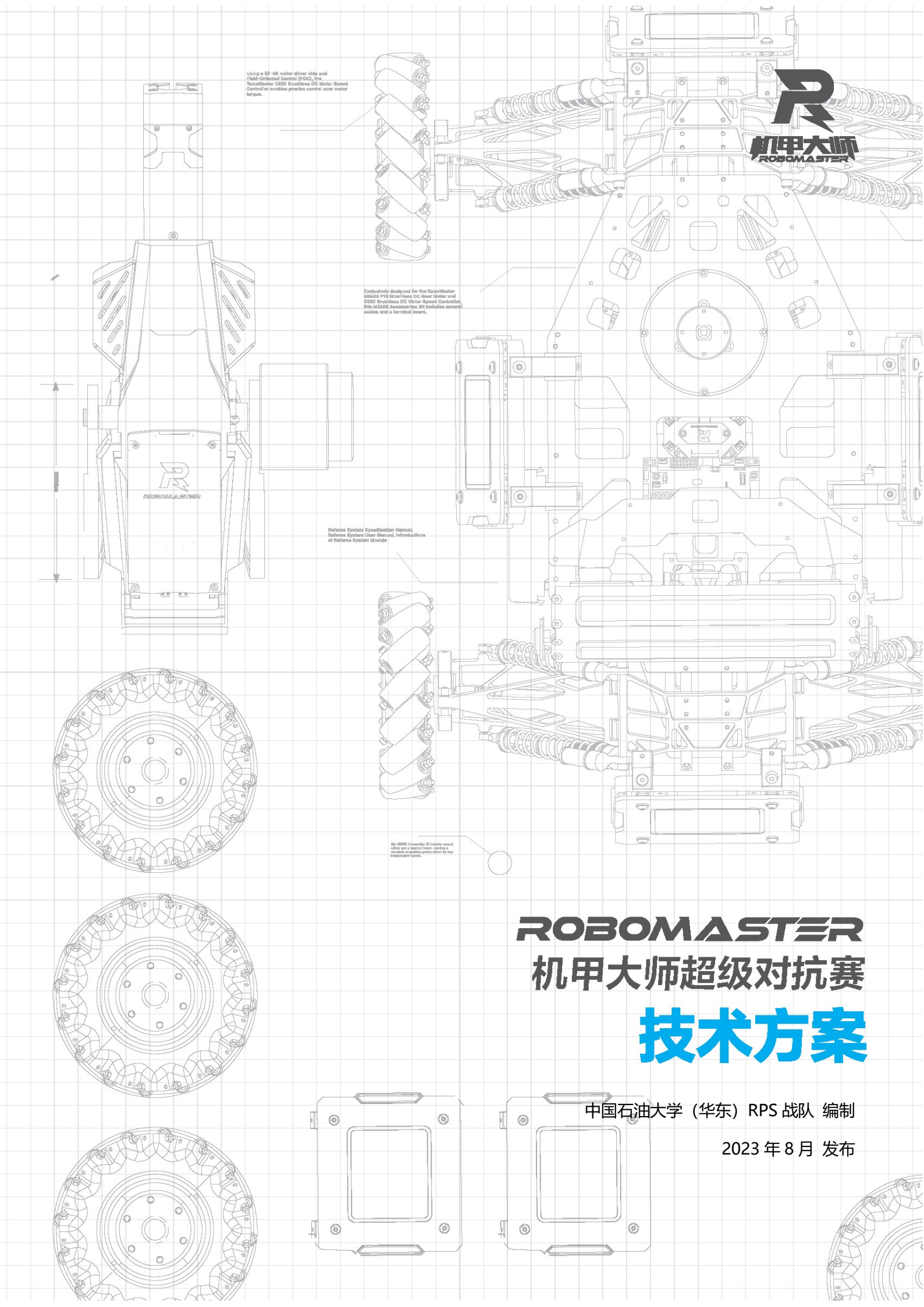


ROBOMASTER
机甲大师超级对抗赛
技术方案

中国石油大学（华东）RPS 战队 编制

2023 年 8 月 发布

前言

本文档由中国石油大学（华东）RPS 战队编制，适用于 RoboMaster 2023 机甲大师超级对抗赛。主要撰写人员包括：

模块	撰写人员 1	撰写人员 2
机械	徐羽成	张博洋
硬件	邢可	
软件	赵栋林	陶承誉
算法	高源	叶裕杰
其他	姚翊	潘睿扬

目录

1. 概述

1.1 背景与目标

步兵主要以发射小弹丸对敌方造成伤害，具有高射速，单发伤害低的特点。从上一届比赛的视频及比赛机制来看，步兵大多被运用于摧毁快速移动目标或移动相对不规律的目标上，这使得步兵机器人可以在短时间内快速推进，破坏敌方防御机制。具体而言，步兵机器人在比赛开始时需迅速抵达预定战术位置，并快速扫描赛场信息，获取敌军机器人位置情报，协助队友获得更全面的战局认知。在整场比赛中，步兵机器人充当流动性高的角色，实时监视并即时传递情报，与各种部队协同调整战术，从而为队友创造更多操作空间。步兵机器人虽然在战场上发挥了中流砥柱的作用，但是受制于步兵机器人发射机构的特点，使得在进行远距离，高血量目标击杀时占劣势，往往没有己方英雄机器人辅助时长时间造成低伤害。因此大力发展步兵机器人发弹稳定性功能，提高步兵机器人的通过性和机动性成了充分发挥该兵种战场性能的最佳方式。

1.2 其它学校机器人分析综述

从往届比赛和论坛开源的步兵机器人来看，在步兵供弹方案的设计上各有千秋，但主要还是保证步兵发弹的稳定性的基础上进行其他性能的提高。同时，在底盘的悬挂设计上，虽然大部分学校都是基于华南虎战队的开源舵轮步兵进行改造，但在保持云台稳定和飞坡稳定性上确实达到了一定的的效果，例如华南农业大学采用 GM6020 电机直接驱动转向轴，确保准确转向，避免了传统齿轮带来的误差。但因对 6020 电机轴向压力处理，使用多根铜柱替代 yaw 轴连接，导致重量较大、底盘高、算法繁重，在起伏路段表现不佳。桂林电子科技大学同样使用 GM6020 电机，搭配低角度气弹簧悬挂，节省空间且具避震效果。然而，避震效果不佳，转向时易偏离轴线，不适用于盲道等特殊场景。在面对如哈工程则采用的轮腿机器人时面对对面的数值压制时必须依靠舵轮全向移动的优势并保证火力予以回击。由此可以看出实现步兵发射稳定功能具有很强烈需求。

1.3 机器人功能定义

- 底盘功能设计
  - ◆ 全地形下不会出现翻车

  指标：以任意速度下比赛场地台阶不会出现翻车和底盘剐蹭卡住。
  - ◆ 有小陀螺功能

  指标：在车体自旋的同时可以全向运动
  - ◆ 可以实现飞坡

  指标：稳定飞坡达到 100%的通过率
  - ◆ 机动性强

  指标：四驱独立悬挂且不易损坏
- 云台设计
  - ◆ 实现两轴云台，保证射击过程中云台稳定不抖动
- 射击系统
  - ◆ 带视觉的辅助瞄准

  指标：在 4m 以内面对非小陀螺目标可以达到自锁击杀且命中率高于 80%
  - ◆ 射击系统稳定

  指标：弹丸初速波动低于 0.2m/s，连续射击 1000 次不出现卡弹现象，射速、射频可以稳定控制在官方设定的上限值
- 其他功能
  - ◆ 便于维护

  指标：大部分的零件或模块可以在 3 分钟能更换，任意一条线路 3 分钟内可以完成检测和更换。
  - ◆ 整体外形美观

1.4 机器人核心参数

名称		参数	
整车重量		22.3kg	
外形尺寸		长*宽*高 539mm*539mm*490mm	
工作电压		24V	
功耗		7W	
电容容量		2050J	
车辆最大移动速度		3.4m/s	
主要传感器	陀螺仪	型号	CH100
		工作电压	3.3V - 5V
		温度范围	-20 度~85 度
		最大输出率	400HZ
		横滚角/俯仰角-静态误差	0.4 度
		横滚角/俯仰角-动态误差	1.0 度
		加速度敏感性	0.1 度/s/g
执行器	M3508	数量	6 个
		用途	用途： 底盘轮组 4 个，用于驱动车辆前进 摩擦轮 2 个，用于加速弹丸

	GM6020	数量	6 个
		用途	1 个用于云台 Pitch 轴控制 1 个用于云台 Yaw 轴控制 4 个用于作为舵轮的转向电机

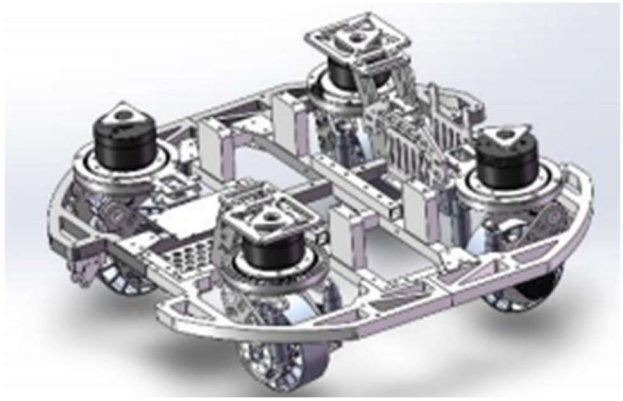
1.5 设计方案

1.5.1 机械结构设计

一、整体设计及核心结构说明

1.底盘部分

（1）底盘悬架选用四驱麦弗逊独立悬架配合舵轮，以实现车辆的全向运动和较好的盲道适应性能以及稳定飞坡的功能。悬架硬点调教由 matlab 进行仿真得出。



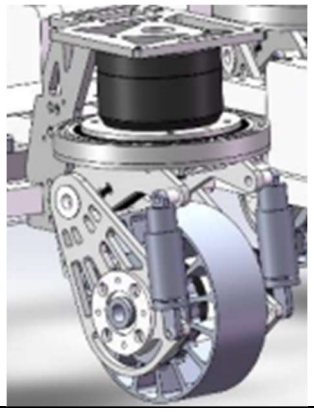
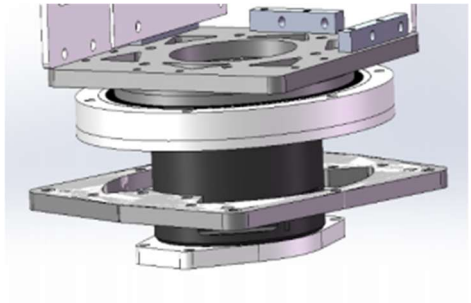


图 1-5-1-1 底盘和轮组展示

（2）Yaw 轴轴系通过 6020 电机满足轴向动力需求。同时利用交叉滚子轴承分别安装在轴系下端保证轴的径向负载能力。



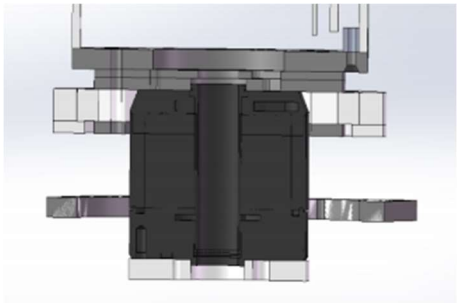
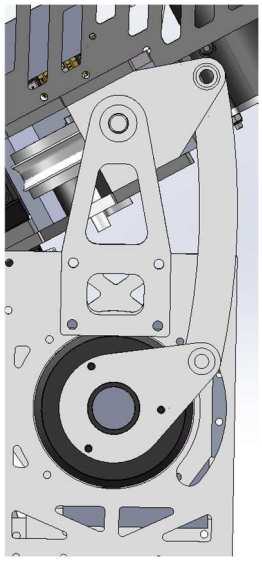


图 1-5-1-2

（3）Pitch 轴通过 6020 电机进行俯仰驱动。通过 2 连杆机构，能够将电机置放于脖颈位置处，以降低整车中心。轴系通过法兰轴承与赛达螺栓的保证了轴系的径向负载能力。



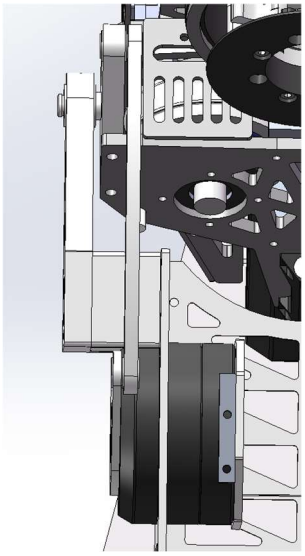


图 1-5-1-3

2.云台部分

① 基于摩擦轮的发射机构在枪管加装 U 型轴承以稳定弹丸进入枪管前的位置，以减少摩擦轮加速弹丸时弹丸加速方向的变化，以稳定弹丸发射初速及弹道。

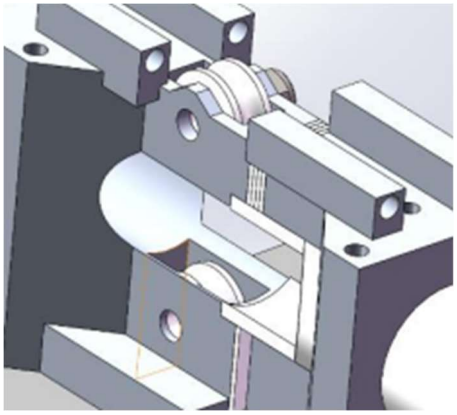


图 1-5-1-4-1

② 采用红点激光瞄准器对弹道方向进行辅助瞄准。同时采用安装座垫入玻纤垫片的方式调节激光瞄准器与枪口发射方向平行，做到所见物体即是所瞄准的目标。

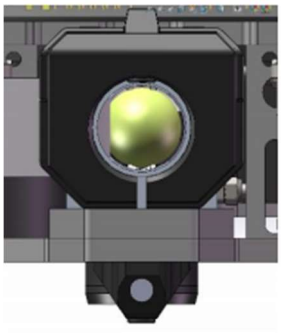


图 1-5-1-4-2

二、工艺选择

摩擦轮材料选择

聚氨酯包胶是在摩擦轮结构当中比较常用的一种材料，在抗疲劳老化、耐磨、防腐蚀等方面性能优异，在正常压力温度下使用寿命可以达到 5~10 年，橡胶与金属粘接强度好。现代工艺的粘结层优异的粘接强度有效避免了橡胶层与金属裂开、脱裂等现象的发生，该材料综合性价比较高

表 1 摩擦轮材料性能参数表

参数	数值
材料	聚氨酯包胶
弹性模量/(N·mm^2)	2.11×10^5
泊松比	0.29
强度极限/MPa	980
屈服极限/MPa	785
硬度/HBS	207
质量密度/(kg·mm^{-3})	7.85×10^{-3}

三、传感器设计安装及电路板固定走线

1.传感器设计安装

陀螺仪安装在云台上，其正方向与枪管所指发向一致。激光测距仪与图传相对固定，安装在云台上，且保证与图传视线平行。

2.电路板固定及走线

主控安装在云台上，用内攻丝尼龙柱连接于底板。分电板和继电器安装在底盘上，用

扎带绑在底板上。超级电容模块及管理模块用玻纤板榫卯结构制成的保护壳保护，用 M3 螺栓安装于底板。

走线均由轧带固定，云台电机的电源线及主控引下的信号线通过导电滑环连通到底上。对于电源线，底盘轮组上电源管理 chassis 接口-超级电容管理模块-功率控制板-超级电容器-底盘分电板-各轮组电机；云台上 Mini PC-继电器输入、输出供电端，底电板-继电器控制端-继电器输出端-云台 Yaw 轴电机和云台 Pitch 轴电机及主控；发射机构上电源管理板 Ammo-Booster 接口-拨盘电机和云台摩擦轮电机。信号线方面，各轮组电机-底盘分电板；云台上 Pitch 轴电机-主控，Yaw 轴电机-底盘分电板；发射机构上拨盘电机-底盘分电板，摩擦轮电机-主控。

