计
        Pdot[0]=Q_angle - PP[0][1] - PP[1][0];
        Pdot[1]=-PP[1][1];
        Pdot[2]=-PP[1][1];
        Pdot[3]=Q_gyro;
        PP[0][0] += Pdot[0] * dt;
        PP[0][1] += Pdot[1] * dt;
        PP[1][0] += Pdot[2] * dt;
        PP[1][1] += Pdot[3] * dt;
        Angle_err = Accel - angle;
        PCt_0 = C_0 * PP[0][0];
        PCt_1 = C_0 * PP[1][0];
        E = R_angle + C_0 * PCt_0;
        K_0 = PCt_0 / E;
        K_1 = PCt_1 / E;
        t_0 = PCt_0;
        t_1 = C_0 * PP[0][1];
        PP[0][0] -= K_0 * t_0;
        PP[0][1] -= K_0 * t_1;
        PP[1][0] -= K_1 * t_0;
        PP[1][1] -= K_1 * t_1;
        angle += K_0 * Angle_err;
        Q_bias       += K_1 * Angle_err;
        angle_dot      = Gyro - Q_bias;
}
```

4.8.2 互补滤波法

互补滤波法的基本解算思路：把加速度计的误差构造成纠正旋转，然后叠加到陀螺仪测量出来的角增量上来纠正陀螺仪的积分误差。

互补滤波法的 C 语言程序实现如下：

```
void sensfusion6UpdateQ(float gx, float gy, float gz, float ax, float ay, float az, float dt)
{
   float recipNorm;
   float halfvx, halfvy, halfvz;
   float halfex, halfey, halfez;
   float qa, qb, qc;
   gx = gx * M_PI / 180;
```

```
    gy = gy * M_PI / 180;
    gz = gz * M_PI / 180;
    if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f)))
    {
       recipNorm = invSqrt(ax * ax + ay * ay + az * az);
       ax *= recipNorm;
       ay *= recipNorm;
       az *= recipNorm;
       halfvx = q1 * q3 - q0 * q2;
       halfvy = q0 * q1 + q2 * q3;
       halfvz = q0 * q0 - 0.5f + q3 * q3;
        halfex = (ay * halfvz - az * halfvy);
       halfey = (az * halfvx - ax * halfvz);
       halfez = (ax * halfvy - ay * halfvx);
       if(twoKi > 0.0f)
       {
          integralFBx += twoKi * halfex * dt;
          integralFBy += twoKi * halfey * dt;
          integralFBz += twoKi * halfez * dt;
          gx += integralFBx;
          gy += integralFBy;
          gz += integralFBz;
       }
       else
       {
          integralFBx = 0.0f;
          integralFBy = 0.0f;
          integralFBz = 0.0f;
       }
       gx += twoKp * halfex;
       gy += twoKp * halfey;
       gz += twoKp * halfez;
    }
    gx *= (0.5f * dt);
    gy *= (0.5f * dt);
    gz *= (0.5f * dt);
    qa = q0;
    qb = q1;
    qc = q2;
    q0 += (-qb * gx - qc * gy - q3 * gz);
    q1 += (qa * gx + qc * gz - q3 * gy);
    q2 += (qa * gy - qb * gz + q3 * gx);
    q3 += (qa * gz + qb * gy - qc * gx);
    recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
    q0 *= recipNorm;
    q1 *= recipNorm;
    q2 *= recipNorm;
    q3 *= recipNorm;
}
```

4.9 经典 PID 控制器 C 语言实现

PID 控制的优点在于不需要精确的数学建模，只需要知道姿态偏差就能够通过参数

整定满足要求的动态响应。

本平衡车控制系统软件中的 PID 控制器的输入量是目标姿态与当前姿态的偏差，输出量是绕三个轴的力矩，最后对应到两个电机的转速上去。

PID 控制器的 C 语言程序实现如下：

```
float pidUpdate(PidObject* pid, const float measured, const bool updateError)
{
   float output;
   if (updateError)
   {
      pid->error = pid->desired - measured;
   }
   pid->integ += pid->error * IMU_UPDATE_DT;
   if (pid->integ > pid->iLimit)
   {
      pid->integ = pid->iLimit;
   }
   else if (pid->integ < -pid->iLimit)
   {
      pid->integ = -pid->iLimit;
   }
   pid->deriv = (pid->error - pid->prevError) / IMU_UPDATE_DT;

   pid->outP = pid->kp * pid->error;
   pid->outI = pid->ki * pid->integ;
   pid->outD = pid->kd * pid->deriv;

   output = (pid->kp * pid->error) +
            (pid->ki * pid->integ) +
            (pid->kd * pid->deriv);

   pid->prevError = pid->error;

   return output;
}
```