Robomasters2015

全国大学生机器人大赛

技术报告

学　　校：　山东科技大学

队伍名称：　**Smart Robot**

2015.07

摘 要

本文是在 RoboMasters 全国大学生机器人大赛背景下诞生的，详细介绍了为迎战决赛所准备的机器人系统方案。该系统采用嵌入式系统 STM32F405RG 作为本系统的主控制器，在 Keil 开发环境中进行软件开发，实现机器人的整体性能。整个系统涉及系统硬件设计、软件算法、车体机械结构的设计与调整和系统调试等多个方面。硬件方面介绍了单片机的选型和各个硬件模块的设计，其中主要介绍了电源管理模块、电机的 H 桥驱动和舵机控制模块；在软件算法方面主要介绍了 PID 算法和滤波算法；机械结构方面介绍了弹仓的修改和补给站的设计，最后介绍了系统的调试工具。

关键词：RoboMasters；机器人；H 桥；PID 算法

ABSTRACT

This paper is based on the context of the Robomasters national college robot contest, detailing robot system control system prepared to the host the finals.The controller of the system is STM32F405RG,do the software development in the Keil development environment and the overall performance of the robot. The whole system involves system architecture and design of hardware, software algorithm ,the mechanical structure design and adjustment and debugging systems, etc. The front tire's optimization, the front position, The MCU hardware selection and hardware design of each module are introduced in the hardware design, in which the power management module, motor derive with H bridge and steering gear control are mainly introduced; In the software, PID algorithm and filtering are the focus; the magazine and supply are introduced in the mechanical structure,finally,introduces the debugging tools.

Key words: RoboMasters；robots；H bridge；PID algorithm

目　录

1 绪论

RoboMasters2015 全国大学生机器人大赛是由共青团中央学校部和全国学联秘书处联合主办、大疆创新科技有限公司承办的竞赛，是被誉为“战斗力爆表”的激战类机器人竞赛，其标语为“不一样的战斗”。其设计内容涵盖了机械、汽车、电子、电气、自动控制、计算机、传感技术、能源等多个学科的的知识领域，其目的宗旨在为大学生提供颠覆传统的创新实践平台，将教学理论与科技实践紧密结合，让高校学生在这个平台体验淋漓尽致的技术对抗，在团队中展现人格魅力、感受合作的力量，探索科技与人之间的默契。

RoboMasters 是一项类似真人 CS 的全新机器人对抗赛，采用红、蓝双方对抗的形式，每场限时 5 分钟。参赛双方各派五名“汽车人”战将，在复杂地形中相互较量，进行“实弹”射击对抗，完成指定任务，电子裁判系统判断比赛胜负。

战车本身的塑造和战斗场地的设计是本次大赛的两大亮点。冲锋陷阵的“汽车人”军团有着严格的步兵、射手、炮手、哨兵等分工。步兵由组委会提供，其余机器人都由参赛队自行设计和组装。比赛使用的“弹丸”是真正可发射的 BB 弹丸，而机器人车身内部设有可以计算被射中次数的传感器，通过机器人身后附带的可视血条装置,能直观反映出机器人当前状况,被攻击次数过多导致“血量”不足的机器人会被切断电源，“扫地出局”。而在篮球场大小的比赛场地内，四周设有防护围栏，场地中设有模拟各种战斗情境的障碍物。基地、维修站、瞭望塔、补给站、战壕、桥梁等军事设施一应俱全。

比赛汇聚了当今科研领域尖端的机器人研发技术，并融入了电子竞技元素。参赛者在场外通过电脑屏幕，用鼠标键盘实时操控战车，在各个队员的配合下进行对战，比赛中以机器人第一人称视角的操作模式，通过图像传输技术实时传回高清图像，实时展现场地状况，使操作手能身临其境地向机器人发出指令。比赛场地实现全智能化实时控制，信号控制灯光与烟雾的搭配，突出声光电效果，使现场充满战斗气氛。此外，比赛还应用了视觉处理应用、自稳云台应用等技术，实现高精度跟踪锁定目标，科技的力量打造了精彩的“汽车人”竞技战场。

2 系统硬件设计

2.1 系统硬件组成

系统的硬件电路中最重要的是系统控制电路的设计。设计机器人控制系统的电路，首先需要分析系统的输入、输出信号，然后选择合适的核心控制嵌入式计算机（单片机），逐步设计各个电路子模块，最后形成完整的控制电路。系统的整个控制电路划分为如下子模块：

（1） 单片机最小系统，程序下载调试接口等；

（2） 电源：DC/DC转换，稳压，滤波、退耦电路；

（3） 舵机接口：云台控制和弹仓盖开关控制；

（4） 摩擦轮电调接口；

（5） 过流保护；

（5） 底盘电机驱动：H桥电路；

（6） 设置与调试：显示系统运行状态、速度设定、程序下载与监控。

2.2 微处理器简介

战车要求具备足够高的灵活性和控制性，因此控制芯片应具有较高的实时处理能力和较高的处理速度。本系统选用嵌入式微处理器STM32F405，该芯片使用ARM先进架构的Cortex-M4内核，32位精简指令集的核心运行频率高达168MHz，15个I/0端口用于连接外部设备，并集成了3个ADC、2个DAC、定时器、实时时钟、CRC计算单元和模拟真随机数发生器等在内的整套先进外设。芯片具有速度快、功耗低、可靠性高、实时性强等优点[1]。为了实现机器人的整体性能，需要利用以下资源：

(1) PWM接口（8路）

4路用于底盘电机驱动；2路用于摩擦轮；1路用于云台舵机；1路用于拨弹电机。

(2) IO接口

底盘电机驱动控制信号；电源电压检测；电流保护。

(3) 通讯接口（2个）

一路用于遥控器接收输入口；一路用于调试入口。

(4) AD（1路）

用于H桥输出电流的过流保护。

2.3 电源管理模块设计

稳定的电源对于一个控制系统来说至关重要，关系到系统能否正常工作，系统总的电源供应来自24V锂电池，由于电路中的不同电路模块所需要的工作电压和电流容量不相同，因此电源模块包含有多个稳压输出电路，例如摩擦轮电机电调为12V供电，单片机STM32F405RG需要3.3V低压供电，底盘驱动电机为24V，由电池直接供电，因此设计的机器人战车电源电路框架如图2.1所示。

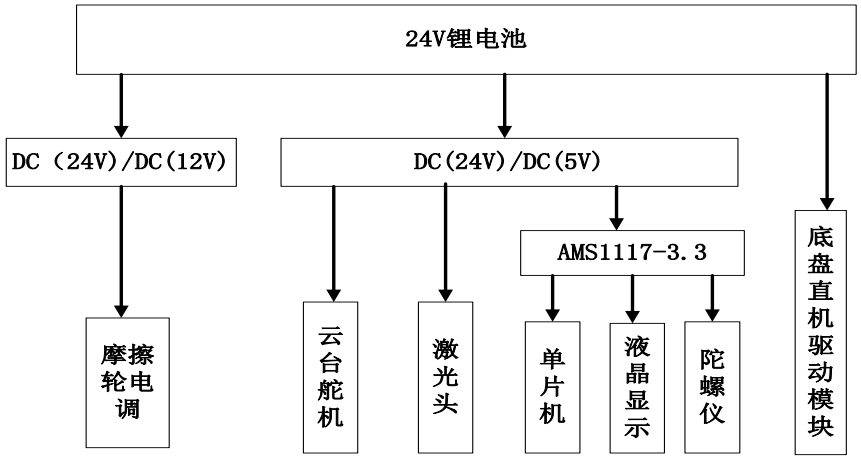


图2.1 电源模块框图

2.3.1 12V 和 5V 稳压电路

车体控制系统中，接入 DC24V 电源线，为提高电源的效率，我们选择 DC/DC 电源稳压电路，分别得到稳定的 12V 和 5V 电压，同时在不同的供电回路，采用电解电容和无极性电容实现滤波，得到稳定可靠的无杂波的电源电压，并设计有 led 发光二极管实现不同电压等级的指示。其优点是电路结构简单，并对电源的高频干扰有较强的抑制作用。具体电路如图 2.2 所示。