四、核心零件有限元分析及静动力学分析模型

1. 基于摩擦轮的 17mm 弹丸发射机构动力学模型

（1） 力学模型建立

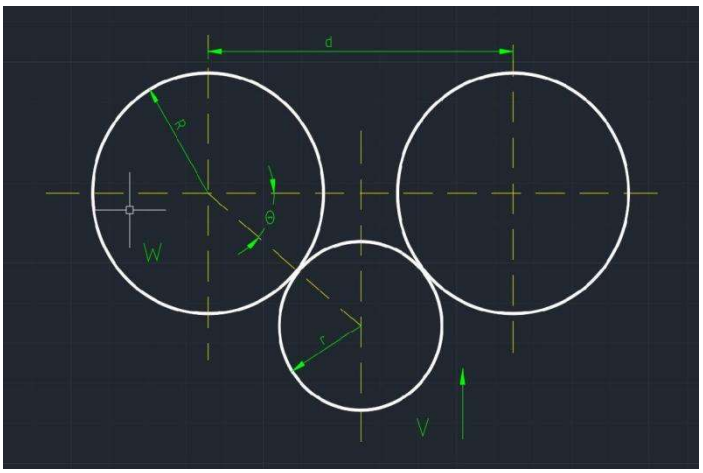


图 1-5-1-5

模型：假设一对摩擦轮转动仅受弹丸作用的摩擦力

已知：摩擦轮的半径为 R,间距为 d,角速度为 w,弹性模量为 E,表面摩擦系数为 u 取 0.7,胶皮厚度 D，弹丸的半径为 r,推动速度为 v,质量为 m

下面计算当弹丸经过加速之后的速度：

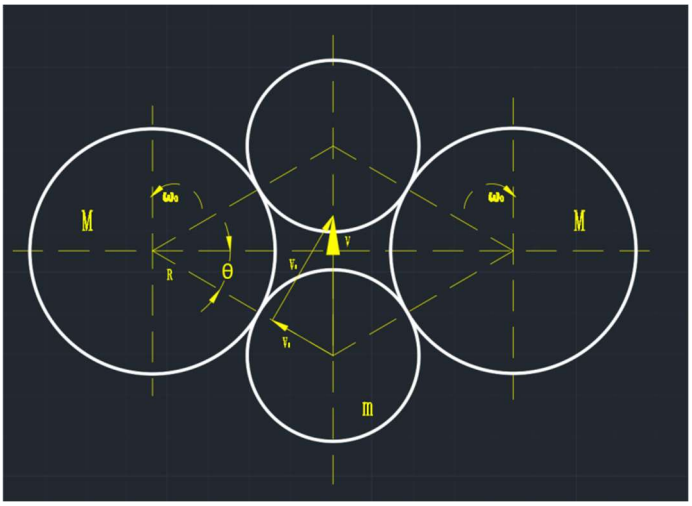


图 1-5-1-6

$$v_1 = v\cos\theta$$

$$v_2 = v\sin\theta$$

$$\cos\theta = \frac{d}{2(R+r)}$$

由于θ较小，这里我们处理为$v_1 \approx v, v_2 \approx 0$

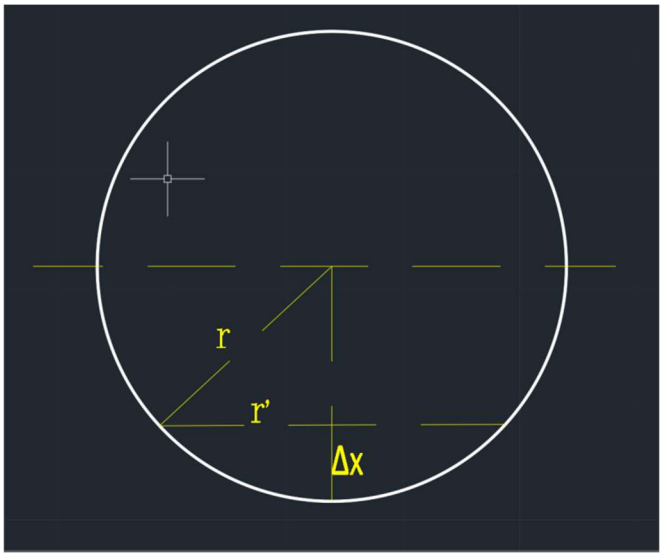


图 1-5-1-7

设摩擦轮由于压力产生的形变为Δx，则有：

$$\Delta x = R + r - \frac{d}{2\cos\theta}$$

$$r' = \sqrt{r^2 - (r - \Delta x)^2} = \sqrt{r^2 - \left(\frac{d}{2\cos\theta} - R\right)^2}$$

设 S 为以r'为半径的圆的面积：

$$S = \pi\left[r^2 - \left(\frac{d}{2\cos\theta} - R\right)^2\right]$$

所以弹丸和球的正压力F_N为：

$$F_N = \pi\left(R + r - \frac{d}{2\cos\theta}\right)\left[r^2 - \left(\frac{d}{2\cos\theta} - R\right)^2\right]\frac{E}{D}\eta$$

上式中的η为补偿系数，对此作如下解释：

我们将摩擦轮曲面处理为平面，弹丸处理为刚性球面，

表达式：$\pi\left(R + r - \frac{d}{2\cos\theta}\right)\left[r^2 - \left(\frac{d}{2\cos\theta} - R\right)^2\right]$

计算的是在如下图情况下产生的正压力。

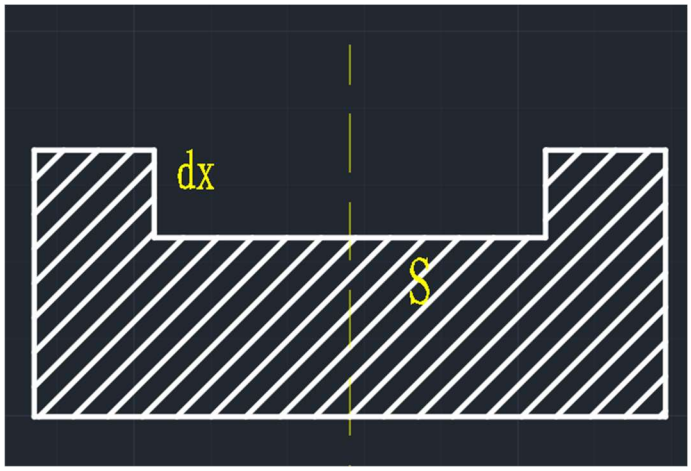


图 1-5-1-8

实际情况是 S 面为弹丸的球面，最大压缩距离为Δx，不难看出实际情况的F_N小于上图情况的F_N，η因此要用系数　　进行补偿

计算η值近似为 $\frac{1}{2}$ ，这里不再详细计算。

下面讨论弹丸加速度情况：

弹丸加速度如上图可分为两种情况：

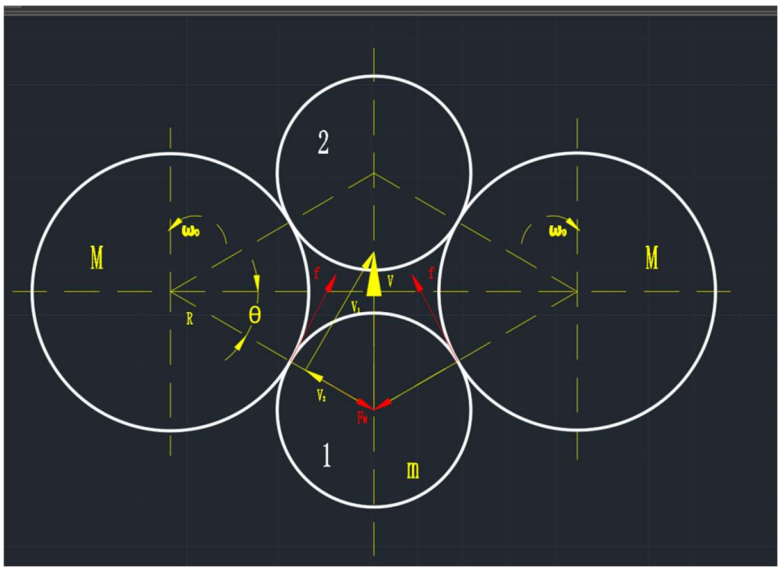


图 1-5-1-9

i.　当弹丸在 1 位置是合力大小为摩擦力f沿发射方向的分力减去正压力F_N沿发射方向的分力

此时加速度大小为：

$$a_1 = \frac{2\pi\eta E}{\mathrm{Dm}}\left(R + r - \frac{d}{2\cos\theta}\right)\left[r^2 - \left(\frac{d}{2\cos\theta} - R\right)^2\right](\mathrm{u}\cos\theta - \sin\theta)$$

ii. 当弹丸在 1 位置是合力大小为摩擦力f沿发射方向的分力加上正压力F_N沿发射方向的分力

此时加速度大小为：

$$a_2=\frac{2\pi\eta E}{\mathrm{Dm}}\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right](\mathrm{ucos}\theta+\sin\theta)$$

不难看出，情况 i 和情况 ii 的临界位置是**当弹丸球心和摩擦轮圆心共面时**。

（2） 运动分析

描述：

i. 下面我们讨论一种基本运动过程：

试着想象用手推动弹丸进入一对完全静止的摩擦轮时，此时摩擦轮和弹丸之间的作用属于纯滚动摩擦，无相对滑动，这里为了表达方便，我们把这种情况称为摩擦轮和弹丸达到耦合状态。

当摩擦轮在电机带动下飞速转动时，弹丸与摩擦轮作用时，弹丸被加速，摩擦轮受到摩擦力产生的力矩作用减速，最终可能会达到耦合状态。

由此我们可以看出弹丸与摩擦轮系统有 3 种运动状态：

①弹丸在图 4 的位置 1 处时就已经和摩擦轮达到耦合状态。

②弹丸在图 4 的位置 2 处时就已经和摩擦轮达到耦合状态。

③摩擦轮转速极快，摩擦轮虽然在摩擦轮作用下减速，但始终不会和弹丸达到耦合状态，也就是说弹丸全程和摩擦轮处于相对滑动状态。

ii. 下面我们来对每一种情况具体分析

首先我们假设单个摩擦轮转动件的转动惯量为 J

① 合力矩 M,角加速度为α：

$$M=\mathrm{fR}=\pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E}{D}\eta\mathrm{Ru}$$

$$\alpha=\frac{M}{J}=\mathrm{fR}=\pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E\eta\mathrm{Ru}}{\mathrm{JD}}$$

最后我们可以得到如下关系式：

$$\left\{(\mathrm{wR})^2+R^2\int_{\theta_0}^{\theta}2\pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E\eta\mathrm{Ru}}{\mathrm{JD}}d\theta\right\}\cos\theta^2=v^2+\int_{\theta_0}^{\theta}\frac{2\pi\eta E}{\mathrm{Dm}}$$

$$\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right](\mathrm{ucos}\theta-\sin\theta)\frac{d}{2\cos\theta}d\theta=v_c$$

由上式我们可以求得在达到耦合时对应的θ角。

下面从能量转换角度计算弹丸离开摩擦轮时的速度：

设当达到耦合后摩擦轮与弹丸储存的弹性势能为E_1，角速度为W_c，

最大弹性势能为E_2：

$$E_1 = 2Er^4\pi\left(\frac{1}{2}k^2+\frac{1}{4}k^4-\frac{2}{3}k^2-\frac{1}{12}\right)$$

$$E_2 = 2Er^4\pi\left(\frac{1}{2}n^2+\frac{1}{4}n^4-\frac{2}{3}n^2-\frac{1}{12}\right)$$

$$w_c = \frac{v_c}{\mathrm{Rcos}\theta}$$

其中：

$$k = \frac{\frac{d}{2\cos\theta}-R}{r}, \quad n = \frac{\frac{d}{2}-R}{r}$$

所以：

$$\frac{1}{2}(w_c)^2J-\frac{1}{2}(w_{\min})^2J+\frac{1}{2}m\left(v_c{}^2-(w_{\min}R)^2\right)=E_2-E_1$$

$$\frac{1}{2}\left(\frac{v_0}{\cos\theta_0 R}\right)^2J-\frac{1}{2}\left(\frac{v}{\cos\theta R}\right)^2J+\frac{1}{2}m\left(v_0{}^2-v^2\right)=E_1$$

其中 v_0 即为弹丸发射最终速度。

② 合力矩 M,角加速度为：

$$M = \mathrm{fR} = \pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E}{D}\eta\mathrm{Ru}$$

$$\alpha = \frac{M}{J} = \mathrm{fR} = \pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E\eta\mathrm{Ru}}{\mathrm{JD}}$$

最后我们可以得到如下关系式：

$$\left\{(\mathrm{wR})^2+R^2\int_{\theta_0}^{0}2\pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E\eta\mathrm{Ru}}{\mathrm{JD}}d\theta-R^2\int_{0}^{\theta}2\pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E\eta\mathrm{Ru}}{\mathrm{JD}}d\theta\right\}\cos\theta^2 = v^2-\int_{\theta_0}^{0}\frac{2\pi\eta E}{\mathrm{Dm}}$$

$$\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right](\mathrm{ucos}\theta-\sin\theta)\frac{d}{2\cos\theta}d\theta+\int_{0}^{\theta}\frac{2\pi\eta E}{\mathrm{Dm}}\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right](\mathrm{ucos}\theta+\sin\theta)\frac{d}{2\cos\theta}d\theta = v_c$$

③ 合力矩 M,角加速度为α：

$$M = \mathrm{fR} = \pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E}{D}\eta\mathrm{Ru}$$

$$\alpha = \frac{M}{J} = \mathrm{fR} = \pi\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right]\frac{E\eta\mathrm{Ru}}{\mathrm{JD}}$$

$$v^2-\int_{\theta_0}^{0}\frac{2\pi\eta E}{Dm}\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right](u\cos\theta-\sin\theta)\frac{d}{2\cos\theta}d\theta+\int_{0}^{\theta_0}\frac{2\pi\eta E}{Dm}\left(R+r-\frac{d}{2\cos\theta}\right)\left[r^2-\left(\frac{d}{2\cos\theta}-R\right)^2\right](u\cos\theta+\sin\theta)\frac{d}{2\cos\theta}d\theta=v_c$$

（3） 加速机理解释

① 摩擦轮通过两种模式加速弹丸：

摩擦轮转速相对较低时，前半段弹丸与摩擦轮处于相对滑动状态，弹丸受滑动摩擦力，后半段处于耦合状态，摩擦轮与弹丸速度由映射关系。摩擦轮转速相对较高状态，弹丸全程受滑动摩擦力。

② 在弹丸离开摩擦轮时会面临摩擦轮弹性势能释放，导致弹丸速度再一次加速。

（4） 技术路线选择

基于该动力学模型，及实际实验得出弹丸落点呈现随机分布的特点，我们认为导致弹丸初速不同的主要原因是由于弹丸表面结构不同，摩擦系数呈现随机性使得初速在一定范围内波动，因此应当选择摩擦轮转速低速并且减少摩擦轮掉速的方案，利用在该情况下弹丸收尾速度与摩擦轮转速有映射关系，因此只要保证摩擦轮转速稳定就可以使得弹丸初速稳定。在实验中我们发现，增大摩擦轮转动惯量使得其掉速低于 200rpm 时，弹丸初速波动小于 0.2m/s。

2. 云台 pitch 轴拉簧力平衡模型的确立

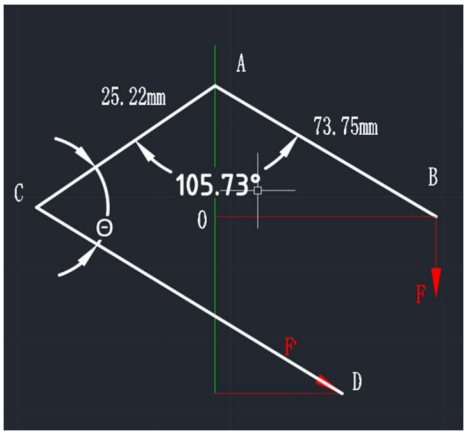


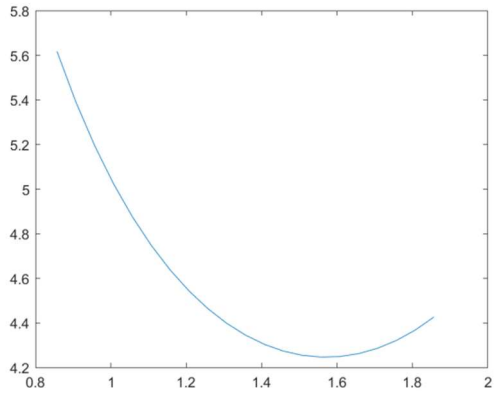
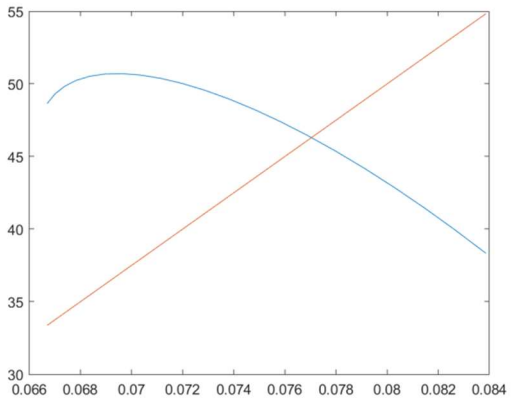
图 1-5-1-10 力平衡示意图

```
%以下代码用于利用 matlab 计算拉簧力学模型
%如图，以在 D 点左边且在 O 点正下方的点为原点；C，D 点为拉簧的两安装点；B 点为配重的质心所在点
clear,clc,close all;
syms M1 M2 a F F1 b F1all CDall i M21 M11 Cos Sin CD CD1 d h LineX LineY dForce dForce1 dForceAll k1;
b=0.857:0.05:1.905;
%以下距离如图标注，单位为 SI。
d=0.016;
h=0.09;
k1=1250;%弹簧劲度系数
M1=0.0735*sin(a)*F%配重在 A 点产生的力矩
```

M1 = $\frac{147F\sin(a)}{2000}$

```
Cos=(0.02522*sin(1.852-a)+d)*0.02522*sin(1.852-a)+0.02522*cos(1.852-a)*(0.02522*cos(1.852-a)-h);%角 ACD 的余弦值
Sin=sin(acos(Cos));
```

```
M2=0.02522*F1*Sin
CD=sqrt((d+0.022522*sin(1.852-a))^2+(0.02522*cos(1.852-a)-h)^2);%CD 长度，即弹簧工作时的长度
for i=b
    %这里是做了一个 M1，M2 的变量复制，复制给 M21，M11，让后者参与循环；目的是为了避免在循环时污染表达式 M1，M2.
    M21=M2;
    M11=M1;
    CD1=CD;
    M21=subs(M21,a,i);%把式子里的 a 全部换成 i
    M11=subs(M11,a,i);
    M11=subs(M11,F,vpa(2*9.8*pi*0.03^2*0.04*7850));
    if 0.022522*cos(1.852-i)/0.022522*sin(1.852-i)-(0.02522*cos(1.852-i)-h)/(0.02522*sin(1.852-i)+d)>0
        F1all(end+1)=vpa(solve(M21==M11,F1));%储存一组拉簧本应该提供的拉力值（刚好使云台力平衡）
        CDall(end+1)=vpa(subs(CD1,a,i));%储存一组拉簧长度值
    end
end
F1all=F1all(2:end);
CDall=CDall(2:end);%把第一个垃圾数值去掉
plot(CDall,F1all);%弹簧本应该提供的拉力大小图像
hold on
%下面开始画图
LineX=CDall;
LineY=k1*(LineX-0.04);%拉簧拉力
plot(LineX,LineY);%弹簧的拉力-长度图像
%这是画出电机所需要提供的扭矩 在摩擦轮安装位置所产生的垂直拉力大小
dForce=max(F1all)-min(F1all)
for i=b
    M21=M2;
    M11=M1;
    M21=subs(M21,a,i);
    M21=vpa(subs(M21,F1,dForce));
    M11=subs(M11,a,i);
    M11=subs(M11,F,dForce1);
    dForceAll(end+1)=vpa(solve(M21==M11,dForce1));
end
dForceAll=dForceAll(2:end)
hold off
plot(b,dForceAll)
```

	
实际所需驱动力（蓝线）和弹簧（黄线）提供驱动力关于 pitch 轴角度曲线	6020 电机需要提供的驱动力在配重部分产生的扭矩关于 pitch 轴角度曲线

（三）加速机理解释：

1.摩擦轮加速弹丸由两种模式，一个是摩擦轮转速相对较低，前半段弹丸与摩擦轮处于相对滑动状态，弹丸受滑动摩擦力，后半段处于耦合状态，摩擦轮与弹丸速度由映射关系。另一个是摩擦轮转速处于相对较高状态，弹丸全程受滑动摩擦力。

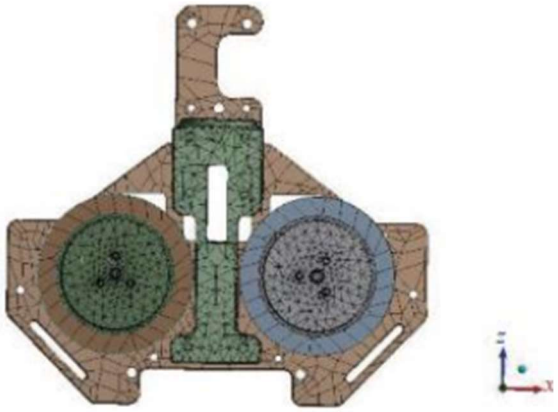
2.在弹丸离开摩擦轮时会面临摩擦轮弹性势能释放，导致弹丸速度再一次加速。

（四）技术路线选择

基于该动力学模型，及实际实验得出弹丸落点呈现随机分布的特点，我们认为导致弹丸初速不同的主要原因是由于弹丸表面结构不同，摩擦系数呈现随机性使得初速在一定范围内波动，因此应当选择摩擦轮转速低速并且减少摩擦轮掉速的方案，利用在该情况下弹丸收尾速度与摩擦轮转速有映射关系，因此只要保证摩擦轮转速稳定就可以使得弹丸初速稳定。在实验中我们发现，增大摩擦轮转动惯量使得其掉速低于 200rpm 时，弹丸初速波动小于 0.2m/s。

2. 摩擦系数对结构受力影响

在 UG 三维软件建立摩擦轮发射装置后，导入 ANSYS 有限元仿真软件时对模型进行合理简化。参考相关研究后略去各零件间的螺栓连接部分，改为刚性连接。为了得到更精确的结果，同时对摩 擦轮部分采取了网格加密处理，利用 ANSYS 软件， 对电机和摩擦轮部分进行网格优化，使得整体网格 节点达到 85575 个，网格单元达到 40974 个。网格划分结果如图所示

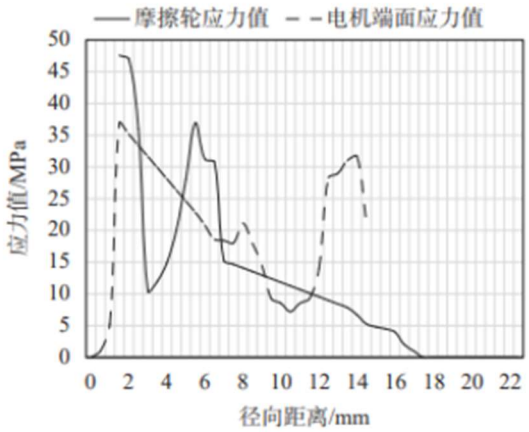


摩擦轮部分网格划分结果

瞬态结构动力学分析（又称时间历程分析）是用来确定固定结构随时间变化的动力学响应的常用方法。通过瞬态动力学 Newmark 隐式时间积分法 分析，可以确定结构在简谐载荷、瞬态载荷、和稳态 载荷随意组合作用下随时间变化的位移、应变、应 力及力。其相关量之间的关系下式所示： $MX'' + CX' + KX = F(t)$

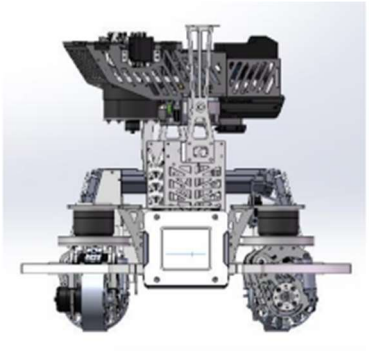
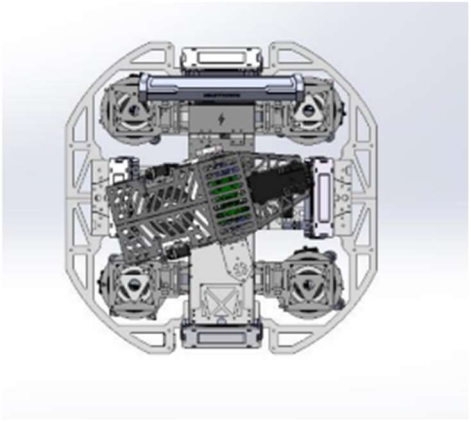
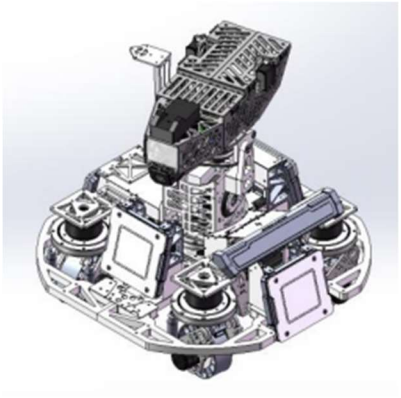
式中：M——质量矩阵；C——阻尼矩阵； K—一刚度矩阵；X'' ——加速度向量；X' ——速度向量； X——位移向量； $F(t)$——变载荷向量。

对摩擦轮发射机构模型在边界条件设置为：摩擦轮施加 350 rad/s 旋转速度，完成摩擦系数分别为 0. 16，0. 18，0. 20，0. 22 的仿真测试，得到不同摩擦系数下摩擦轮所受应力的变换趋势如图所示。



摩擦系数对最大应力的影响

正视图	侧视图

	
俯视图	斜视图
	

1.5.2 硬件设计

一、整机硬件方案框图

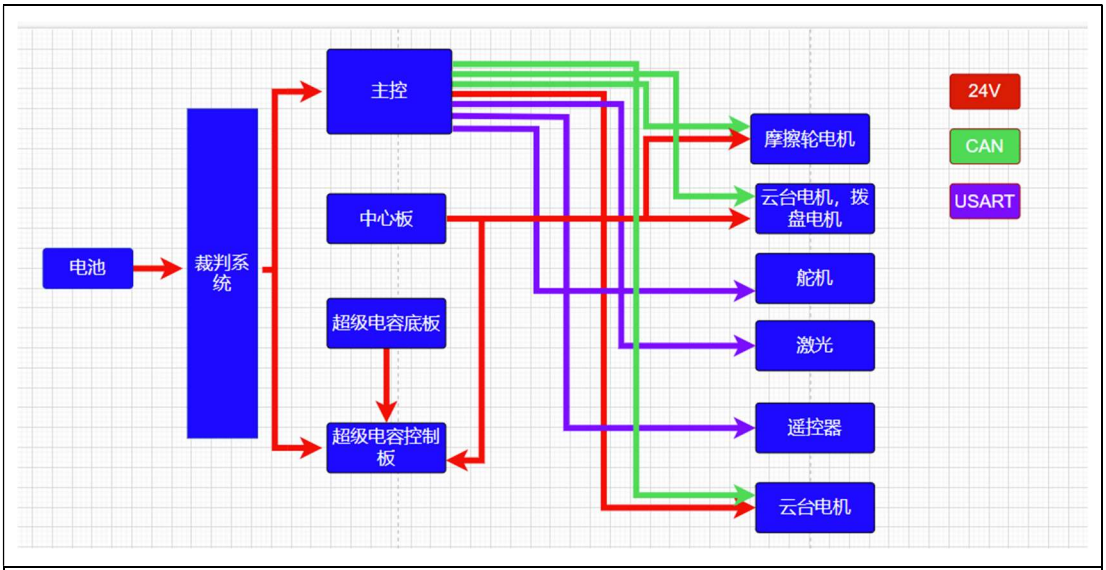


图 1-5-2-1 硬件系统框图

二、硬件详细设计（自研）

1.主控模块

主控板选用 STM32F405RGT6 芯片，硬件电路设计如下

① 复位模块

在运行过程中，如果单片机进入了休眠状态或者处于程序不可控的状态，要使系统运行，完成初始化，需要应用重新上电的方式，但是这种方式对系统会造成不利的影响。为了解决这一问题，更好地保护系统，设计复位电路，通过复位电路来控制系统的初始化，在系统中设置复位按键，能防止重新上电造成的影响。复位功能的具体实现方式是，将单片机引脚和电容接成回路，并设置按键，通过按键来控制充放电，通过这样的设计就可以使系统进行初始化。复位电路选用 MAX823 芯片。

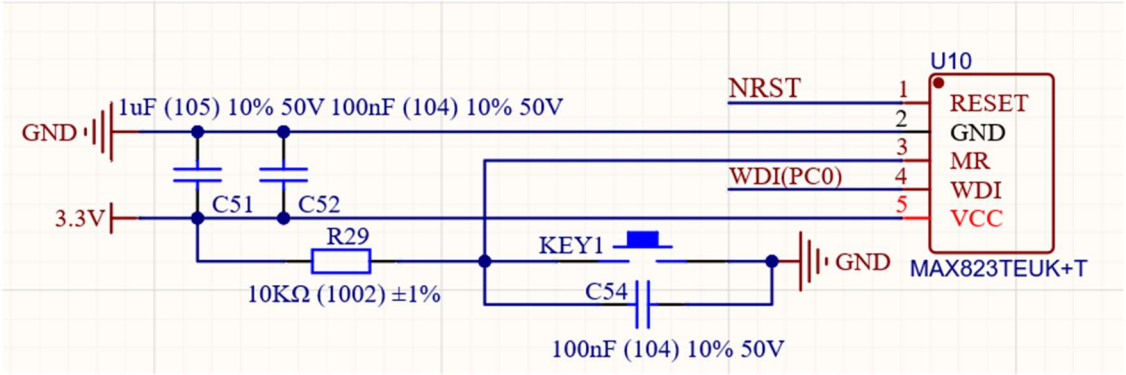


图 1-5-2-2 复位电路图

② 供电模块

在供电模块方面，STM32 单片机是 32 位低功耗的高速 MCU，同时具有非常高的性价比，通过数据线将其和计算机进行连接，就能对其进行充电，从而保证其正常的工作和运行。但是，STM32 单片机采用的处理器内核具有较宽的供电范围，在对其进行充电时可以采用适中的电压。可以使用芯片进行降压，通过这样的供电模块设计就能有效地满足 STM32 单片机的供电需求。此外，通过这样的供电电路设计，并且在电源的输入端和输出端加装滤波电容，能降低电源本身的波动性以及提高系统本身的稳定性。供电模块选用的芯片包括 ME3116，ME3148A,LM1117。通过 ME3116,ME3148A 以及 LM1117 降压芯片将 24V 转为 5V 和 5V 转为 3.3V，ME3116 用于内部降压，ME3148A 用于外部降压。

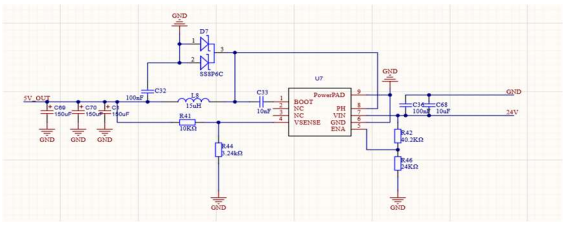


图 1-5-2-3 ME3148A 芯片 24V 转 5V 原理图

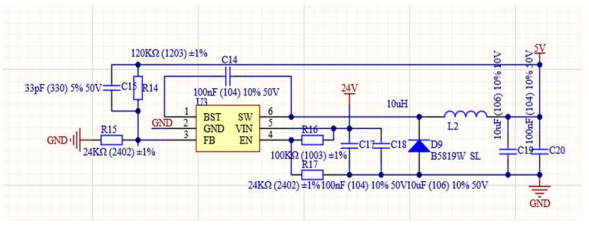


图 1-5-2-4 ME3116 芯片 24V 转 5V 原理图

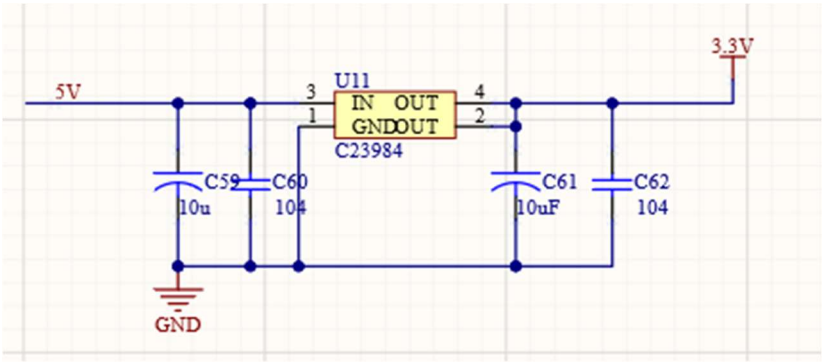


图 1-5-2-5 LM1117 芯片 5V 转 3.3V 原理图

③ CAN 通讯模块

CAN 控制器根据两根线上的电位差来判断总线电平。总线电平分为显性电平和隐性电平，二者必居其一。发送方通过使总线电平发生变化，将消息发送给接收方。CAN 通讯电路选用 MCP2562-E/SN 芯片。