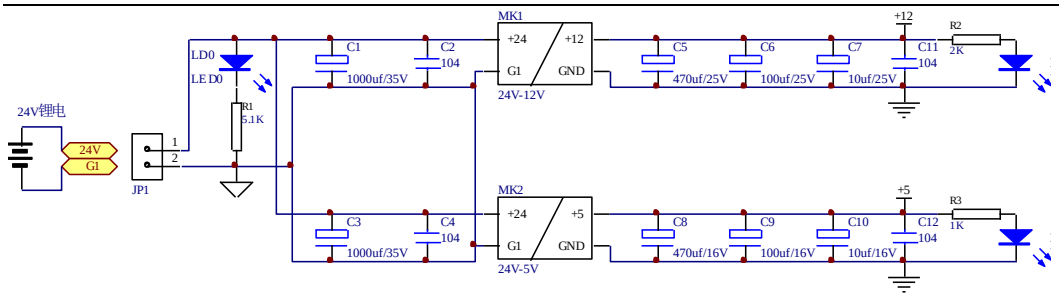


图 2.2 12V、5V 稳压电路

2.3.2 3.3V 稳压电路

我们所使用的微控制器 STM32F405 为 3.3V 供电。因为单片机是整个系统的核心，所以其电源的稳定性尤为重要，由 DC/DC 变换输出的 5V 电压经 AMS1117-3.3 稳压芯片输出产生 3.3V 电源。该电源芯片最大输出电流为 1A，可以实现低压差输出，其输出纹波小，并内部集成过流和过热保护功能，输出精度为 1%，较好的满足系统要求。这样既可以有效抑制电源纹波，又可以减小 AMS1117-3.3 的功耗，输出电压用电感和电容滤波，保证其工作的稳定性。电路如图 2.3 所示。

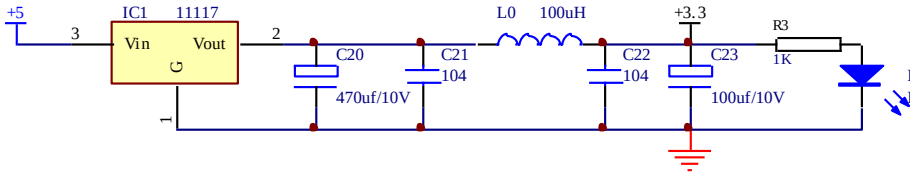


图 2.3 3.3V 稳压电路

2.3.2 5V 可控电源

可控电源如图 2.4 所示，由 P 沟道的 MOS 管 AO3401，PNP 型小功率三极管 9012 组成，控制舵机供电。系统上电时，单片机 PB5 脚输出高电平，禁止向舵机供电；运行时，PB5 脚输出低电平，舵机工作；当单片机检测到舵机堵转或过流时，PB5 脚输出高电平，以保护舵机。

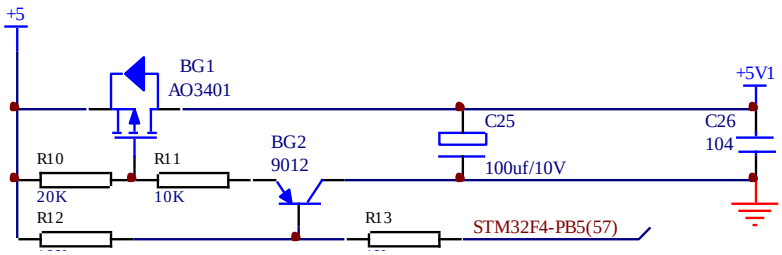


图 2.4 5V 可控电源电路

2.4 电机驱动模块

2.4.1 H 桥电路

底盘直流有刷电机由 PMOS、NMOS 功率开关管搭建的 H 桥电路驱动，如图 2.5 所示。我们采用 MOSFET 管和专用栅极驱动芯片设计，MOSFET 是由一种载流子导电的单极型器件，栅极驱动电流很小，是一种电压控制型器件，具有开关速度快、损耗低、驱动功率小、无二次击穿的优点。H 桥的两个上臂选用 IR 公司 IRF4905 型 P 沟道 MOSFET，选用两片 TLP250--MOS 栅极驱动芯片驱动；两个下臂选用 IR 公司 IRF3205 型 N 沟道 MOSFET，其驱动芯片选用 IR4426。此 H 桥电路输出额定电流达 40A，该设计大大提高了电机的驱动能力，H 桥电路接入四个并联的 0.5 欧姆的大功率电阻，实现 H 桥电路的过流检测和保护功能。

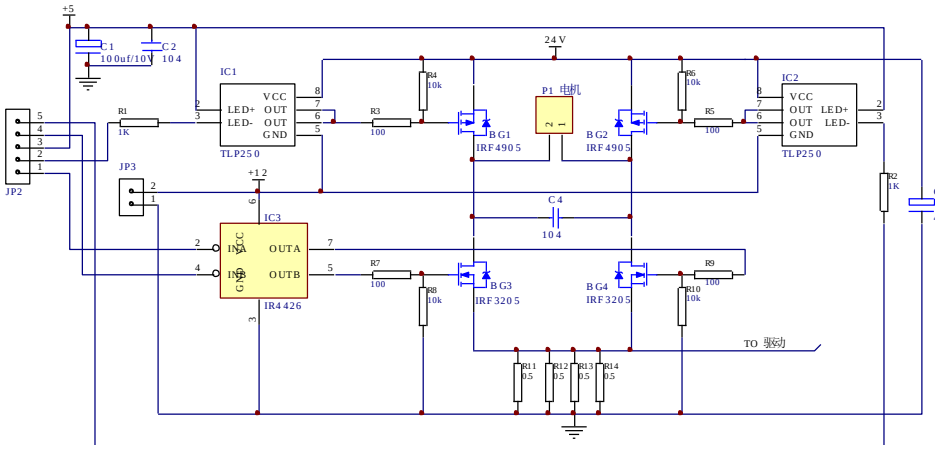


图 2.5 H 桥电路

H 桥电路可以很方便实现电机的正反转控制，如电机正向转动， H 桥对角线上的一对 MOS 管导通，例 BG1 和 BG4 导通，此时 BG2 和 BG3 必须截止。反之，BG2 和 BG3 导通，BG1 和 BG4 必须截止。控制时序图如图 2.6 所示。