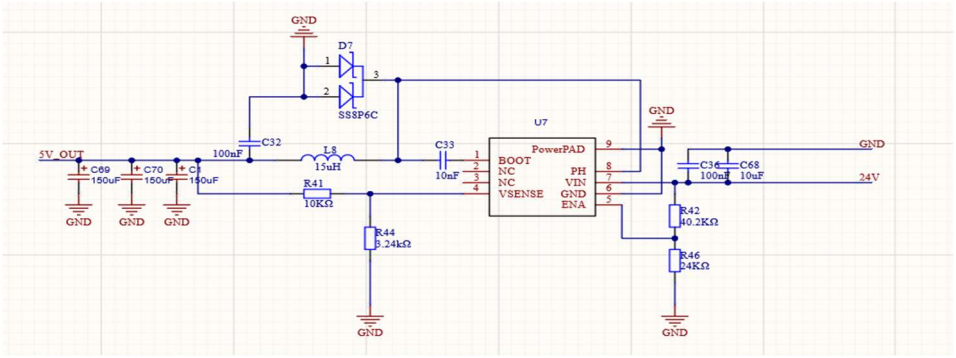
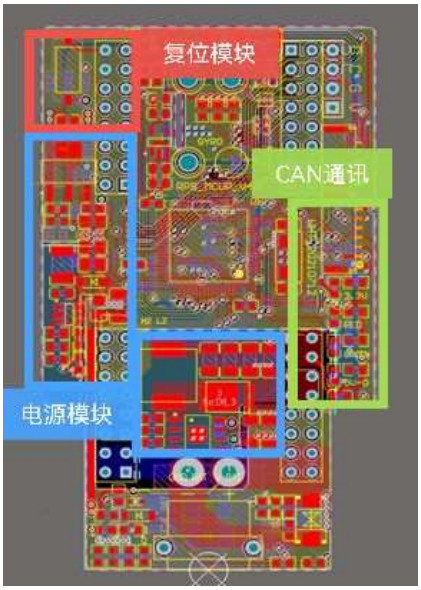
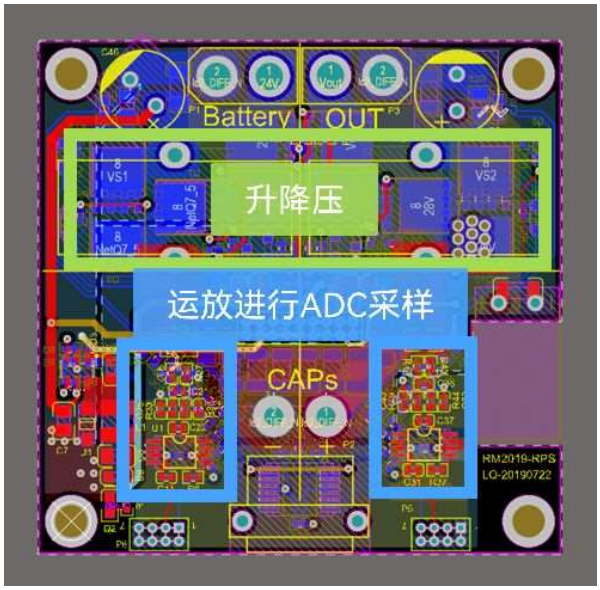


图 1-5-2-6 MCP2562-E/SN 芯片 CAN 通讯原理图



主控板



超级电容控制板

2.超级电容模块

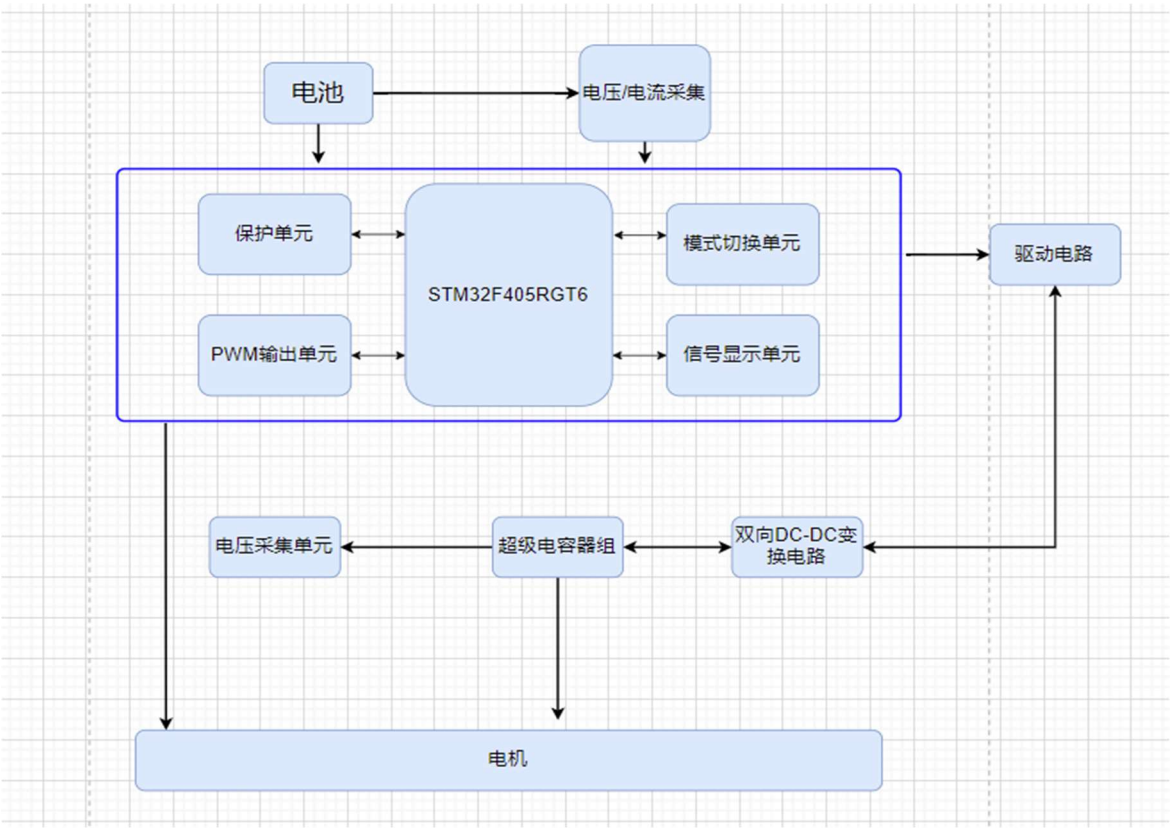


图 1-5-2-7 超级电容整体框图

① 硬件部分

电路采用 STM32F103RE 芯片，通过 I2C 总线连接数据采集芯片。STM32F103RE 芯片使用 GPIO 端口连接 DXL345 和 ITG3205。由于 STM32F103RE 的片内外设 I2C 模块直接使用存在问

题，因此程序中使用 GPIO 模拟了 I2C 总线的数据和时钟，保证了控制芯片和采集芯片的良好通信。

通过多个 MOS 管和三极管进行升降压，将电池升压到 28V 对电容组进行充电，从电池降压为 24V 输出，Is_DIFF_N，Is_DIFF_P，Is2_DIFF_N，Is2_DIFF_P，Ic_DIFF_P，Ic_DIFF_N，Ic2_DIFF_P，Ic2_DIFF_N 通过电流放大器进行放大之后通过 ADC 采样检测，最后通过栅极驱动器对 HO1,VS1,LO1 进行驱动,使电路正常工作。

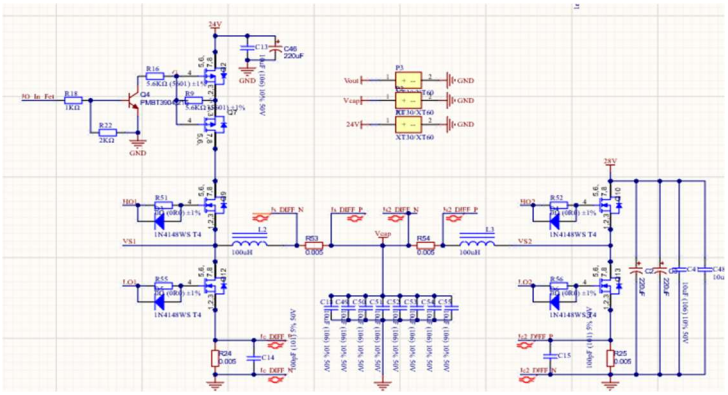


图 1-5-2-8 升降压原理图

② 软件部分

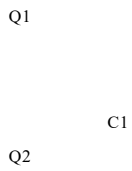


图 1-5-2-9 电路原理图

根据电感电流和电容电压列写微分方程：

$$\begin{cases} U_i - U_2 = L\dfrac{di_L}{dt} + R_L i_L \\ i_c = C\dfrac{dU_o}{dt} \end{cases}$$

由功率守恒可得：

$$U_i i_L = U_o(i_c + i_{load})$$

将以上式子转换到 s 域：

$$\begin{cases} i_L = \dfrac{U_i - U_2}{Ls + R_L} \\ U_o = \dfrac{i_c}{Cs} \\ i_c + i_{load} = \dfrac{U_o}{U_i} i_L \end{cases}$$

可以构建 PI 双环控制系统：

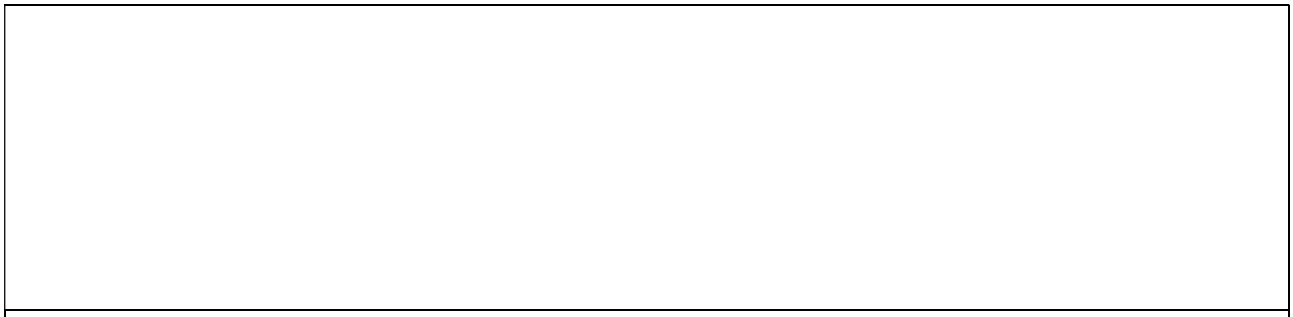


图 1-5-2-10 PI 双环系统结构图

电压环为外环，控制输出电压U_o；电流环为内环，控制电感电流i_L。其中，$\frac{1}{1+1.5T_s s}$为系统总的控制延时环节，包含采样、计算、PWM 装载和 PWM 的等效一阶惯性环节。（T_s为控制周期）。

```
324
325         //电压环
326         PI_CapVoltage.OutMax =  SysPara.ChargePower/Vc;   //电压环输出限幅值，对应于充电电流的幅值
327         if(PI_CapVoltage.OutMax > SysPara.MaxChargeCurrent) //限制电流给定
328           PI_CapVoltage.OutMax = SysPara.MaxChargeCurrent;
329         PI_CapVoltage.OutMin = -PI_CapVoltage.OutMax;
330         PI_CapVoltage.UiMax  =  PI_CapVoltage.OutMax;
331         PI_CapVoltage.UiMin  =  PI_CapVoltage.OutMin;
332
333         PI_CapVoltage.Ref = Vin-1.0f;//22.0f;       //电容充电电压为(电池电压-1)V
334         if(PI_CapVoltage.Ref > SysPara.MaxCapVoltage)
335           PI_CapVoltage.Ref = SysPara.MaxCapVoltage;
336         PI_CapVoltage.Fdb = Vc;
337         PI_CapVoltage.calc(&PI_CapVoltage);
338
339         //电流环
340         PI_ChargeCurrent.Ref = PI_CapVoltage.Out;
341   //      PI_ChargeCurrent.Ref = 2.0f;
342         PI_ChargeCurrent.Fdb = Icharge;
343         PI_ChargeCurrent.calc(&PI_ChargeCurrent);
344         Duty = (PI_ChargeCurrent.Out + Vc)/Vin;
345       }
346     }
347     else
348     {
349       EN_BUCK_DRV = 0;
350       IN_FET = 0;
351       Timer_BUCK_Recovery = 0;
352       Reset_Charge_Patrameters();
353       Duty = Vc/Vin;
354     }
355     #endif
```

超级电容软件控制

Ⅰ．电流控制器参数计算

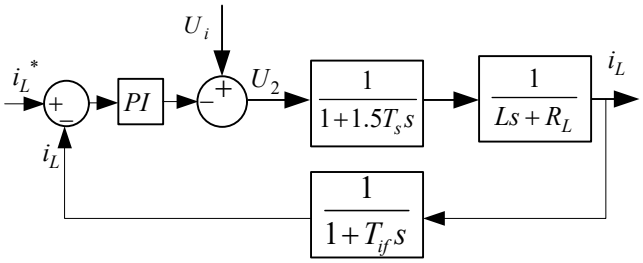
1） 已知参数：

i.电感值和电阻值$L=1.2\text{mH}$　$R=80\text{m}\Omega$

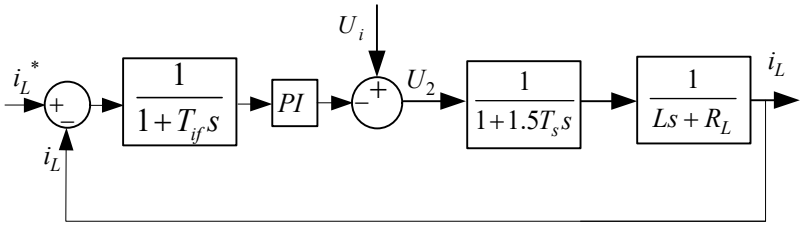
ii.电流环的反馈滤波时间常数：RC 滤波，时间常数为$T_{if}=RC=100*10e-9=1\mu s$

iii.控制频率取 20kHz，即 $T_s=50\mu s$

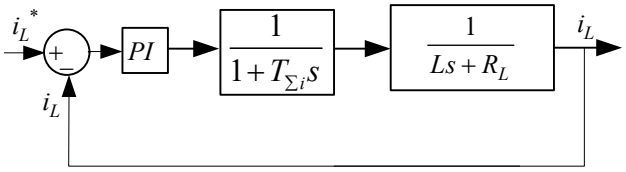
2） 电流环结构框图推导：



如果电流给定进行同样的滤波，上面的框图可画成单位负反馈的形式：



合并小时间常数，忽略前馈项，可得：



其中，电流环小时间常数为：

$$T_{\Sigma i} = T_{if} + 1.5T_s = 1 + 50 * 1.5 = 76\mu s$$

显然，电流环的控制对象为双惯性型的，根据《电力拖动自动控制系统》(陈伯时)可知，电流环应设计成 I 型系统，故控制器选择 PI 控制器，其传递函数为：

$$\frac{K_{Pi}(T_{Ii}s+1)}{T_{Ii}s}$$

电流控制系统开环传递函数为：

$$\frac{K_{Pi}(T_{Ii}s+1)}{T_{Ii}s}\frac{1}{1+T_{\Sigma i}s}\frac{1/R_L}{(L/R_L)s+1}$$

为构成 I 型系统，使控制器零点与控制对象大时间常数极点相抵消，即：

$$T_{Ii} = L/R_L$$

从而开环传递函数变为典型 I 型系统：

$$\frac{K_{Pi}}{T_{Ii}s}\frac{1/R_L}{1+T_{\Sigma i}s} = \frac{K_{Pi}/L}{s(1+T_{\Sigma i}s)}$$