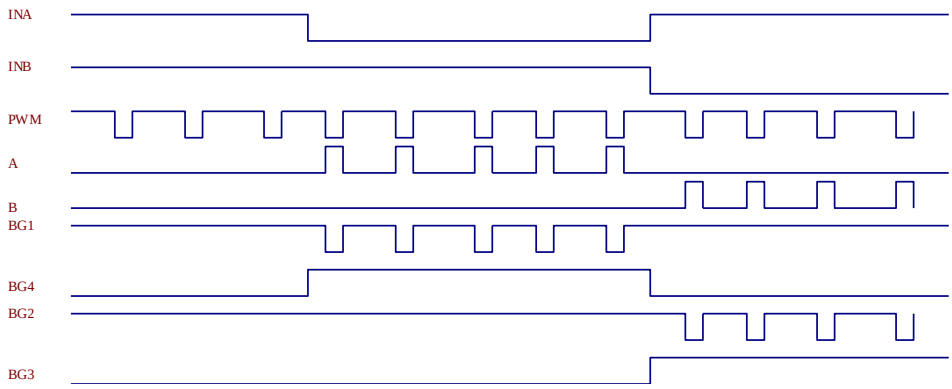


图 2.6 输出信号时序图

2.4.2 底盘电机驱动电路

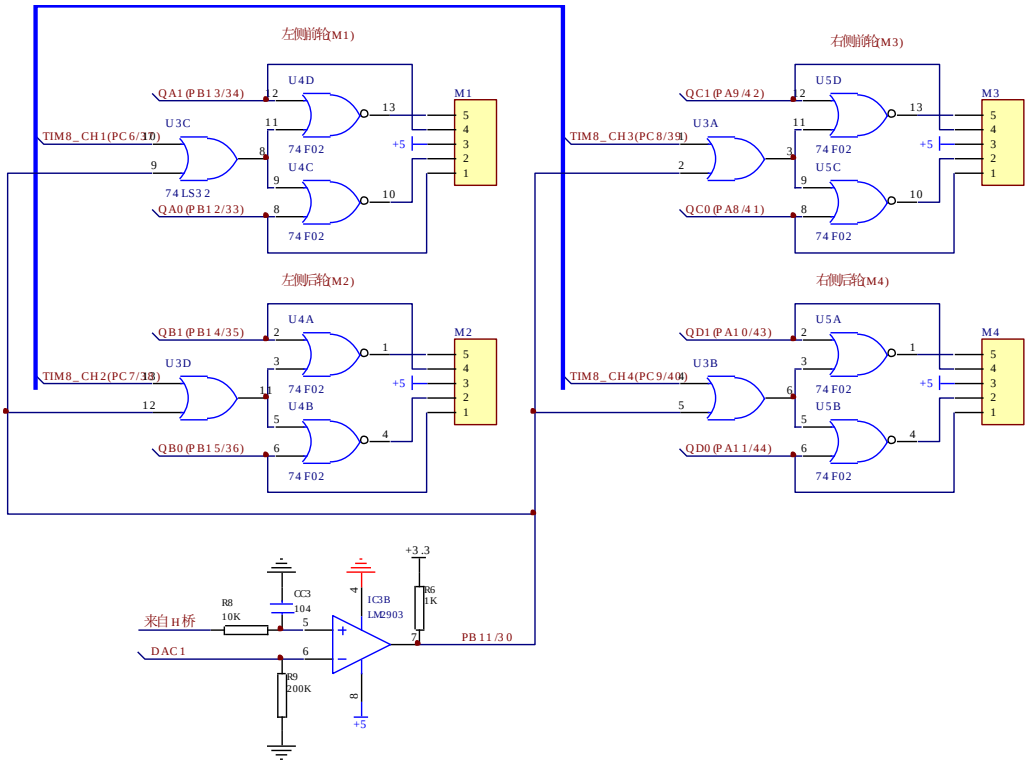


图 2.7 底盘电机逻辑输出驱动电路原理图

图 2.7 为底盘的四个电机逻辑输出驱动电路，以一个电机驱动为例说明其工作原理，若 PWM 输出高电平，单片机 PB11 脚也输出高电平，经过 74LS32 也得到高电平，PB13 若输出低电平，PB12 输出高电平，从而 BG1 导通，BG2 关闭，BG4 管导通，BG3 管关闭，实现了正转。保证了用两个个信号就能控制整个电路开关。电压比较器 LM2903 实现了电机的过流保护,单片机的 DAC 输出电机过流阈值(由多次实验确定)，当电机发生过流时，比较器输出高电平，封锁了 H 桥的

控制信号，同时被单片机捕获，进行保护处理。

2.5 舵机及拨弹电机控制

本系统设计了 4 路舵机和 2 路直流电机的控制，战车只使用了 2 个舵机和 1 个拨弹电机。舵机用于云台控制和弹仓的开/关盖。拨弹电机安装在弹仓内，防止弹丸堵塞。拨弹电机适应于+12V 或+24V 电压。

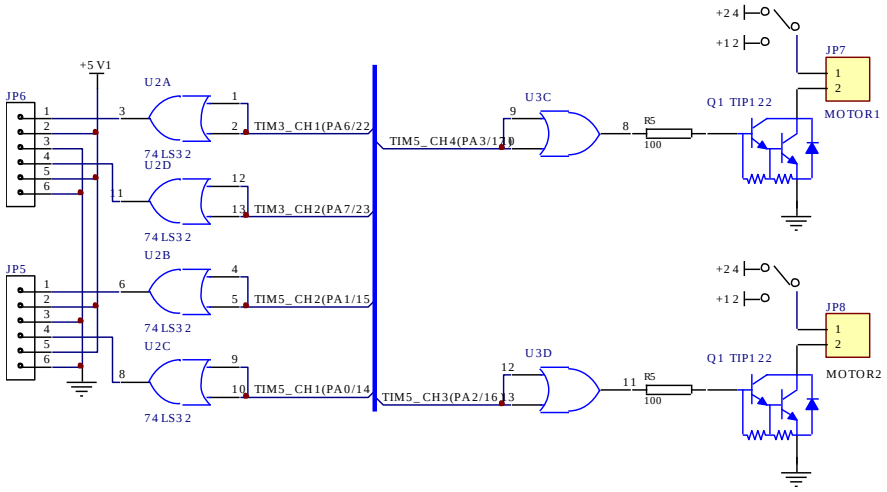


图 2.7 电机驱动电路原理图

两台射手和一台炮手使用相同的控制板，电路板是整个系统的核心，我们分别制作了电机驱动板和总控制板，大大提高了系统的稳定性，图 2.8 为安装在机器人车上的电路板实际图[2]。

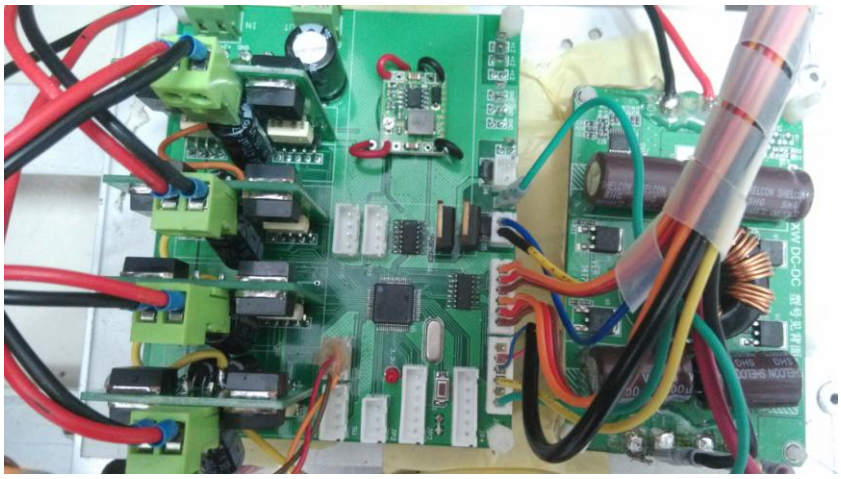


图 2.8 主控制电路板

3 控制算法说明

3.1 PID 控制算法

底盘驱动电机应快速响应和实现机器人战车的加速和减速过程，云台驱动电机可以使云台平滑的运动，有效地实现 P 轴、Y 轴和 R 轴灵活移动，所以电机能否快速稳定地输出将决定整个系统的特性。作为一个运动性很强的战斗机器人，电机驱动是至关重要的。以 PID 为基础的电机驱动控制能够很好地实现这个目标，并且以 MOS 为主的驱动电路可以实现较大功率的驱动[3]。

3.1.1 PID 原理

PID 控制器是一种线性控制器，它根据给定值与实际输出值构成控制偏差。将偏差的比例(P)、积分(I)和微分(D)通过线性组合构成控制量，对被控对象进行控制，故称 PID 控制器。在自动控制系统中，PID 控制器是得到广泛应用的一种控制方法。由于电机转速与电枢外加电压的大小基本上成正比，这就构成了 PID 调节的基础[5]。在采样时刻 t =i×T(T 为采样周期,i 为正整数)，模拟 PID 控制器调节规律控制原理图如图 3.1 所示。

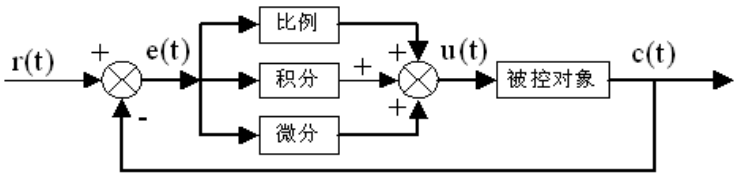


图 3.1PID 控制器原理框图

由于计算机只能识别数字量,不能对连续的控制算式直接进行运算,在计算机控制系统中必须对控制规律进行离散化的算法设计。方程如下：

$$e(k) = r(k) - \mathrm{c}(k) \tag{3.1}$$

$$\mathrm{u}(k) = K_p\{e(k) + \frac{T}{T_I}\sum_{j=0}^{k} e(j) + \frac{T_D}{T}[\mathrm{e}(k) - e(k\text{-}1)]\} \tag{3.2}$$

式中

k——采样序号，k = 0，1，2…；　　r(k)——第 k 次给定值；

c(k)——第 k 次实际输出值；　　u(k)—— 第 k 次输出控制量；

e(k)—— 第 k 次偏差；　　e(k-1)—— 第 k-1 次偏差；

K_P——比例系数； T_I——积分时间常数；

T_D——微分时间常数； T——采样周期。

当执行机构需要的是控制量的增量时，可由式(3.2)推导出提供增量的 PID 控制算式。由式(3.2)可以推出式(3.3)，式(3.2)减去式(3.3)可得式(3.4)。

$$u(k-1)=K_P\{e(k-1)+\frac{T}{T_I}\sum_{j=0}^{k-1}e(j)+\frac{T_D}{T}[e(k-1)-e(k-2)]\} \tag{3.3}$$

$$\begin{aligned}\Delta u(k)&=K_P\{[e(k)-e(k-1)]+\frac{T}{T_I}e(k)+\frac{T_D}{T}[e(k)-2e(k-1)+e(k-2)]\}\\&=K_P\Delta e(k)+K_I e(k)+K_D[\Delta e(k)-\Delta e(k-1)]\}\end{aligned} \tag{3.4}$$

式中 $\Delta e(k)=e(k)-e(k-1)$； $K_I=K_P\frac{T}{T_I}$； $K_D=K_P\frac{T_D}{T}$

公式(3.4)称为增量式 PID 控制算法，可以看出由于一般计算机控制系统采用恒定的采样周期 T，一旦确定了 K_p、T_I、T_D，只要使用前后三次测量值的偏差，即可由式(3.4)求出控制增量。

机器人微控制器以 STM32F405 为控制核心，机器人系统只有把直流电机的驱动控制、人机交换以及各种舵机的控制完美准确地组合在一起，才能实现平稳快速行驶。相比于其他的控制算法，增量式 PID 有着保持较小的超调和调控快速的优点，并且其算法简洁，理论完善成熟。在控制过程中，增量式 PID 的稳态误差对于控制系统的影响不是很大，但控制性能与其参数相关，在一般情况下，能够满足直流电机调速系统的相关要求。利用光电编码器采集脉冲数，计算出机器人当前速度，然后比较给定速度和当前速度的差值，通过增量式 PID 算法调节单片机的 PWM 波的输出，实现对战车直流电机的调速，以最短的时间完成速度调节，借此提高其控制效果。基于增量式 PID 控制器控制精度高，具有较高的可靠性，算法容易理解[6]。

根据增量方程并结合测控系统的具体情况,可以编出 PID 算法子程序。PID 运算过程中所有参数和计算值均以多字节浮点数表示,在控制过程中不断改变控制参数。系统运行中,通过定时器每间隔 T 秒中断一次,完成一次 PID 控制计算,从而不断调整被控参数,主程序中的 PWM 驱动模块根据控制参数而改变 PC6、PC7、PC8、PC9 四个 I/O 口输出值，调整 PWM 输出波形,完成实时控制任务。在一般

情况下,输出控制增量会在一个相对较小的范围内波动最后达到平滑控制。在程序中对输出增量大小规定了上限值 $u_i - \max$ 和下限值 $u_i - \min$,可以防止在突发情况下系统对电机的控制崩溃现象。实验证明，电机可灵敏快捷的达到预先设定的转速。

3.2 滤波算法

加速度计和陀螺仪等采用 MPU6050 微处理器实现对战车状态等的判别，加速度计对战车的加速度比较敏感，取瞬时值计算倾角误差比较大；而陀螺仪积分得到的角度不受战车加速度的影响，但是随着时间的增加积分漂移和温度漂移带来的误差比较大。所以这两个传感器正好可以弥补相互的缺点。

这里讲的互补滤波就是在短时间内采用陀螺仪得到的角度做为最优，定时对加速度采样来的角度进行取平均值来校正陀螺仪的得到的角度。也即短时间内用陀螺仪比较准确，以它为主；长时间用加速度计比较准确，这时候加大它的比重，这就是两个传感器的互补应用。

机器人战车的云台姿态解算中采用互补滤波算法，利用俯仰角 α 补偿 y 轴的角速度,平移角 β 补偿 x 轴的角速度。通过对比积分所得到的角度与加速度计(或电子罗盘)测得的角度,使用它们之间的偏差来改变陀螺仪的输出,从而使积分的角度逐步跟踪到加速度计所得到的角度。这样便完成了加速度计俯仰角、平移角的精确计算。

互补滤波算法的目的就是综合加速度计、电子罗盘和陀螺仪各自的频率响应优势,从频域角度对三个传感器数据进行融合，以减小测量和估计的误差。加速度计和电子罗盘的静态特性非常好，但动态特性较差，即在低频段动态响应特性好,在高频段动态响应特性差；陀螺仪虽然动态特性较好，但是存在温度漂移和零点漂移,因此在低频段动态响应特性差而高频段动态响应特性好。虽然陀螺仪、加速度计和电子罗盘都存在一定缺陷，不能在整频率范围中都表现出良好的动态响应特性。但是它们在频域上的动态响应特性是互补的，三个传感器的动态响应频率覆盖了整个频域范围。可见，从频域的角度来处理多个传感器的噪声是一种很有优势的滤波方式[7]。

采用互补滤波算法，通过融合加速度计或电子罗盘的数据信息，对陀螺仪的积分数据进行补偿，成功完成了机器人战车云台姿态角的精确解算。解算精度完

全满足云台姿态控制的需要，为机器人完成各方向射击任务提供了根本保证。同时，互补滤波算法的应用也有效地解决了陀螺仪的温度漂移和加速度计与电子罗盘的噪声干扰等。

4 系统安装与调试

4.1 弹仓的修改与安装

在 Robomasters 比赛分赛区上规定每个射手机器人可装 100 发子弹，我们使用一个底下口径 20cm 的漏斗作为弹仓，拨弹电机固定在漏斗上。在深圳决赛的官方规则也做了一些调整,其中最大的变化就是一个射手机器人的弹量可达 200 发，故必修修改原的弹仓装置。选用直径 100mm 的 PVC 管，高 10cm。自行设计了拨弹装置，3D 打印机打印的拨弹电机安装在弹仓内，防止弹丸堵塞。如图 4.1 所示

图 4.1 弹仓装置

Pvc 管材料普遍而且便宜，通过不断测试，确定拨轮的尺寸。为防止机器人在颠簸时弹丸飞出，我们在弹仓上面加上弹仓盖，用舵机控制它的开关。在经过多次的实验，此设计不堵弹，不漏弹，容易控制。

4.2 补给站的设计

在比赛中第一局中，射手机器人的初始载弹量为 0，需到补给站补给弹量，为了使两辆射手更快的补给到子弹，补给站的设计就尤为重要。为了节省时间，我们设计了 T 形补给站，即两辆射手同时补给。补给站的弹仓同样适用漏斗，漏斗的下方用托盖托住；在 T 形板后面安装了限位开关，当检测到射手触发到漏弹开关，控制舵机把托盖打开，此补给站设计简单灵活，补给快速。如图 4.2 所示。

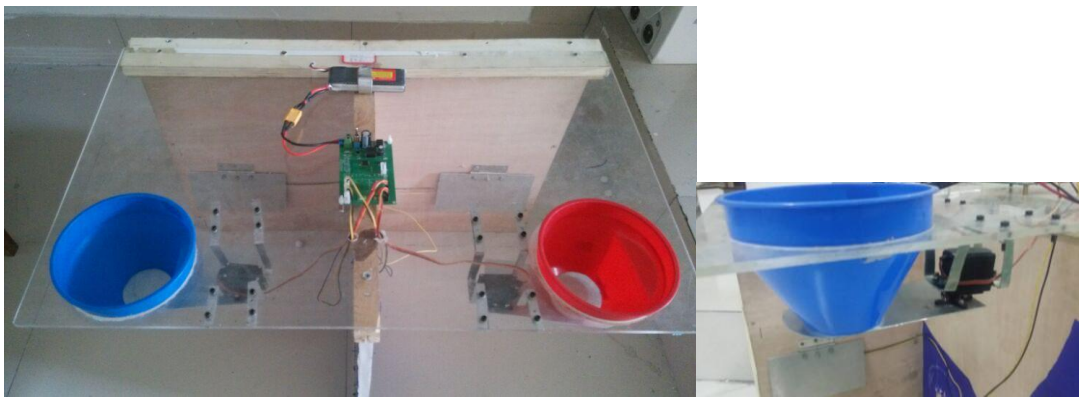


图 4.2 补给站

4.3 系统软件安装与调试

机器人战车主控板在开发中除必要的硬件外，同样离不开软件，我们采用 Keil μVision4 软件开发系统。

2009 年 2 月，Keil 公司发布了 Keil μVision4，Keil μVision4 引入灵活的窗口管理系统，使开发人员能够使用多台监视器，提供可在虚拟接口上随意放置窗口的完整控制能力。新的用户界面可以更好地利用屏幕空间和更有效地组织多个窗口，提供一个整洁，高效的环境来开发应用程序。新版本支持更多最新的 80C51 兼容芯片及 ARM 芯片，还添加了一些其他新功能，如系统查看器(System Viewer)窗口、多项目工作空间(Multi-Project Workspace)等。

Keil μVision4 集成开发环境（Integrated Development Environment，IDE）是一个基于 Windows 的开发平台，它包含高效的源代码编辑器、项目（Project）管理器和程序生成（MAKE）工具。Keil μVision4 支持所有的 80C51 嵌入式应用工具，它包括 C/C++编译器宏汇编器、连接/定位器和一个 HEX 文件生成器。Keil μVision4 通过以下特性加速 MCU 嵌入式应用系统的开发过程：

★ 全功能的源代码编辑器；

★ 器件库用来配置开发工具设置；

★ 项目管理器用来创建和维护项目；

★ 集成的 MAKE 工具可以汇编、编译和连接用户的嵌入式应用；

★ 所有开发工具的设置都是以对话框的形式出现的；

★ 具有真正的源代码级的对 CPU 和外围器件的调试器；

★ 高级 GDI 接口用来在目标硬件上进行软件调试以及和 Monitor-51 进行通

信；

★ 与开发工具手册、器件数据手册和用户指南有直接的链接。

4.3.1 C51 编译器和 A51 汇编器

源代码由 μVision4 创建，并被 C51 编译成 A51 汇编。编译器和汇编器从源代码生成可重定位的目标文件。

Keil C51 编译器完全遵照 ANSIC 语言标准，支持 C 语言的所有标准特性。另外，直接支持 80C51 结构的几个特性被添加里面。

Keil A51 宏汇编器支持 80C51 及其派生系列的全部指令集。

4.3.2 LIB51 库管理器

LIB51 库管理器允许从由编译器或汇编器生成的目标文件创建目标库。库是一种被特别地组织过并在以后可以被连接重用的对象模块。当连接器处理一个库时，仅仅那些被使用的目标模块才被真正使用.