一般情况下，希望电流超调量不超过 5%，参考下表，可以选择 $\xi=0.707$，此时 *KT*=0.5，对应到电流环控制系统，可得：

$$T_{\Sigma i}K_{Pi}/L=0.5$$

从而，PI 比例项参数可得：

$$K_{Pi}=0.5L/T_{\Sigma i}=\frac{0.5\times1.2\times10^{-3}}{76\times10^{-6}}=7.8947$$

PI 控制器的积分参数为：

$$K_{Ii}=\frac{K_{Pi}}{T_{Ii}}=\frac{K_{Pi}}{L/R_L}=\frac{7.8947}{1.2/80}=526.3158$$

为在单片机中实现 PI 算法，需要对 PI 进行离散化，离散化后的比例系数不变，而积分系数应再乘以采样时间 T_s 所以有：

$$K_{Pi\_z}=7.8947$$

$$K_{Ii\_z}=K_{Ii}T_s=526.3158\times50e-6=0.0263$$

作为典型的 I 型系统，其开环传递函数选择为$W(s)=\frac{K}{s(Ts+1)}$

表 2－2　典型 I 型系统动态跟随性能指标和频域指标与参数的关系

参数关系 KT	0.25	0.39	0.50	0.69	1.0
阻尼比 ξ	1.0	0.8	0.707	0.6	0.5
超调量 σ	0%	1.5%	4.3%	9.5%	16.3%
上升时间 t_r	∞	$6.6T$	$4.7T$	$3.3T$	$2.4T$
峰值时间 t_p	∞	$8.3T$	$6.2T$	$4.7T$	$3.6T$
相角稳定裕度 γ	76.3°	69.9°	65.5°	59.2°	51.8°
截止频率 ω_c	0.243 $/T$	0.367 $/T$	0.455 $/T$	0.596 $/T$	0.786 $/T$

具体选择参数时，如果工艺上主要要求动态响应快，可取 $\xi=0.5\sim0.6$，把 K 选大一些；如果主要要求超调小，可取 $\xi=0.8\sim1.0$，把 K 选小一些；如果要求无超调，则取 $\xi=1.0$，$K=0.25/T$；无特殊要求时，可取折中值，即 $\xi=0.707$，$K=0.5/T$，此时略有超调（$\sigma=4.3\%$）。也可能出现这种情况：无论怎样选择 K 值，总是顾此失彼，不可能满足所需的全部性能指标，这说明典型Ⅰ型系统不能适用，须采用其他控制方法。

上述折中的 $\xi=0.707$，$KT=0.5$ 的参数关系就是西门子“最佳整定”方法的“模最佳系统”，或称“二阶最佳系统”。其实这只是折中，不能算“最佳”，根据不同的工艺要求可有不同的最佳参数选择。

Ⅱ．电压控制器参数计算

1）时间常数计算

电流环在电压环中可以等效为一阶惯性环节，其时间常数为：

$$2T_{\Sigma i}=2\times76=152\mu s$$

电压采样及处理电路的滤波时间常数：

$$T_{vf}=RC=1000*100e-9=100\mu s$$

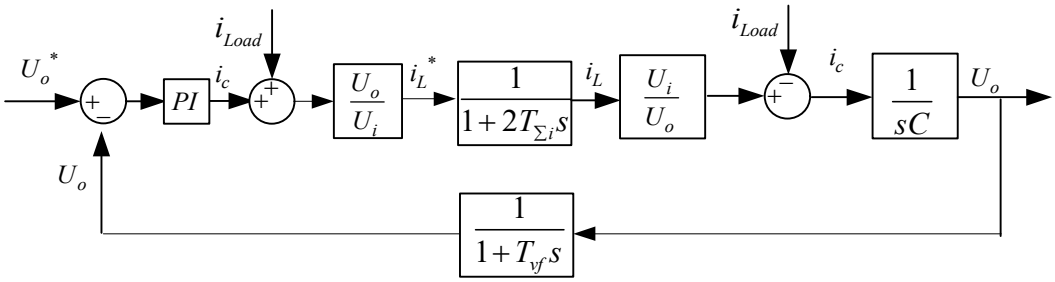
电压环小时间常数总和为：

$$T_{\Sigma V}=2T_{\Sigma i}+T_{Vf}=252\mu s$$

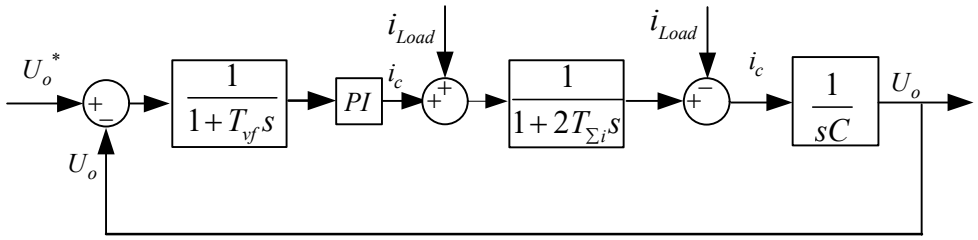
2）控制器参数的设计

经 PI 闭环控制后，电流环可以近似等效一阶惯性环节，且时间常数较小，这就表明，电流环的控制改造了控制对象，使得电流相应加快。这也就是双环系统中，电流环作为内环的一个重要作用。

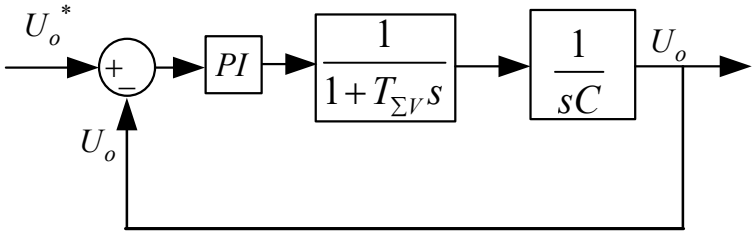
电压环等效控制框图如下所示：



等效成单位负反馈系统：



合并小惯性环节，并忽略前馈项，控制系统框图进一步简化为：



一般情况下，为实现电压外环误差控制，控制器需要包含积分环节，由上图可以看出，系统中包含一个积分环节（惯性环节），因此电压环开环传递函数应有两个积分环节，所以应设计成典型 II 型系统，同时，这样的系统动态抗扰性能较好。

电压环 PI 调节器的传递函数为：

$$\frac{K_{PV}(T_{IV}s+1)}{T_{IV}s}$$

于是系统的开环传递函数为：

$$\frac{K_{PV}(T_{IV}s+1)}{CT_{IV}s^2(T_{\Sigma V}s+1)}$$

令电压环开环增益：

$$K_V=\frac{K_{PV}}{CT_{IV}}$$

则系统开环传递函数为：

$$\frac{K_V(T_{IV}s+1)}{s^2(T_{\Sigma V}s+1)}$$

电压环 PI 控制器的参数包含 K_{PV} 和 T_{IV}。按照典型 II 型系统的参数关系，由式 2-38（电力拖动自动控制系统），则有：

$$T_{IV} = hT_{\Sigma V}$$

根据跟随和抗扰性能都比较好的原则，取 $h=5$，则有：

$$T_{IV} = 5\times 252\times 10^{-6} = 0.0013$$

由式 2-39（电力拖动自动控制系统），电压环开环增益：

$$K_V = \frac{h+1}{2h^2T_{\Sigma V}^2} = \frac{6}{2T_{IV}^2} = \frac{3}{2\times 1.3^2\times 10^{-6}} = 0.8876\times 10^6$$

于是 PI 调节器的比例常数为：

$$K_{PV} = K_V C T_{IV} = 0.8876\times 10^6\times 1000\times 10^{-6}\times 1.3\times 10^{-3} = 1.1539$$

校验近似条件：电压环截止频率为：

$$\omega_{CV} = \frac{K_V}{\omega_1} = K_V T_{IV} = 0.8876\times 10^6\times 1.3\times 10^{-3} = 1153.9$$

电流环传递函数简化的条件：

$$\omega_{CV} < \frac{1}{3}\sqrt{\frac{0.5/T_{\Sigma i}}{T_{\Sigma i}}} = \frac{1}{3}\frac{0.5/(76\times 10^{-6})}{76\times 10^{-6}} = 3101.3$$

满足简化条件。

所以电压环比例和积分常数分别为：

$$K_{PV} = 1.1539,$$

$$K_{IV} = \frac{K_{PV}}{T_{IV}} = \frac{1.1539}{1.3\times 10^{-3}} = 887.6$$

离散之后：

$$K_{PV} = 1.1539,$$

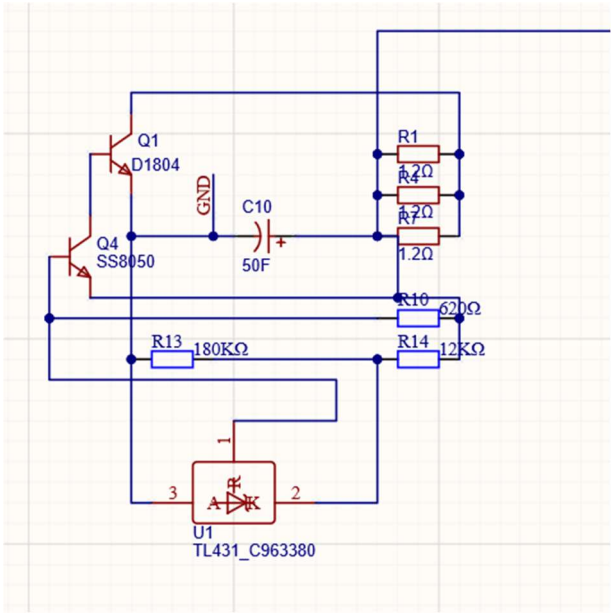
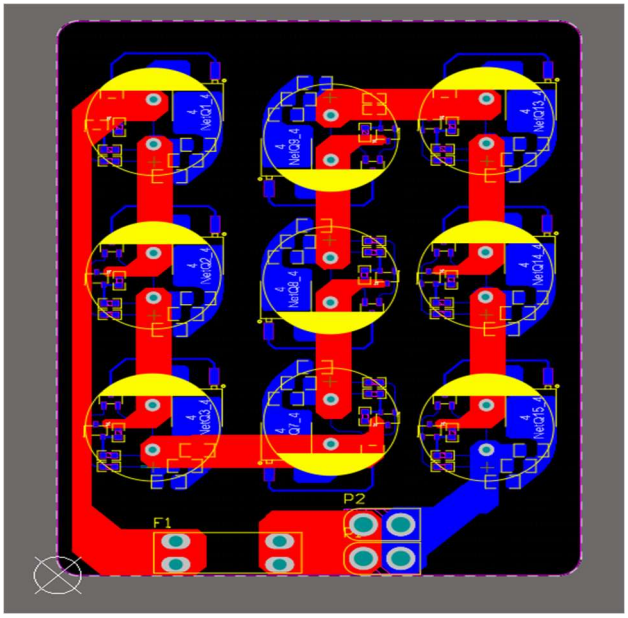
$$K_{IV\_z} = K_{IV} T_s = 887.6\times 5e-5 = 0.0444$$

3.超级电容底板

这个赛季选用九个 3V50F 电容。通过能量公式，计算出总能量为 2025J。

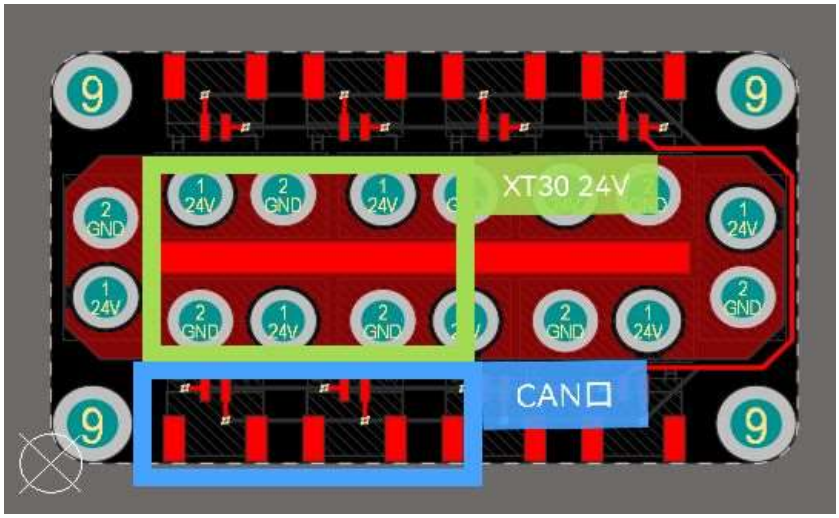
超级电容器组各包括九个 3V50F 超级电容器，每个超级电容器配置独立的电压控制，主要通过并联稳压电路完成。该稳压电路包括可调精密电压基准集成芯片、两个三级管、电压调整电阻网以及若干限流电阻组成。

其中 TL431 为可调精密电压基准集成芯片，内部含有一个 2.8V 的基准电压。当在调整端引入输出负反馈时，器件可以通过从阴极到阳极很宽范围的分流控制输出电压。超级电容器的最大输出电压为 3V，为安全起见，将该电压定位 2.8V。9 个超级电容器串联后，超级电容器组最大输出电压为 27 V 通过三个 1.2 欧的电阻并联，使得多余的能量通过电阻损失掉，通过 180K 和 12K 两个电阻进行分压，使得电容两端的电压为 2.8V。如果超过 2.8V，控制晶体管开启，让电容里的电流消耗，进而电压下降。通过 620 的电阻能减少一些能量的损失。

<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>超级电容底板原理图</td><td>超级电容底板 PCB 图</td></tr>
</table>

① 分电板

分电板由 8 个 XT30 和 8 个 CAN 口组成，XT30 用来提供 24V 电，8 个 CAN 口相连提供控制信号。



分电板 PCB 图

三、关键器件选型

1. 主控板

选用 STM32F405RGT6 芯片,24V 转 5V 内外分别选用微盟电子 ME3148 和 ME3116 芯片，持续输出 3A 的电流（峰值电流为 4A），转换效率可以达到 95%，输出电压初始精度为 1.5%。电路设计较为简单，芯片性价比较高。5V 转 3.3V 选用 LM1117 芯片，LM1117 芯片提供电流限制和过热保护。电路包含 1 个齐纳调节的带隙参考电压以确保输出电压的精度在±1%以内，使用较为普遍，带负载能力较强。陀螺仪选用 HI226,HI226 集成了六轴加速度计、六轴陀螺仪和一款微控制器。可输出经过传感器融合算法计算得到的基于当地地理坐标的三维方位数据，包含无绝对参考的相对航向角，俯仰角和滚转角。同时也可以输出校准过的原始传感器数据。CAN 模块选用 MCP2562-E/SN CAN 芯片，芯片为 CAN 协议控制器提供差分发送和接收能力，并与 ISO-11898-2 和 ISO-11898-5 标准完全兼容，以 1Mbps 的速度运行。

2. 超级电容控制板

选用 STM32F405RGT6 芯片，同时选用 MP2456 降压芯片将 24V 降压到 12V，芯片体积小，具有功耗小，效率高，输出电流高，静态电流低的特点。选用 TCAN334 CAN 芯片,芯片兼容 ISO 11898 高速 CAN（控制器局域网）物理层标准。数据传输速率均高达 1Mbps。AD8418 电流放大器，AD8418 是一款高电压、高分辨率的电流分流器

3. 超级电容底板

电压基准芯片选用 TL431， TL431 是一种三端可调并联调节器，能提供极好的温度稳定性。该设备的动态输出阻抗为 0.2Ω,同时在许多应用中取代齐纳二极管与超级电容器 3V50F。