4.3.3 μVision4 调试器

μVision4 源代码级调试器是一个理想的快速、可靠的程序调试器。此调试器包含一个高速模拟器，能够模拟整个 8051 系统，包括片上外围器件和外部硬件。当从器件库中选择器件时，这个器件的特性将自动配置。

μVision4 调试器为在实际目标板上测试程序提供了以下 2 种方法：

安装 MON51 目标监控器到目标系统并且通过 Monitor-51 接口下载程序；

利用高级的 GDI（AGDI）接口，把 μVision4 调试器绑定到目标系统。

Keil μVision4 集成开发环境中包括一个项目管理器，它可以使基于 80C51 内核的 MCU 应用系统设计变得简单。要创建一个应用，需要按下列步骤操作：

1.启动 Keil μVision4，新建一个项目文件并从器件库中选择一个器件；

2.新建一个源文件并把它加入到项目中；

3.设置目标硬件选项；

4.编译项目并生成可以编程到程序存储器的 HEX 文件；

5. 软件模拟调试及下载到 MCU 中进行仿真调试。

使用界面如下图：

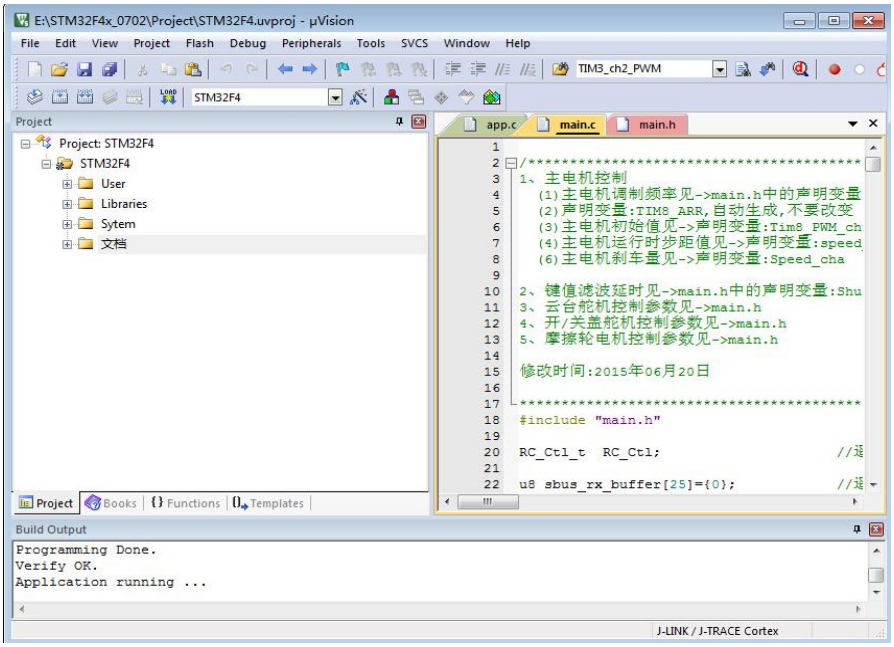


图 4.3 Keil µVision4 集成开发环境界面

这里可以看到，Keil µVision4 集成开发环境具有典型的 Windows 界面风格。整个编程界面主要包括菜单栏、工具栏、项目管理区、源代码工作区和输出信息窗口。

Keil µVision4 的菜单栏提供了项目操作、编辑操作、编译调试及帮助等各种常用操作。所有的操作基本上都可以通过菜单命令来实现。为了快速执行 Keil µVision4 的许多功能，有些菜单命令在工具栏上还具有工具条。为了更快速执行一些功能，Keil µVision4 提供了比工具栏上的工具条更为快捷的操作，即快捷键。在 Keil µVision4 集成开发环境中不仅提供了常用功能的默认快捷键，同时用户也可以根据自己的需要自定义快捷键。

总结

作为国内首档机器人竞赛，Robomasters2015 第十四届全国大学生机器人大赛不仅汇聚了当今科研领域尖端的机器人研发技术，还融入了炫酷的电子竞技元素。第一人称视角的真实打击、5V5 的实战对抗、哨兵机器人高空侦察以及充斥着未来科技感的竞技战场。为大学生提供科技创新平台，考验学生的综合素质和团队协作能力。

报告主要介绍了制作战车的整体思路，整个机器人制作涉及系统硬件设计、软件算法、车体机械结构的设计与调整和系统调试等多个方面。由于现有知识和经验的不足，我们在装配过程中遇到了很多问题，但是队员优势互补，齐心协力，事半功倍，在老师的指导下，高质、高效地完成任务。

通过这次机器人大赛，巩固了我们的理论知识、专业基础知识和基本技能，培养了我们独立思考和动手实践的能力，更培养了我们团队协作的精神。但针对我们在比赛中出现的问题，例如机器结构的不合理等，我们一定会努力改进。一份耕耘，一份收获，我们相信，在深圳总决赛，我们一定会取得优异的成绩！

参考文献

[1] 薛涛，宫辉，曾鸣等. 单片机与嵌入式系统开发方法[M].北京：清华大学出版社

[2] 蔡朝晖, 安向明, 张宇. C#程序案例设计教程[M]．北京:清华大学出版社,2012

[3] Yamaguch，Advanced Automated Fingerprint Identification system[C].32nd Annual 1998 International camahan conference on Security Technology，1998，10:154-157