4.分电板

选用 XT30 和 PH1.25 端子。

1.5.3 软件设计

一、系统架构

1. 软件系统分成三层架构，硬件驱动层、功能模块层、应用层

（1）硬件驱动层：对单片机上不同功能的芯片和 IO 口进行使能驱动

（2）功能模块层：使用已使能的功能对外设进行驱动

（3）应用层：按照流程来调用不同的模块来进行工作

2. 日志系统使用 GitKraken 对软件程序进行迭代和记录

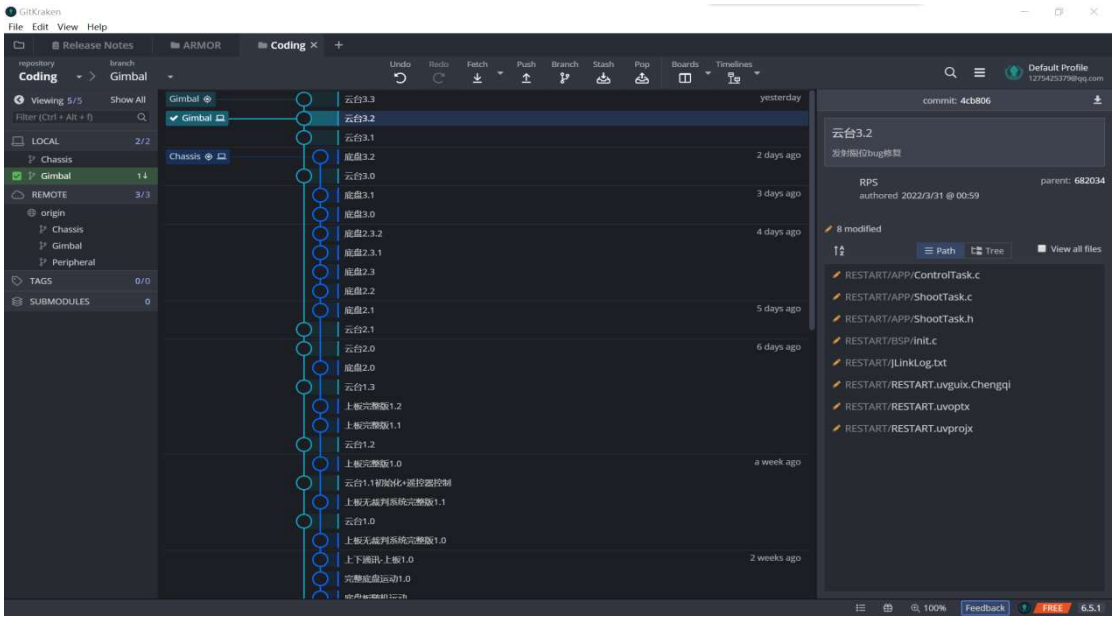


图 1-5-3-1 日志中的迭代记录

3. 开发环境使用 Keil uVision5 对 STM32 单片机进行开发调试

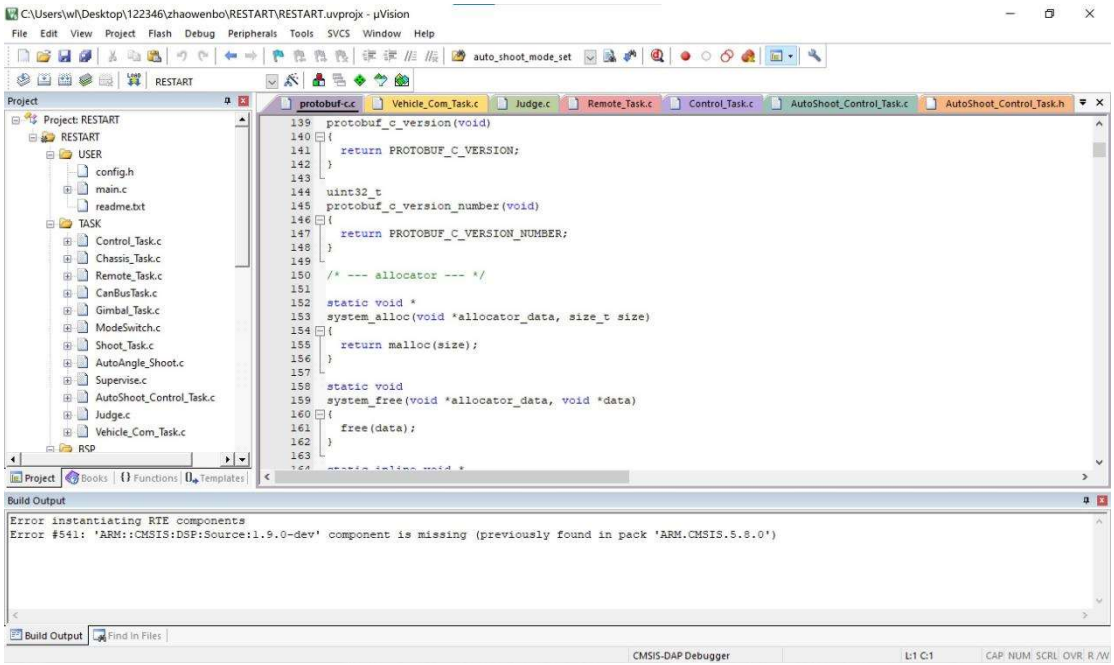
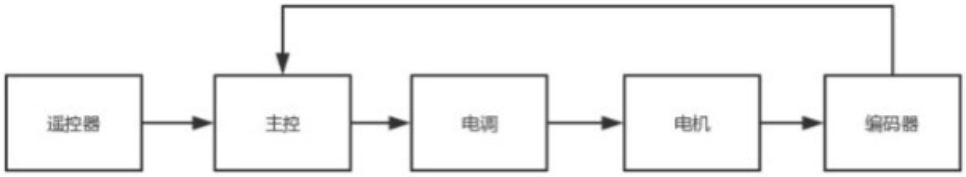


图 1-5-3-2 调试环境展示

二、运行流程

1. 单片机主控从 PC 上位机或遥控器接收指令，在受裁判系统模块的监测下通过 CAN 通信，将从电调收到的编码器反馈值经处理后和上位机发送的给定值经 PID 算法计算，将计算结果发向伺服电机，实现电机带动机械结构运动来实现机器人的功能。

图 1-5-3-3 控制系统框图



2. 单片机主控将从工业相机采集到的视觉图像信息通过串口通信经 Protobuf 通信协议打包发给 NVIDIA NX 进行计算。在从 NVIDIA NX 接收到数据包在解包得到自瞄数据参数后，经过单片机计算处理和通过 PID 控制算法发给云台控制电机进而改变双轴云台的 pitch 和 yaw 轴角度来实现对装甲板的自动瞄准功能。

三、重点功能

.

1. 成熟的舵轮底盘运动解算

已知：

四个轮组初始角度的设定 呈 45° 角，以 id 区分初始角 θ_k，（k=0,1,2,3,）

自转角速度 w，由 yaw 轴 pid 给定

平移分速度 vx，vy，通过遥控器给定

求：

（1）平移速度 v，其方向角 θ 始终为正值角，范围 0~2π

$$v = \sqrt{vx^2 + vy^2}$$

$$\theta = \arctan\frac{vy}{vx}$$

代码实现：

```
1333  void remotecontrolangle()
1334  {
1335
1336    vx = chassis.vy;
1337    vy = chassis.vx;
1338    Chassis_angle.Remote_speed = sqrt((vx*vx)+(vy*vy));
1339  if(Chassis_angle.Remote_speed >= 50)
1340  {
1341    if(vx > 0)
1342    {
1343      Chassis_angle.Remote_angle = atan(vy / vx);
1344      if(Chassis_angle.Remote_angle < 0)
1345      {
1346        Chassis_angle.Remote_angle += 2*PI;
1347      }
1348    }
1349    else if(vx<0)
1350    {
1351      Chassis_angle.Remote_angle = atan(vy / vx);
1352      Chassis_angle.Remote_angle += PI;
1353    }
1354    else
1355    {
1356        if(vy < 0)
1357        {
1358        Chassis_angle.Remote_angle = 3 * PI /2;}
1359        else if(vy > 0)
1360        {
1361        Chassis_angle.Remote_angle = PI / 2;
1362        }
1363    }
1364  }
1365  }
```

角速度 w 的给定

```
43      yaw_num_get = -GMYawEncoder.ecd_angle / 360;
44    if (-GMYawEncoder.ecd_angle < 0)
45    {yaw_num_get -= 1;}
46  Chassis_angle.get_yaw_angle = (-GMYawEncoder.ecd_angle - yaw_num_get * 360) * ANGLE_TO_RAD;
47

204
205
206
207
208
209
210
211  if (chassis.follow_gimbal&&gim.ctrl_mode!=GIMBAL_AUTO_SMALL_BUFF&&gim.ctrl_mode!=GIMBAL_AUTO_BIG_BUFF)
212    {
213       Chassis_angle.get_speedw = pid_calc(&pid_chassis_angle,get_yaw_angle,vw_homing);
214    }
215  else
216    {
217       Chassis_angle.get_speedw = 0;
218    }
219
220  if(fabs(pid_chassis_angle.get-pid_chassis_angle.set)<0.05)
221  {
222    Chassis_angle.get_speedw = 0;
223  }
224  }
```

（2）平移速度方向角与各初始角的夹角 θ_{ik}，（k=0,1,2,3）

平移自转和速度的大小 v_{sk}，（k=0,1,2,3）

相对于初始角的偏航角 θ_{dk}，（k=0,1,2,3）

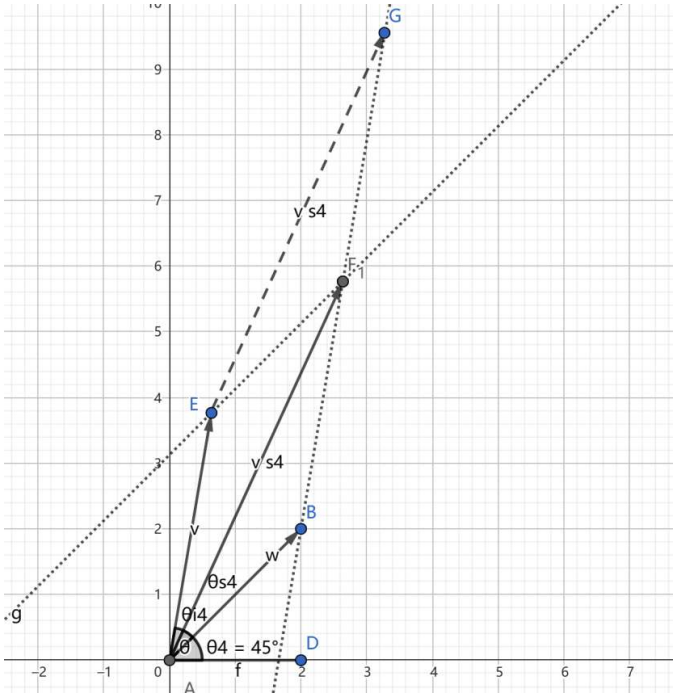
$$v_{sk} = \sqrt{\omega^2 + v^2 - 2\omega v \cos\theta_{ik}}$$

$$\theta_{ik} = \pi + \theta - \theta_k$$

正弦定理关系

$$\frac{v_{sk}}{\sin\theta_{ik}} = \frac{v}{\sin\theta_{dk}}$$

以 k=4 的情况为例，几何关系如下：



代码实现：

```
946   void start_angle_handle(void)
947   {
948   //leftangle=Chassis_angle.get_speedw/4000;
949   //VAL_LIMIT(leftangle,-0.15,0.15);
950   if(Chassis_angle.get_mode_flag==2)
951   {
952   if(Chassis_angle.get_speedw>0)
953   {Chassis_angle.get_yaw_angle+=leftangle;}
954   else if(Chassis_angle.get_speedw<0)
955   {Chassis_angle.get_yaw_angle+=leftangle;}
956   }
957   //  Chassis_angle.get_speedw=Chassis_angle.get_speedw*2;
958   Chassis_angle.start_angle[0] = Chassis_angle.get_yaw_angle + ((3*PI)/4);
959   Chassis_angle.start_angle[1] = Chassis_angle.get_yaw_angle + ((5*PI)/4);
960   Chassis_angle.start_angle[2] = Chassis_angle.get_yaw_angle + ((7*PI)/4);
961   Chassis_angle.start_angle[3] = Chassis_angle.get_yaw_angle + ((1*PI)/4);
962
963   for(int k=0;k<4;k++)
964   {
965
966     if(Chassis_angle.start_angle[k]>= 2*PI)
967     {Chassis_angle.start_angle[k] -= 2*PI;}
968     else if(Chassis_angle.start_angle[k] < 0)
969     {Chassis_angle.start_angle[k] += 2*PI;}
970   }
971 for(int k=0;k<4;k++)
972 {
973  Chassis_angle.include_angle[k] = PI + Chassis_angle.Remote_angle - Chassis_angle.start_angle[k];
974   if(Chassis_angle.Remote_speed <= 50)
975   {Chassis_angle.Remote_speed = 0;}
976     if(Chassis_angle.include_angle[k]>(2*PI))
977   {Chassis_angle.include_angle[k] -= 2*PI;}
978   else if(Chassis_angle.include_angle[k] <0)
979   {Chassis_angle.include_angle[k] += 2*PI;}
980 Chassis_angle.handle_speed[k] = sqrt(Chassis_angle.get_speedw*Chassis_angle.get_speedw +
981 Chassis_angle.Remote_speed*Chassis_angle.Remote_speed - 2*Chassis_angle.get_speedw*Chassis_angle.Remote_speed*(cos(Chassis_angle.include_angle[k])));
```

```
988 for(int k=0;k<4;k++)
989 {
990
991   transition1[k] = sin(Chassis_angle.include_angle[k]);
992   transition2[k] = Chassis_angle.Remote_speed*transition1[k];
993   transition3[k] = (transition2[k])/Chassis_angle.handle_speed[k];
994   VAL_LIMIT(transition3[k],-1,1);
995   Chassis_angle.deviation_angle[k] = asin(transition3[k]);
996
997   retransition2[k] = Chassis_angle.get_speedw*Chassis_angle.get_speedw+Chassis_angle.handle_speed[k]*Chassis_angle.handle_speed[k];
998   retransition3[k] = Chassis_angle.Remote_speed*Chassis_angle.Remote_speed;
999
1000  if(Chassis_angle.get_speedw==0)
1001  {
1002  if(Chassis_angle.include_angle[k]>=0&&Chassis_angle.include_angle[k]<=(PI/2))
1003  {
1004  Chassis_angle.deviation_angle[k] += PI;
1005  Chassis_angle.deviation_angle[k] = -Chassis_angle.deviation_angle[k];
1006  }
1007  else if(Chassis_angle.include_angle[k]>=(3*PI/2)&&Chassis_angle.include_angle[k]<=(2*PI))
1008  {Chassis_angle.deviation_angle[k] += PI;
1009  Chassis_angle.deviation_angle[k] = -Chassis_angle.deviation_angle[k];}
1010  }
1011
1012  else
1013  {
1014  if(Chassis_angle.get_speedw<0)
1015  {Chassis_angle.deviation_angle[k] += PI;
1016  Chassis_angle.deviation_angle[k] = -Chassis_angle.deviation_angle[k];}
1017
1018  if(retransition3[k]>retransition2[k])
1019  {Chassis_angle.deviation_angle[k] += PI;
1020  Chassis_angle.deviation_angle[k] = -Chassis_angle.deviation_angle[k];}
1021  }
```

至此解算完成，进行过零处理后，分别控制 6020 偏航舵向和 3508 轮速即可进行舵轮运动。

2. 与 NVIDIA NX 通信

从 NVIDIA NX 接收到数据包使用 Google 的 Protobuf 通信协议解包，通过对预先设定好的帧头、帧尾进行校验来判断收到的是否为有效数据，经 CRC 校验算法对两位 CRC 校验位进行计算处理来判断接收到的有效数据是否丢帧或有误。在校验完成后，从数据段中获取视觉图像坐标参数和自瞄测距信息。

3. 自瞄云台控制的优化

电控与视觉的通信中，视觉以 50hz 的频率发数据，但电控的云台控制频率是 500hz，这会导致在视觉看来，他们发的曲线是平滑的，但是在电控看来这却是由一个个方波组成的锯齿状曲线，这会导致云台的控制非常卡顿，一点也不丝滑，甚至会在装甲板上剧烈抖动，而我们利用卡尔曼滤波器异步预测同步观测的方法，在视觉数据未到的时候跑先验估计，在视觉数据到的时候跑滤波，由此将数据插到了 1000hz。

自瞄的响应还不够迅速，难以跟上高速移动的装甲板，因此在去年前馈的作用下我们通过设计卡尔曼滤波器在除估计角位置，和角速度外同时估计角加速度，并将估计出来的角速度作为速度环的前馈。