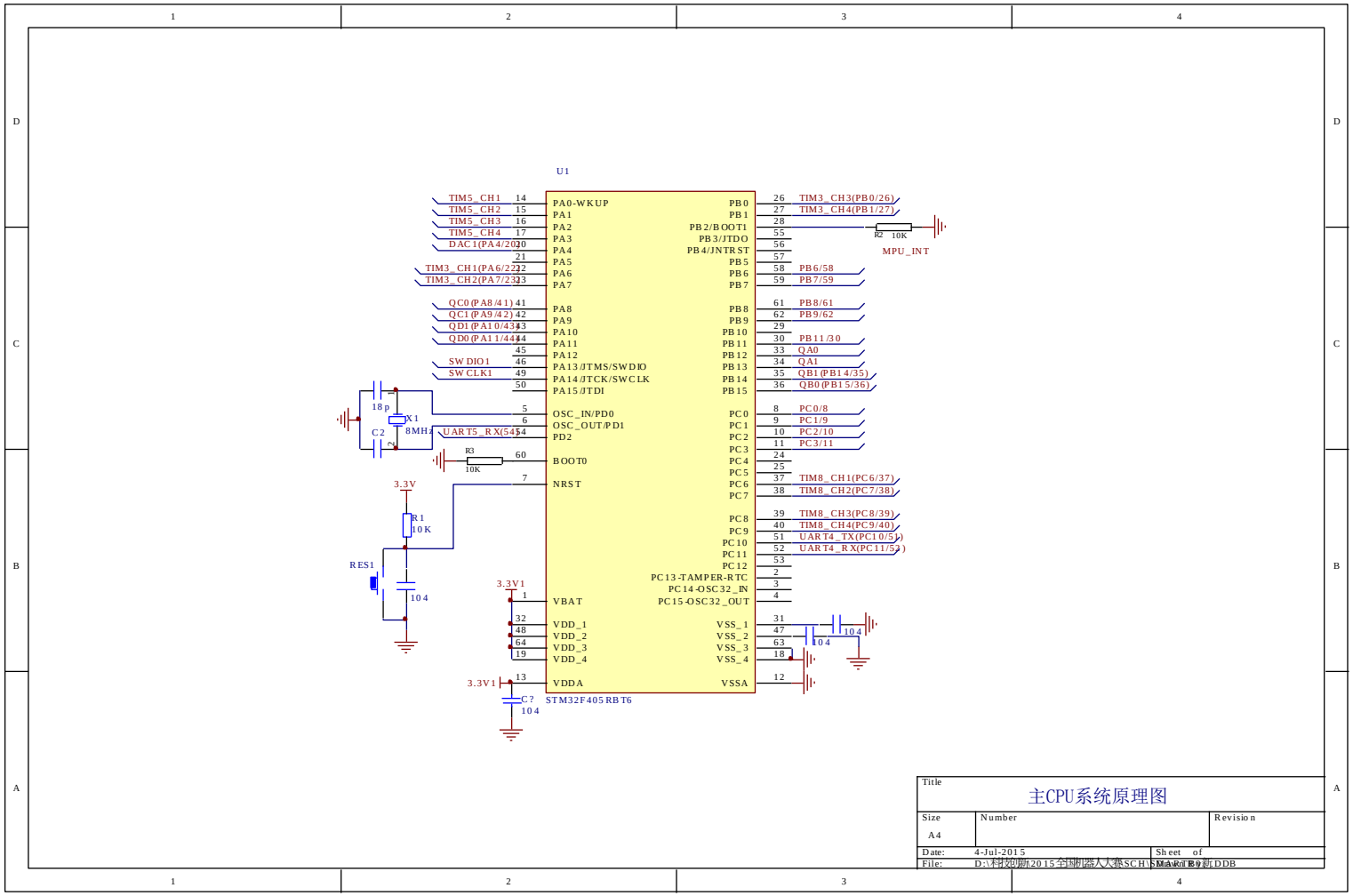
[4] 谭浩强. C 语言程序设计，北京：清华大学出版社，1999

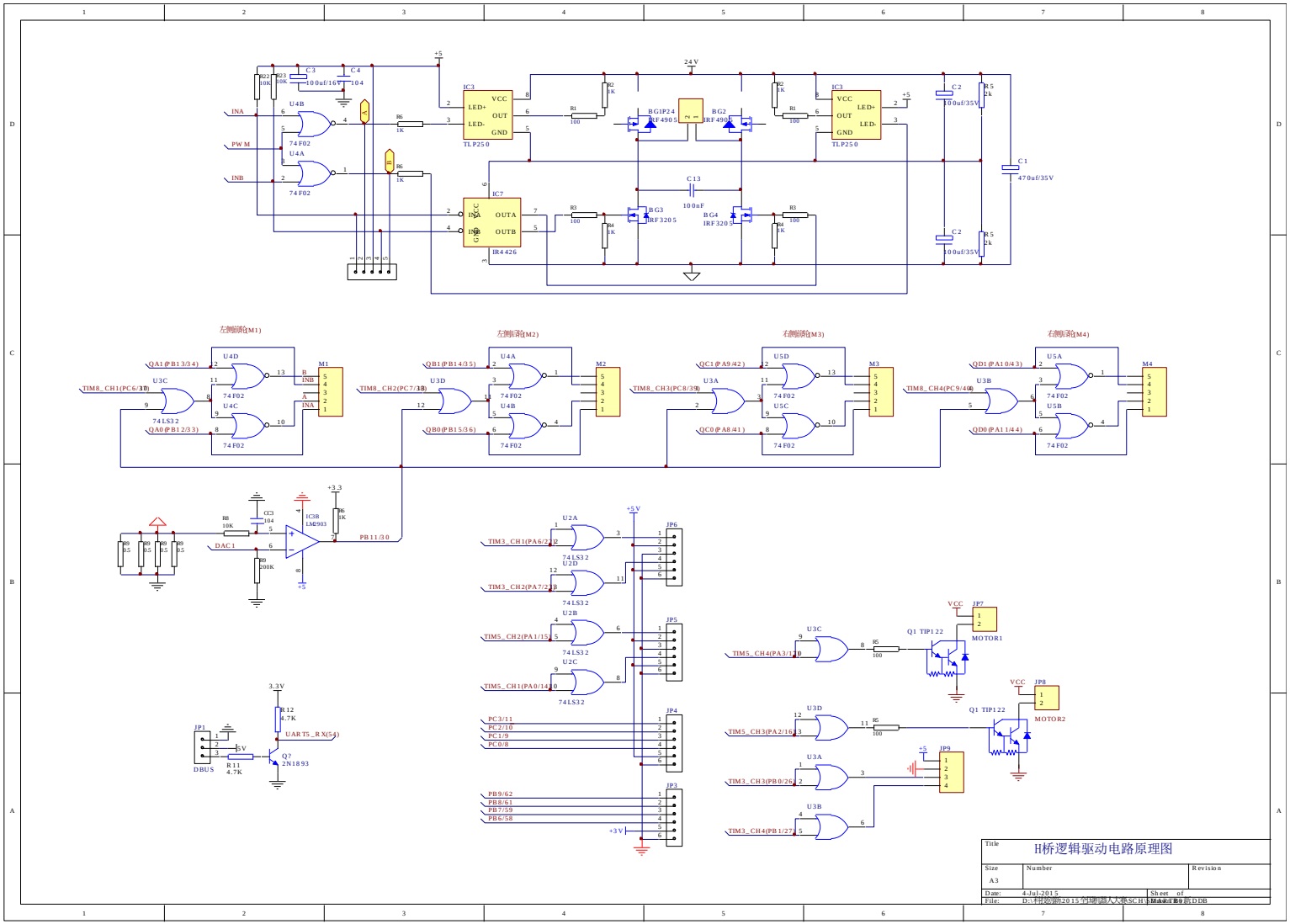
[5] 谢世杰，陈生潭，楼顺天. 数字 PID 算法在无刷直流电机控制器中的应用[J]，现代电子技术，2004 27(2):59-61

[6] 范咏峰，李平. 浅析 PID 参数整定. 中国仪器仪表，2002，vol2，No.3:24-28

[7] 张陪仁，杨兴明. 机器人系统设计与算法[M]. 中国科学技术大学出版社，2008

附录 I 部分硬件电路图


U1
STM32F405RBT6
PA0-WKUP
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PA8
PA9
PA10
PA11
PA12
PA13/JTMS/SWDIO
PA14/JTCK/SWCLK
PA15/JTDI
PB0
PB1
PB2/BOOT1
PB3/JTDO
PB4/JNTRST
PB5
PB6
PB7
PB8
PB9
PB10
PB11
PB12
PB13
PB14
PB15
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10
PC11
PC12
PC13-TAMPER-RTC
PC14-OSC32_IN
PC15-OSC32_OUT
VBAT
VDD_1
VDD_2
VDD_3
VDD_4
VDDA
VSS_1
VSS_2
VSS_3
VSS_4
VSSA
OSC_IN/PD0
OSC_OUT/PD1
PD2
BOOT0
NRST
TIM5_CH1
TIM5_CH2
TIM5_CH3
TIM5_CH4
DAC1(PA4/20)
TIM3_CH1(PA6/22)
TIM3_CH2(PA7/23)
QC0(PA8/41)
QC1(PA9/42)
QD1(PA10/43)
QD0(PA11/44)
SWDIO1
SWCLK1
UART5_RX(54)
TIM3_CH3(PB0/26)
TIM3_CH4(PB1/27)
MPU_INT
PB6/58
PB7/59
PB8/61
PB9/62
PB11/30
QA0
QA1
QB1(PB14/35)
QB0(PB15/36)
PC0/8
PC1/9
PC2/10
PC3/11
TIM8_CH1(PC6/37)
TIM8_CH2(PC7/38)
TIM8_CH3(PC8/39)
TIM8_CH4(PC9/40)
UART4_TX(PC10/51)
UART4_RX(PC11/52)
3.3V
3.3V1
18p
8MHz
X1
C2
R1
10K
R2
R3
RES1
104
C?
Title
主CPU系统原理图
Size
A4
Number
Revision
Date: 4-Jul-2015
Sheet of
File: D:\科艾斯\2015全国机器人大赛\SCH\STM32F405RBT6.DDB