卡尔曼滤波公式的推导

① 先验估计

基于上一次最优估计与模型向前预测一步

$$\widehat{\boldsymbol{x_k^-}} = \boldsymbol{A\widehat{x_{k-1}} + Bu_{k-1}}$$

② 最优估计

因为 $z_k = Hx_k$

所以 $x_{kmea} = H^- z_k$

由前面的递归公式

$\widehat{x_k} = \widehat{x_k^-} + G(x_{kmea} - \widehat{x_k^-})$

G->1 $\widehat{x_k} = x_{kmea}$

G->0 $\widehat{x_k} = \widehat{x_k^-}$

所以

$\widehat{x_k} = \widehat{x_k^-} + G(H^- z_k - \widehat{x_k^-})$

令 G = $K_K H$

$\boldsymbol{\widehat{x_k} = \widehat{x_k^-} + K_K(z_k - H\widehat{x_k^-})}$

由此K_K即为我们要找的卡尔曼增益

③ 卡尔曼增益与先验误差协方差

卡尔曼滤波最厉害的地方便在于卡尔曼增益的求解与过程噪声的方差和观测噪 声的方差相挂钩，最终得出一个最优的权重，我们当然希望在做数据融合的过程中方差较大的那一部分数据占比较小，方差较小的那一部分数据占比较大，因此引入了过程噪声协方差矩阵 Q 和观测噪声协方差矩阵 R，分别用来衡量先验估计值的方差和测量值的方差。

除此之外，我们还需要一个指标来衡量什么卡尔曼增益为最优权重，因此我们引入误差协方差矩阵 P，当误差协方差矩阵 P 的迹最小时，误差的方差最小，也是最理想的，此时的K_K就是最优解。接下来开始K_K的推导，化简过程繁杂，这里给出联立方程与结论：

$$\begin{cases} P_k = E[e_k\, e_k{}^T] \\ e_k = x_k - \hat{x}_k \\ e_k^- = x_k - \hat{x}_k^- \\ \widehat{x_k} = \widehat{x_k^-} + K_K(z_k - H\widehat{x_k^-}) \\ Z_k = HX_k + V_k \\ P_k^- = E[e_k^-\, e_k^{-T}] \\ \mathrm{R} = \mathrm{E}[\mathrm{v}_k\, v_k{}^T] \end{cases}$$

最终我们化简完成，得出P_k的表达：

$$P_k = P_k^- - K_K H P_k^- - P_k^- H^T K_K{}^T + K_K H P_k^- H^T K_K{}^T + K_K R K_K{}^T$$

两边同时求迹：

$$tr(P_k) = tr(P_k^- - K_K H P_k^- - P_k^- H^T K_K{}^T + K_K H P_k^- H^T K_K{}^T + K_K R K_K{}^T)$$

然后我们对$tr(P_k)$关于K_K求导：

$$\frac{dtr(P_k)}{dK_K} = -P_k^- H^T + K_K\left(H P_k^- H^T + R\right) = 0$$

最终解得：

$$\boldsymbol{K_K = \frac{P_k^- H^T}{H P_k^- H^T + R}}$$

这就是卡尔曼增益的最终计算公式。

但是这个公式中还有一个未知量先验误差协方差矩阵P_k^-，接下来给出先验误差协方差矩阵的计算：

$$\begin{cases} P_k^- = E[e_k^- \, e_k^{-T}] \\ P_{k-1} = E[e_{k-1} e_{k-1}{}^T] \\ e_k^- = x_k - \hat{x}_k^- \\ \widehat{x_k^-} = A\widehat{x_{k-1}} + B u_{k-1} \\ X_k = AX_{k-1} + BU_k + W_{k-1} \\ \mathrm{Q} = \mathrm{E}[W_{k-1} \, W_{k-1}{}^T] \end{cases}$$

最终得出：

$$\boldsymbol{P_k^- = A P_{k-1} A^T + Q}$$

我们将P_k^-的计算带入K_K得出：

$$\boldsymbol{K_K = \frac{(A P_{k-1} A^T + Q) H^T}{H(A P_{k-1} A^T + Q) H^T + R}}$$

由此我们可以分析：

K_K的计算，仅由P_{k-1}，Q,R,决定，若 QR 阵皆为常数，决定卡尔曼收敛的过程仅仅在卡尔曼初期完成，得出最优卡尔曼增益，收敛完成后，各项数据将不再变动。

④ 超参数 QR 阵参数的整定

$$\boldsymbol{K_K = \frac{(A P_{k-1} A^T + Q) H^T}{H(A P_{k-1} A^T + Q) H^T + R}}$$

$$\boldsymbol{\widehat{x_k} = \widehat{x_k^-} + K_K(z_k - H\widehat{x_k^-})}$$

我们可以分析：

当过程噪声更大即 Q 更大的时候$\boldsymbol{K_K}$趋于 1，而最优估计则更接近$\boldsymbol{z_k}$

此时卡尔曼滤波器更加信任传感器，响应速度更快同时噪声也会更多。

当观测噪声更大即 R 更大的时候$\boldsymbol{K_K}$趋于 0，而最优估计则更接近$\boldsymbol{\widehat{x_k^-}}$

此时卡尔曼滤波器更加信任模型预测，响应速度更慢同时噪声少数据平滑。

而 Q 与 R 矩阵就是我们需要整定的参数

R 阵的整定较为容易，市面上购买的传感器大多都会告诉我们方差是多少，我们只需要按照 z 矩阵的顺序在对角阵 R（一般都是对角阵）中依次输入方差即可。

Q 阵的求解则同可建模噪声与不可建模噪声相关，同样以步兵自瞄为例：

$$\begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ v_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}at^2 \\ at \end{bmatrix}$$

$$z_k = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix}$$

在这个模型中，我们将加速度作为可建模噪声建模出来，那么：

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{2}at^2 \\ at \end{bmatrix}$$

$$Q = E\left[\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} [w_1 w_2] \right]$$

$$= E \begin{bmatrix} \frac{1}{4}a^2t^4 & \frac{1}{2}a^2t^3 \\ \frac{1}{2}a^2t^3 & a^2t^2 \end{bmatrix}$$

$$= E \begin{bmatrix} \frac{1}{4}t^4 & \frac{1}{2}t^3 \\ \frac{1}{2}t^3 & t^2 \end{bmatrix} D(a)$$

因此我们只需拟定一个加速度的方差即可，在实际调车中也要以实际效果为准。

⑤ 先验误差协方差的更新

在完成最优估计的计算后，我们需要完成对误差协方差矩阵的更新：

$$\begin{cases} P_k = P_k^- - K_K H P_k^- - P_k^- H^T K_K{}^T + K_K H P_k^- H^T K_K{}^T + K_K R K_K{}^T \\ \boldsymbol{K_K = \dfrac{P_k^- H^T}{H P_k^- H^T + R}} \end{cases}$$

联立解出：

$$\boldsymbol{P_k = (I - K_K H) P_k^-}$$

⑥ 卡尔曼滤波五大公式的闭合

预 测	更 新
$\boldsymbol{\widehat{x_k^-} = A\widehat{x_{k-1}} + Bu_{k-1}}$	$\boldsymbol{K_K = \dfrac{(AP_{k-1}A^T + Q)H^T}{H(AP_{k-1}A^T + Q)H^T + R}}$
$\boldsymbol{P_k^- = AP_{k-1}A^T + Q}$	$\boldsymbol{\widehat{x_k} = \widehat{x_k^-} + K_K(z_k - H\widehat{x_k^-})}$

$$\boldsymbol{P_k = (I - K_K H) P_k^-}$$

先预测后更新，循环往复。

⑦ 基于上述理论，我们设计了一个卡尔曼滤波器

$$\begin{bmatrix} yawangle_k \\ pitchangle_k \\ yawspd_k \\ pitchspd_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & t & 0 \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} yawangle_{k-1} \\ pitchangle_{k-1} \\ yawspd_{k-1} \\ pitchspd_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}at^2 \\ at \\ \frac{1}{2}at^2 \\ at \end{bmatrix}$$

$$\begin{bmatrix} yawangle \\ pitchangle \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} yawangle_k \\ pitchangle_k \\ yawspd_k \\ pitchspd_k \end{bmatrix}$$

⑧ 代码实现

```
#include "main.h"
float autoshoot_kf_flag = 0;

autoshoot_kalman_filter_t autoshoot_kalman_filter =
{
  .Q_data = {
              1 , 0 , 0 , 0 ,
              0 , 1 , 0 , 0 ,
              0 , 0 , 9 , 0 ,
              0 , 0 , 0 , 0.1 ,

            },
  .R_data = {
              20 , 0 ,
              0 , 10 ,

            },
  .A_data = {
              1 , 0 , Dt , 0 ,
              0 , 1 , 0 , Dt ,
              0 , 0 , 1 , 0  ,
              0 , 0 , 0 , 1  ,

           },
  .H_data = {
              1 , 0 , 0 , 0 ,
              0 , 1 , 0 , 0 ,

            },
  .I_data = {
              1 , 0 , 0 , 0 ,
              0 , 1 , 0 , 0 ,
              0 , 0 , 1 , 0 ,
              0 , 0 , 0 , 1 ,
```

```
115
116   //1. xhat'(k)= A xhat(k-1)
117   mat_mult(&B->b_A, &B->b_xhat,&TEMP51);
118   mat_copy(TEMP51_data,B->xhat_data,5);
119
120   if((abs(B->b_Z.pData[0] - B->b_xhat.pData[0])>=20)|(abs(B->b_Z.pData[1] - B->b_xhat.pData[1])>=15))
121     autoshoot_kalman_filter_reset(&autoshoot_kalman_filter);
122
123   if(autoshoot_kf_flag)
124   {
125   //2. P'(k) = A P(k-1) AT + Q
126   mat_mult(&B->b_A, &B->b_P,&TEMP);
127   mat_mult(&TEMP,&B->b_AT,&TEMP2);
128   mat_add(&TEMP2,&B->b_Q,&B->b_P);
129
130   //3. K(k) = P'(k) HT / (H P'(k) HT + R)
131   mat_mult(&B->b_P,&B->b_HT,&TEMP53);
132
133   mat_mult(&B->b_H,&B->b_P,&TEMP35);
134   mat_mult(&TEMP35,&B->b_HT,&TEMP33);
135   mat_add(&TEMP33,&B->b_R,&TEMP_33);
136   mat_inv(&TEMP_33,&TEMP33);
137
138   mat_mult(&TEMP53,&TEMP33,&B->b_K);
139
140   //4. xhat(k) = xhat'(k) + K(k) (z(k) - H xhat'(k))
141   mat_mult(&B->b_H,&B->b_xhat,&TEMP_3);
142   mat_sub(&B->b_Z,&TEMP_3,&TEMP__3);
143   mat_mult(&B->b_K,&TEMP__3,&TEMP51);
144   mat_add(&B->b_xhat,&TEMP51,&TEMP_51);
145   mat_copy(TEMP_51_data,B->xhat_data,5);
146   //5. P(k) = (1-K(k)H)P'(k)
147   mat_mult(&B->b_K,&B->b_H,&TEMP);
148   mat_sub(&B->b_I,&TEMP,&TEMP2);
149   mat_mult(&TEMP2,&B->b_P,&TEMP);
```

4.功率限制

步兵机器人的速度主要取决于给轮组电机的电流大小以及电机转速比，减速比更改的影响颇多，此处按下不表，而电流的大小由于功率限制的存在无法无限增大。为了提高功率的利用率，我们加入了超级电容模块。超级电容由超级电容组、超级电容管理模块以及电容控制板组成，可以将其认为是一个较大的储能电池，在比赛过程中不停的充电、放电，以此提供给机器人底盘较大的电流。

```
109 //获得ADC值
110 //ch: @ref ADC_channels
111 //通道值 0~16取值范围为：ADC_Channel_0~ADC_Channel_16
112 //返回值:转换结果
113 u16 Get_Adc(u8 ch)
114 {
115       //设置指定ADC的规则组通道，一个序列，采样时间
116   ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_3Cycles ); //ADC1,ADC通道,28个周期,提高采样时间可以提高精确度
117   ADC_SoftwareStartConv(ADC1);     //使能指定的ADC1的软件转换启动功能
118   while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束
119   return ADC_GetConversionValue(ADC1);  //返回最近一次ADC1规则组的转换结果
120 }
121
122 u16 Get_Adc_2(u8 ch)
123 {
124       //设置指定ADC的规则组通道，一个序列，采样时间
125   ADC_RegularChannelConfig(ADC2, ch, 1, ADC_SampleTime_3Cycles ); //ADC1,ADC通道,28个周期,提高采样时间可以提高精确度
126   ADC_SoftwareStartConv(ADC2);     //使能指定的ADC1的软件转换启动功能
127   while(!ADC_GetFlagStatus(ADC2, ADC_FLAG_EOC ));//等待转换结束
128   return ADC_GetConversionValue(ADC2);  //返回最近一次ADC1规则组的转换结果
129 }
130 //获取通道ch的转换值，取times次,然后平均
131 //ch:通道编号
132 //times:获取次数
133 //返回值:通道ch的times次转换结果平均值
134 u16 Get_Adc_Average(u8 ch,u8 times)
135 {
136   u32 temp_val=0;
137   u8 t;
138   for(t=0;t<times;t++)
139   {
140     temp_val+=Get_Adc(ch);
```

超级电容 ADC 模块代码

对电机性能进行分析计算，计算理论上电机所消耗的功率，并通过控制底盘电机的给定来限制超级电容的输出功率，使超级电容发送的抽入功率的最大值不会超出裁判系统给定的功率上限。

对于底盘电机单个底盘电机的输出功率大致等于

$$k1\times V\times I+f2\times I^2+f1\times I+f0$$

其中 k1 和 f2、f1、f0 都是比例系数需要实验测得，V 和 I 分别是电机速度和给定电流，设底盘电机的速度分别为 $V1$、$V2$、$V3$、$V4$，电机给定电流为 $I1$、$I2$、$I3$、$I4$。则底盘功率

$$W = \sum_{i=1}^{4} k1 \times Vi \times Ii + \mathrm{f}2 \times Ii^2 + f1 \times Ii + f0$$

其中 $k1 \times Vi \times Ii$ 称为 drive-power, $f2 \times Ii^2 + f1 \times Ii + f0$ 称为 heat-power;

Ii 需要通过 pid 控制给出，当给定速度确定时，Ii 就被确定了，而 Vi 是此时此刻的电机速度。因次只需要控制电机的给定速度就可以达到控制功率的目的。设最后乘以给定速度的系数为 k 可用来控制底盘速度给定从而控制功率。

```
482       if(PowerSum >(float) Max_Power)
483         {
484           //设中间变量i_n=a[n]*k+b[n]
485           float a[4];
486           for(int i=0; i<4; i++)
487             a[i]=(float)chassis.wheel_speed_ref[i]*(pid_spd[i].p+pid_spd[i].d);
488           float b[4];
489           for(int i=0; i<4; i++)
490             b[i]=-pid_spd[i].p*(float)chassis.wheel_speed_fdb[i]+pid_spd[i].iout \
491                  -pid_spd[i].d*(float)chassis.wheel_speed_fdb[i]-pid_spd[i].d*pid_spd[i].err[LAST];
492           // Max_power=heat_power+drive_power
493           //  i_n=a[n]*k+b[n] 代入
494           //Max_Power=m*k^2+n*k+o
495           //0=m*k^2+n*k+l(l=o-Max_Power)
496
497           float m=(a[0]*a[0]+a[1]*a[1]+a[2]*a[2]+a[3]*a[3])*FACTOR_2;
498
499           float n=2*FACTOR_2*(a[0]*b[0] + a[1]*b[1] + a[2]*b[2] + a[3]*b[3]) + \
500                   FACTOR_1*(a[0] + a[1] + a[2] + a[3]) + \
501                   I_TIMES_V_TO_WATT*(a[0]*(float)chassis.wheel_speed_fdb[0] + \
502                                      a[1]*(float)chassis.wheel_speed_fdb[1] + \
503                                      a[2]*(float)chassis.wheel_speed_fdb[2] + \
504                                      a[3]*(float)chassis.wheel_speed_fdb[3]);
505
506           float l=(b[0]*b[0] + b[1]*b[1] + b[2]*b[2] + b[3]*b[3])*FACTOR_2 + \
507                   (b[0] + b[1] + b[2] + b[3])*FACTOR_1 + \
508                   I_TIMES_V_TO_WATT*(b[0]*(float)chassis.wheel_speed_fdb[0] + \
509                                      b[1]*(float)chassis.wheel_speed_fdb[1] + \
510                                      b[2]*(float)chassis.wheel_speed_fdb[2] + \
```