H桥逻辑驱动电路原理图
Size A3
Number
Revision
Date: 4-Jul-2015
Sheet of
File: D:\科技部2015全国机器人大赛\SCH\Mastar.DDB
左侧驱动(M1)
左侧驱动(M2)
右侧驱动(M3)
右侧驱动(M4)
IC3 TLP250
IC7 IR4426
BG1P24 IRF4905
BG2 IRF4905
BG3 IRF3205
BG4 IRF3205
74F02
74LS32
LM903
TIP122
MOTOR1
MOTOR2
2N1893
DBUS
DAC1
UART5_RX(54)
TIM8_CH1(PC6/37)
TIM8_CH2(PC7/38)
TIM8_CH3(PC8/39)
TIM8_CH4(PC9/40)
TIM3_CH1(PA6/22)
TIM3_CH2(PA7/23)
TIM5_CH2(PA1/15)
TIM5_CH1(PA0/14)
TIM5_CH4(PA3/17)
TIM5_CH3(PA2/16)
TIM3_CH3(PB0/26)
TIM3_CH4(PB1/27)
PC3/11
PC2/10
PC1/9
PC0/8
PB9/62
PB8/61
PB7/59
PB6/58
INA
INB
PWM

附录 II 部分源代码

```
#include "main.h"
RC_Ctl_t    RC_Ctl;                          //遥控器接收数据机构体
u8 sbus_rx_buffer[25]={0};                   //遥控器原始数据
u8 STM32_reset=0;
u8 ContrlA;                                  /*接收到 DR16 置 1,处理后清 0*/
u8 ContrlB;                                  /*过流保护置
/*主电机变量*/
u8 move;                                     /*=0~6,指示车体运行方向*/
u16 PWM_initLQ;
u16 PWM_initRQ;
u16 PWM_initLH;
u16 PWM_initRH;
u16 PWM_LQ;
u16 PWM_RQ;
u16 PWM_LH;
u16 PWM_RH;
u16 PWM_LR;
u16 Ctl_V;                                   /*--键盘*/
int16_t Ctl_16X = 0;                         /*--鼠标 X 轴位移*/
int16_t Ctl_16Y = 0;                         /*--鼠标 Y 轴位移*/
u16 Ctl_16V[8]={0};                          /*--键盘*/
u8 V_cnt = 0x00;
u16 DAC1_data = 0x0300;
/*************************************************************
函数名称:main(void)
功能:    系统 main(void)主程序
输入:
```

```
输出:
返回:
**********************************************************/
int main(void)
{
    RCC_Configuration();
    IO_Configuration();
    /*置 1--关闭*/
    GPIOB->BSRRL = 0xf000;
    GPIOA->BSRRL = 0x0f00;
        PWM_Configuration();
    move = 0x00;
    TIM8->CCR1 = 0x0000;
    TIM8->CCR2 = 0x0000;
    TIM8->CCR3 = 0x0000;
    TIM8->CCR4 = 0x0000;
    TIM3_PWM();                 /*初始化 TIM3--开盖舵机,云台舵机*/
    TIM5_PWM();                 /*初始化 TIM5--摩擦轮电机*/
  UART5_int();                  //遥控器 DBUS  初始化
    DAC_int();
    EXTI_Configuration();
    NVIC_Config();
        delay_ms(200);
        ContrlA = 0x00;         /*接收到 DR16 置 1,处理后清 0*/
    ContrlB = 0x00;             /*过流保护置 1,常态为 0*/
    move = 0x00;
    while(1)
    {
        if (ContrlB == 0x00)
        {
```

```
            if (ContrlA != 0x00)
            {
                ContrlA = 0x00;
                ESC_P_Y_PID();
                motor_control();
    //解读 RD16 的值
            }
        }
        else
        {
            delay_ms(50);
            /*延时 50ms*/
            ContrlB = 0x00;
        }
    }
}
/************************************************************
函数名称:RCC_Configuration()
功能:    初始化系统时钟
输入:
输出:
返回:
************************************************************/
void     RCC_Configuration()
{
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIO
B|RCC_AHB1Periph_GPIOC|
    RCC_AHB1Periph_GPIOD|RCC_AHB1Periph_DMA1,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8|RCC_APB2Periph_USART1,
ENABLE);
```

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2|RCC_APB1Periph_TIM3|RC
C_APB1Periph_TIM4|
    RCC_APB1Periph_TIM5|RCC_APB1Periph_TIM6|

    RCC_APB1Periph_DAC|RCC_APB1Periph_UART5,ENABLE);
}
/*************************************************************
函数名称:NVIC_Config(void)
功能:初始化
输入:
输出:
返回:
*************************************************************/
void NVIC_Config(void)
{
  NVIC_InitTypeDef    nvic;

  /*向量表位置和偏移*/
    NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x00000000);
    /*配置优先级和次优先级*/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    nvic.NVIC_IRQChannel = DMA1_Stream0_IRQn;
    nvic.NVIC_IRQChannelPreemptionPriority = 0;
    nvic.NVIC_IRQChannelSubPriority = 1;
    nvic.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&nvic);
    nvic.NVIC_IRQChannel = EXTI15_10_IRQn;
    nvic.NVIC_IRQChannelPreemptionPriority = 0;
    nvic.NVIC_IRQChannelSubPriority = 0;
    nvic.NVIC_IRQChannelCmd = ENABLE;
```

```
    NVIC_Init(&nvic);
}
/*************************************************************
函数名称:IO_Configuration()
功能:    初始化 I/O
输入:
输出:
返回:
*************************************************************/
void IO_Configuration()
{
    GPIO_InitTypeDef gpio;
        /*左侧电机正反控制*/
  gpio.GPIO_Pin = GPIO_Pin_12 |GPIO_Pin_13|GPIO_Pin_14 | GPIO_Pin_15;
    gpio.GPIO_Mode = GPIO_Mode_OUT;
    gpio.GPIO_OType = GPIO_OType_OD;
    gpio.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOB,&gpio);
    /*右侧电机正反控制*/
    gpio.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9|GPIO_Pin_10 | GPIO_Pin_11;
  GPIO_Init(GPIOA,&gpio);

    /*PWM 驱动底盘电机
        TIM8_CH1(PC6/37)--左侧前轮
        TIM8_CH2(PC7/38)--左侧后轮
        TIM8_CH3(PC8/39)--右侧前轮
        TIM8_CH4(PC9/40)--右侧后轮*/
    gpio.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7|GPIO_Pin_8 | GPIO_Pin_9;
  gpio.GPIO_Mode = GPIO_Mode_AF;
    gpio.GPIO_OType = GPIO_OType_PP;
```

```
gpio.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOC,&gpio);
  /*1、模拟舵机控制(插座 JP5)
        TIM5_CH1(PA0/14)
        TIM5_CH2(PA1/15)
     2、数字舵机控制(插座 JP6)
        TIM3_CH1(PA6/22)--开盖舵机
        TIM3_CH2(PA7/23)--云台舵机*/
gpio.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_6|GPIO_Pin_7;
  gpio.GPIO_Mode = GPIO_Mode_AF;
  gpio.GPIO_OType = GPIO_OType_PP;
  gpio.GPIO_Speed = GPIO_Speed_25MHz;
  GPIO_Init(GPIOA,&gpio);
     /*直流电机控制
     TIM5_CH3(PA2/16)--插座 JP8
     TIM5_CH4(PA3/17)--插座 JP7--拨弹*/
gpio.GPIO_Pin = GPIO_Pin_2|GPIO_Pin_3;
  gpio.GPIO_Mode = GPIO_Mode_OUT;
  gpio.GPIO_OType = GPIO_OType_PP;
  gpio.GPIO_Speed = GPIO_Speed_25MHz;
  GPIO_Init(GPIOA,&gpio);
  /*UART4--插座 JP4*/
  gpio.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
  gpio.GPIO_Mode = GPIO_Mode_AF;
gpio.GPIO_OType = GPIO_OType_PP;
gpio.GPIO_Speed = GPIO_Speed_25MHz;
gpio.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOC,&gpio);

  /*UART5--插座 JP1*/
```

```
    gpio.GPIO_Pin = GPIO_Pin_2;
    gpio.GPIO_Mode = GPIO_Mode_AF;
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_Speed = GPIO_Speed_25MHz;
    gpio.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOD,&gpio);
    /*过流保护*/
  gpio.GPIO_Pin = GPIO_Pin_11;
    gpio.GPIO_Mode = GPIO_Mode_IN;
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_Speed = GPIO_Speed_100MHz;
    gpio.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOB, &gpio);
    /*DAC1*/
  gpio.GPIO_Pin = GPIO_Pin_4;
    gpio.GPIO_Mode = GPIO_Mode_AF;
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_Speed = GPIO_Speed_25MHz;
    GPIO_Init(GPIOA,&gpio);
}

/*************************************************************
函数名称:EXTI_Configuration(void)
功能:   初始化线中断
            PB11---过流保护
输入:
输出:
返回:
*************************************************************/
void EXTI_Configuration(void)
```

```
{
   EXTI_InitTypeDef* EXTI_InitStruct;


     EXTI_InitStruct->EXTI_Line = EXTI_Line11;
   EXTI_InitStruct->EXTI_Mode = EXTI_Mode_Event;
   EXTI_InitStruct->EXTI_Trigger = EXTI_Trigger_Rising;
   EXTI_InitStruct->EXTI_LineCmd = DISABLE;
     EXTI_Init(EXTI_InitStruct);
}
/*************************************************************
函数名称:PWM_Configuration(void)
功能:    初始化 TIM8
          TIM8CLK = 168 MHz, Prescaler = 14, TIM8_counter_clock =12 MHz
          TIM8 frequency=TIM8_counter_clock/TIM8_ARR
             TIM8_ARR = TIM8_counter_clock/TIM8 frequency
          例 1：TIM8 frequency=2000Hz
                    TIM8->ARR=12000000/2000=6000
        PWMA---(PC6---TIM8_CH1)--左前轮
        PWMB---(PC7---TIM8_CH2)--左后轮
        PWMC---(PC8---TIM8_CH3)--右前轮
        PWMD---(PC9---TIM8_CH4)--右后轮
输入:
输出:
返回:
*************************************************************/
void PWM_Configuration(void)
{
    TIM_TimeBaseInitTypeDef     tim;
    TIM_OCInitTypeDef               oc;
    GPIO_PinAFConfig(GPIOC,GPIO_PinSource6, GPIO_AF_TIM8);
```

```
    GPIO_PinAFConfig(GPIOC,GPIO_PinSource7, GPIO_AF_TIM8);
    GPIO_PinAFConfig(GPIOC,GPIO_PinSource8, GPIO_AF_TIM8);
    GPIO_PinAFConfig(GPIOC,GPIO_PinSource9, GPIO_AF_TIM8);

    tim.TIM_Prescaler = 13;
    tim.TIM_CounterMode = TIM_CounterMode_Up;
    tim.TIM_Period = TIM8_ARR;
    tim.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInit(TIM8,&tim);

    oc.TIM_OCMode = TIM_OCMode_PWM2;
    oc.TIM_OutputState = TIM_OutputState_Enable;
    oc.TIM_OutputNState = TIM_OutputNState_Disable;
    oc.TIM_Pulse = 0;
    oc.TIM_OCPolarity = TIM_OCPolarity_High;
    oc.TIM_OCNPolarity = TIM_OCNPolarity_Low;
    oc.TIM_OCIdleState = TIM_OCIdleState_Set;
    oc.TIM_OCNIdleState = TIM_OCNIdleState_Reset;
    TIM_OC1Init(TIM8,&oc);
    TIM_OC2Init(TIM8,&oc);
    TIM_OC3Init(TIM8,&oc);
    TIM_OC4Init(TIM8,&oc);
    TIM_OC1PreloadConfig(TIM8,TIM_OCPreload_Enable);
    TIM_OC2PreloadConfig(TIM8,TIM_OCPreload_Enable);
    TIM_OC3PreloadConfig(TIM8,TIM_OCPreload_Enable);
    TIM_OC4PreloadConfig(TIM8,TIM_OCPreload_Enable);
    TIM_ARRPreloadConfig(TIM8,ENABLE);
    TIM_CtrlPWMOutputs(TIM8,ENABLE);
    TIM_Cmd(TIM8,ENABLE);
}
```

```
/************************************************************
函数名称:TIM3_PWM(void)
功能:    初始化 TIM3
         TIM3CLK = 84 MHz, Prescaler = 42, TIM3_counter_clock =2 MHz
         TIM3 frequency=TIM3_counter_clock/TIM3_ARR
           TIM3_ARR = TIM3_counter_clock/TIM3 frequency
         例 1：TIM3 frequency=400Hz
                  TIM3->ARR=2000000/400=5000
        TIM3_CH1(PA6/22)--开盖舵机
        TIM3_CH2(PA7/23)--云台舵机
输入:
输出:
返回:
************************************************************/
void TIM3_PWM(void)
{
    TIM_TimeBaseInitTypeDef     tim;
    TIM_OCInitTypeDef               oc;
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource6, GPIO_AF_TIM3);
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource7, GPIO_AF_TIM3);
    tim.TIM_Prescaler = 41;
    tim.TIM_CounterMode = TIM_CounterMode_Up;
    tim.TIM_Period = 5000;
    tim.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInit(TIM3,&tim);
    oc.TIM_OCMode = TIM_OCMode_PWM1;
    oc.TIM_OutputState = TIM_OutputState_Enable;
    oc.TIM_Pulse = TIM3_ch1_on;                                    //开盖
    oc.TIM_OCPolarity = TIM_OCPolarity_High;
    oc.TIM_OCIdleState = TIM_OCIdleState_Set;
```

```
    TIM_OC1Init(TIM3,&oc);
    oc.TIM_Pulse = TIM3_ch2_int;
    TIM_OC2Init(TIM3,&oc);
    TIM_OC1PreloadConfig(TIM3,TIM_OCPreload_Enable);
    TIM_OC2PreloadConfig(TIM3,TIM_OCPreload_Enable);
                    TIM_Cmd(TIM3,ENABLE);
}
/*************************************************************
函数名称:TIM5_PWM(void)
功能:   初始化 TIM5
        TIM5CLK = 84 MHz, Prescaler = 42, TIM5_counter_clock =2 MHz
        TIM5 frequency=TIM5_counter_clock/TIM5_ARR
          TIM5_ARR = TIM5_counter_clock/TIM5 frequency
        例 1：TIM5 frequency=400Hz
                TIM5->ARR=2000000/400=5000
      TIM5_CH1(PA0/14)--摩擦轮
      TIM5_CH2(PA1/15)--

输入:
输出:
返回:
*************************************************************/
void TIM5_PWM(void)
{
    TIM_TimeBaseInitTypeDef     tim;
    TIM_OCInitTypeDef             oc;
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource0, GPIO_AF_TIM5);
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource1, GPIO_AF_TIM5);

    tim.TIM_Prescaler = 41;
```

```
    tim.TIM_CounterMode = TIM_CounterMode_Up;
    tim.TIM_Period = 5000;
    tim.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInit(TIM5,&tim);
    oc.TIM_OCMode = TIM_OCMode_PWM1;
    oc.TIM_OutputState = TIM_OutputState_Enable;
    oc.TIM_Pulse = Tim5_ch1_chu;   //摩擦轮电机(初值频率 400Hz,初始占空比
40%)
    oc.TIM_OCPolarity = TIM_OCPolarity_High;
    oc.TIM_OCIdleState = TIM_OCIdleState_Set;
    TIM_OC1Init(TIM5,&oc);

    oc.TIM_Pulse = 0;
    TIM_OC2Init(TIM5,&oc);

    TIM_OC1PreloadConfig(TIM5,TIM_OCPreload_Enable);
    TIM_OC2PreloadConfig(TIM5,TIM_OCPreload_Enable);

    TIM_Cmd(TIM5,ENABLE);
}
/*************************************************************
函数名称:TIM6_Configuration(void)
功能:    定时 100ms,定时
             TIM6_CLK=84MHz。42000 分频,周期 0.5ms
             自动重装载寄存器 ARR=50ms/0.5ms=100
输入:
输出:
返回:
*************************************************************/
void TIM6_Configuration(void)
```

```
{
    TIM6->CR1 = 0x0000;
  /*配置 TIM2  预分频器: 42000 分频*/
  TIM6->PSC = 41999;
    /*配置 TIM4 自动重装载寄存器:      0.5ms*100=50mS */
    TIM6->ARR = 0x0064;
    TIM_ITConfig(TIM6, TIM_IT_Update,ENABLE);
}
/*************************************************************
函数名称:DAC_int(void)
功能:   初始化 DAC
输入:
输出:
返回:
*************************************************************/
void DAC_int(void)
{
    DAC_InitTypeDef DAC_InitStruct;
  /* Initialize the DAC_Trigger member */
  DAC_InitStruct.DAC_Trigger = DAC_Trigger_None;
  /* Initialize the DAC_WaveGeneration member */
  DAC_InitStruct.DAC_WaveGeneration = DAC_WaveGeneration_None;
  /* Initialize the DAC_LFSRUnmask_TriangleAmplitude member */
  DAC_InitStruct.DAC_LFSRUnmask_TriangleAmplitude                    =
DAC_LFSRUnmask_Bit0;
  /* Initialize the DAC_OutputBuffer member */
  DAC_InitStruct.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
    DAC_Init(DAC_Channel_1, &DAC_InitStruct);
  /* Enable DAC Channel1 */
  DAC_Cmd(DAC_Channel_1, ENABLE);
```

```
    DAC_SetChannel1Data(DAC_Align_12b_R, DAC1_data);
}
```