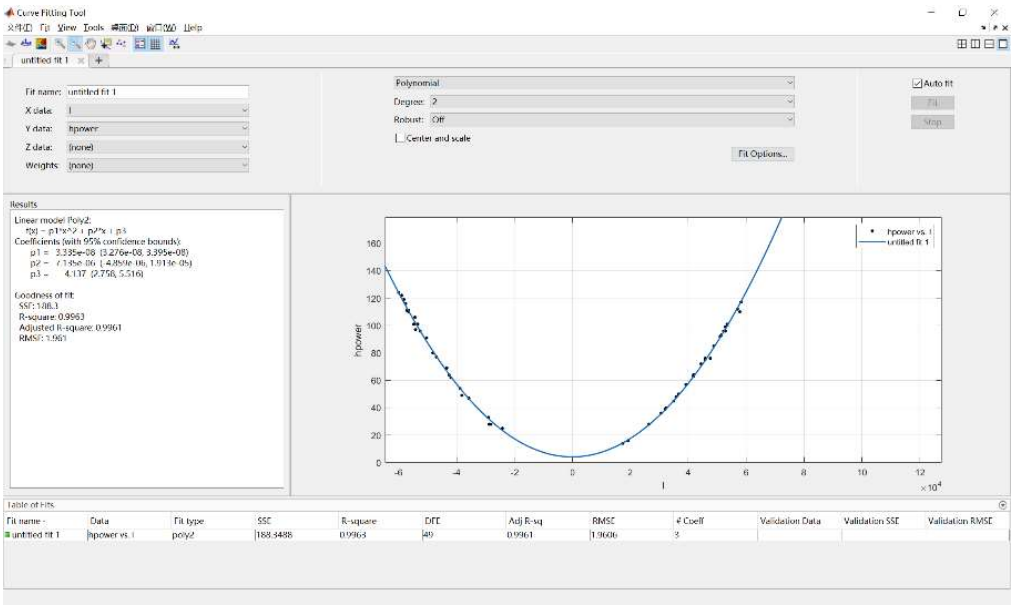
按上述式子测出比例系数，使用 MATALAB 对底盘电机的功率曲线进行理论仿真，通过调整电机功率方程的参数 K 实现对电机功率曲线的拟合。

经测试得到的步兵的功率方程参数

```
26 //电机发热计算 p=i^2*FACTOR_2+i*FACTOR_1+FACTOR0;
27 //i是直接发给电调的数，也是pid计算出来的out,但不可以直接用pid.out，需要自己根据数据手册换算出电机电流，C620电调控制电流范围16384~16384
28 //通过使用示波器读值与matlab拟合
29 //(多种参数均可拟合功率限制曲线)
30 #define FACTOR_2  0.00000016f//0.0000000000561f//
31 #define FACTOR_1  -0.0000229f//-0.0000006107f//
32 #define FACTOR_0  0.458f//0.006234f//
```

使用 MATALAB 对底盘电机的功率曲线进行理论仿真，通过调整电机功率方程的参数

实现对电机功率曲线的拟合。



5.PID 算法控制电机

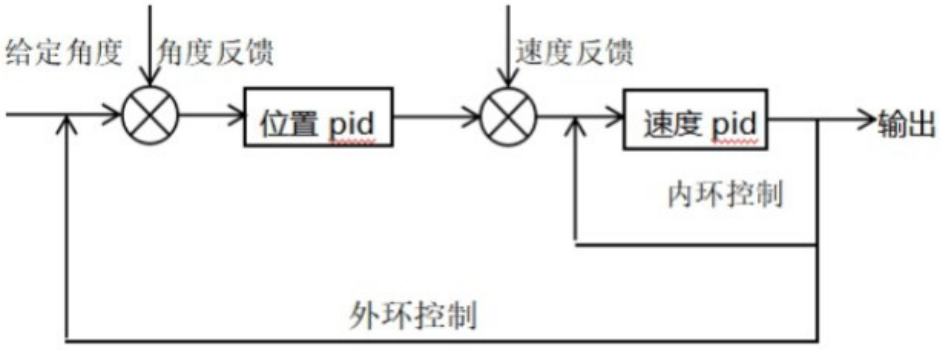
步兵机器人上使用的控制电机的算法是闭环 PID 算法。闭环算法是将控制对象电机的输出量进行负反馈来实现对输出校正的控制方式，它是在测量出实际反馈与计划给定之间发生偏差时，经 PID 运算来进行纠正的。PID 计算算法根据输入输出的偏差值，按照比例环节、积分环节、微分环节的函数关系进行运算，运算结果经 CAN 通信发向电机的编码器来控制调节电机输出，使电机的输出跟随给定输入的变化。

```
83  float pid_calc(pid_t *pid, float get, float set)
84  {
85      pid->get = get;
86      pid->set = set;
87      pid->err[NOW] = set - get;
88
89      if ((pid->input_max_err != 0) && (fabs(pid->err[NOW]) > pid->input_max_err))
90          return 0;
91
92      if (pid->pid_mode == POSITION_PID) //position PID
93      {
94          pid->pout = pid->p * pid->err[NOW];
95          pid->iout += pid->i * pid->err[NOW];
96          pid->dout = pid->d * (pid->err[NOW] - pid->err[LAST]);
97
98          abs_limit(&(pid->iout), pid->integral_limit);
99          pid->out = pid->pout + pid->iout + pid->dout;
100         abs_limit(&(pid->out), pid->max_out);
101     }
102     else if (pid->pid_mode == DELTA_PID) //delta PID
103     {
104         pid->pout = pid->p * (pid->err[NOW] - pid->err[LAST]);
105         pid->iout = pid->i * pid->err[NOW];
106         pid->dout = pid->d * (pid->err[NOW] - 2 * pid->err[LAST] + pid->err[LLAST]);
107
108         pid->out += pid->pout + pid->iout + pid->dout;
109         abs_limit(&(pid->out), pid->max_out);
110     }
111
```

6.优化单环 PID 算法控制电机

由于 PID 单环控制算法无法实现精确控制电机转动角度，但可以实现精确控制电机转动速度。而且我们实际上只对电机角度有一个较高精度的期望，但对电机的升力、速度没有期望。因此我们希望能通过优化单环 PID 算法实现通过控制速度达到控制位置的效果，即我们给电机发送的期望速度必须能减小位置误差，且在电机位置误差为 0 时，期望速度也为 0。故通过增加一个反馈闭环和再通过一次闭环 PID

控制算法即双环串级 PID 来改善电机在转动固定角度时的动态性能指标。



7.陀螺仪解算

读取云台姿态时使用三轴陀螺仪，陀螺仪根据加速度计和地磁计的数据，转换到地理坐标系后，与对应参考的重力向量和地磁向量进行求误差，这个误差用来校正陀螺仪的输出，然后用陀螺仪数据进行四元数更新，再转换到欧拉角。得到角度后单片机主控通过给定目标角度、获取实时角度信息，在 PITCH 轴、YAW 轴两个维度进行对云台的控制，以云台转动实现全方位旋转并且保持云台姿态稳定。

```
257  void Hi220_getYawPitchRoll()
258  {
259    volatile static float Last_yaw_temp,Yaw_temp,Last_pitch_temp,Pitch_temp,Last_roll_temp,Roll_temp; //
260    volatile static int Yaw_count,Pitch_count,Roll_count;
261
262    yaw_Gyro = -HI220_Data_From_Usart.Ang_Velocity_Z * 0.1f;
263    pitch_Gyro = -HI220_Data_From_Usart.Ang_Velocity_Y * 0.1f;
264    roll_Gyro = -HI220_Data_From_Usart.Ang_Velocity_X * 0.1f;
265
266    Last_yaw_temp = Yaw_temp;
267    Yaw_temp = -HI220_Data_From_Usart.Euler_Angle_Yaw_s16_2_f;
268    if(Yaw_temp-Last_yaw_temp>=330)
269    {
270      Yaw_count--;
271    }
272    else if (Yaw_temp-Last_yaw_temp<=-330)
273    {
274      Yaw_count++;
275    }
276  //  yaw_Angle = Yaw_temp + Yaw_count*360;
277
278  //  pitch_Angle = HI220_Data_From_Usart.Euler_Angle_Pitch_s16_2_f;   //************************去负号*****************************
279    Last_pitch_temp = Pitch_temp;
280    Pitch_temp = HI220_Data_From_Usart.Euler_Angle_Pitch.Euler_Angle_Pitch_f;;
281    if(Pitch_temp-Last_pitch_temp>=330)
282    {
283      Pitch_count--;
284    }
285    else if (Pitch_temp-Last_pitch_temp<=-330)
```

8.斜坡输入函数

斜坡函数的求解即根据 系统 的 当前值， ， 目标值和 延迟时间进行一次计算，得到斜坡函数需要执行的步数和斜坡函数的斜率。在开始驱动电机时，应用斜坡输入函数来替代阶跃函数，使输入信号变得更加平缓，减少系统超调量，从而优化系统的时间响应，达到优化动态性能指标的效果。

```
49  }
50    return rmb_accourq->out;
51  \\  rmb_accourq->out =  rmb_accourq->counr \ ((float)rmb_accourq->acsje);
52    rmb_accourq->out = rmb_accourq->counr * rmb_accourq->counr \ ((float)rmb_accourq->acsje * (float)rmb_accourq->acsje);
53
54      rmb_accourq->counr = rmb_accourq->acsje;
55    if (rmb_accourq->counr++ >= rmb_accourq->acsje)
56
57      return 0;
58    if (rmb_accourq->acsje <= 0)
59  {
60    float rmb_accourq_csjc(rmb_accourq_t *rmb_accourq)
```

9.监测任务

为防止系统因受到来自外界电磁场的干扰，造成各种寄存器和内存的数据混乱从而陷入死循环，导致整个系统的陷入停滞状态，发生不可预料的后果。现在步兵机器人单片机主控上使用一个定时器电路对整个系统的各个功能进行监控，一旦发生错误就向芯片发出重启信号，并将监控结果通过控制主控板上显示灯的亮灭来显示。

```c
uint32_t lost_err = 0xFFFFFFFF;     //每一位代表一个错误
void SuperviseTask()
{
  int i = 0;
//  if(Is_AppParam_Calied())   //校准过，则喂狗
//  {
//    LostCounterFeed(GetLostCounter(LOST_COUNTER_INDEX_NOCALI));
//  }
  //step 1;check the error
  LostCounterFeed(GetLostCounter(LOST_COUNTER_INDEX_NOCALI));
  LostCounterFeed(GetLostCounter(LOST_COUNTER_INDEX_ZGYRO));
  for(i = 0; i < LOST_COUNTER_NUM; i++)
    {
      if(LostCounterOverflowCheck(lost_counter[i], lost_counter_len[i]) == 0)    //4ms*200 = 2s error check time
        {
          LostCounterCount(&lost_counter[i], 1);   //add 1 everytime
          lost_err &= ~(uint32_t)(1 << i);    //clear the error bit
        }
      else
        {
          lost_err |= (uint32_t)(1 << i);    //set the error bit
        }
    }

  //step 2:action to error
  if(Get_Lost_Error(LOST_ERROR_NOCALI) == LOST_ERROR_NOCALI)
```

主控 LED 控制显示部分代码

```c
void LED_TOGGLE(uint8_t led, uint16_t CYCLE_S, uint16_t TOGGLE_S, uint16_t shine_times,uint16_t TIME_SPAN)//灯，循环，切换，发光时间，时间间隔
{
  static uint32_t timeSpanCount = 0;
  static uint32_t tempCount = 0;
  static uint32_t sec = 0;
  uint32_t sec_cycle = 0;
  //assert_param(shine_times>=0);
  timeSpanCount++;  //以时间间隔计数
  sec = timeSpanCount/(1000/TIME_SPAN);   // second count
  sec_cycle = sec % CYCLE_S;  //range: 0----(CYCLE_S-1)

  //0-TOGGLE_S-1 TOGGLE_S-CYCLE_S
  if(sec_cycle/TOGGLE_S == 0)    //process of the led toggle
    {
      tempCount++;
      if(shine_times <=0 )    //if shine times is negative
        {
          if(led == 0)
            {
              RED_LED_ON();
              GREEN_LED_OFF();
            }
          else
            {
              GREEN_LED_ON();
              RED_LED_OFF();
            }
        }
      else if(tempCount%(((1000/TIME_SPAN)*TOGGLE_S)/(2*shine_times)) == 0)     //tempCount可以被分母整除，就闪灯
```

1.5.4 算法设计

一、装甲板识别算法

1、功能简介

秉持着持续创新的观念，开发人员乐于尝试新兴技术，积极对现有算法进行改革，以适应不断变化的局势。同时，鉴于传统视觉方法的应用范围相对有限，遂决定采用深度学习算法以在确保准确性的基础上增强系统的稳健性。

23 赛季的装甲板识别算法是一款基于 yolox 的不规则四边形目标检测，可识别灯条的 4 个顶点和类别并定位其图像坐标的神经网络识别算法，其功能是使己方步兵机器人能够自动瞄准和预测敌方装甲板并进行有效射击，或在操作手处于复杂状况下辅助操作手瞄准从而起到一个优化击打的效果。

在总结 22 赛季识别算法抖动和帧率低的经验和教训后，开发者在本赛季的算法在保证了识别率的基础上中加入了防抖和提升帧率的功能。

2、重要算法原理阐述、公式推导

本识别算法采用了数据增强、focal loss 以及 smoothL1 loss 的使用，以及 Label Assignment 等策略。

①数据增强

由于是不规则四边形检测，本算法对数据标注有很大的要求，使数据标注费时费力。所以如何用好每一张数据集很关键。为此在本算法中数据增强包括了基础的图像旋转、缩放、翻转、裁剪、拼接等操作大大扩充了样本的数量，同时根据我们的任务需求和个人经验加入了更换目标背景、类红点干扰、类小弹丸遮挡、类大弹丸遮挡等相关功能来提升网络的鲁棒性。

图 1-5-4-1 类红点干扰、类大弹丸遮挡效果展示

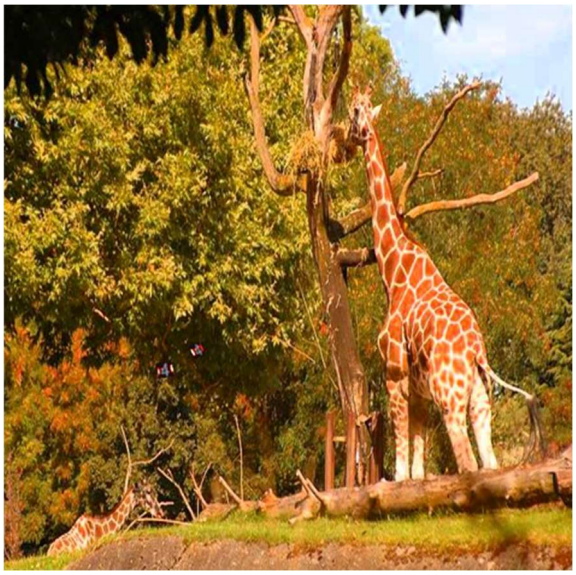


图 1-5-4-2 更换目标背景、传统数据增强效果展示

②损失函数

损失函数方面，本算法主要采用了 focal loss 和 smooth L1 loss。

本算法采用的数据集中一张图片大概率只有 1 个正例，而有 8400 个负例，导致网络把所有的都判成负例也会有很低的 loss。采用 Focal loss 能够平衡正负例带来损失的权重，缓解类别不均衡问题，从而使模型更加关注难分类的样本。

$$L_f(y) = \begin{cases} -\alpha(1-y)^{\gamma} logy, & y = 1 \\ -\alpha(y)^{\gamma} log(1-y), & y = 0 \end{cases}$$

Smooth L1 loss 则在减小训练时的噪声影响的同时，仍能够保持相对较好的训练效果，这也是目标检测任务中常用的损失函数。

$$Smooth_L1(x) = \begin{cases} -0.5x^2, & -1 < x < 1 \\ |x|, else \end{cases}$$

③标签匹配

标签匹配策略是指将每个目标和一个先验框进行匹配，以便模型能够预测目标的位置和类别。具体来说该算法采用和标签与正例一一对应的方法，即对于每个先验框，选择与其 IoU 值最大的真实目标作为正例进行匹配，其余则视为负例。此外，该算法还考虑过滤掉面积过小的目标，以兼顾小目标和网络总体性能的阈值过滤目标，这有助于提高目标检测的精度和鲁性。

④网络结构

在联盟赛阶段，由于基于 yolov7 的算法效果不佳，因此在分区赛和国赛期间开发者将工作重心转变为在保证识别效果的前提下提高网络帧率。作为一个推理任务，网络结构是影响神经网络推理速度的主要方面。在综合了各种网络结构之后开发者决定采用 yolox 的网络结构同时减少输入大小至 418*418 和输出目标类型至 10 个来提高网络的运行速率。经测试这种网络结构在保证了识别率的基础上对识别速率有了很大的提高。

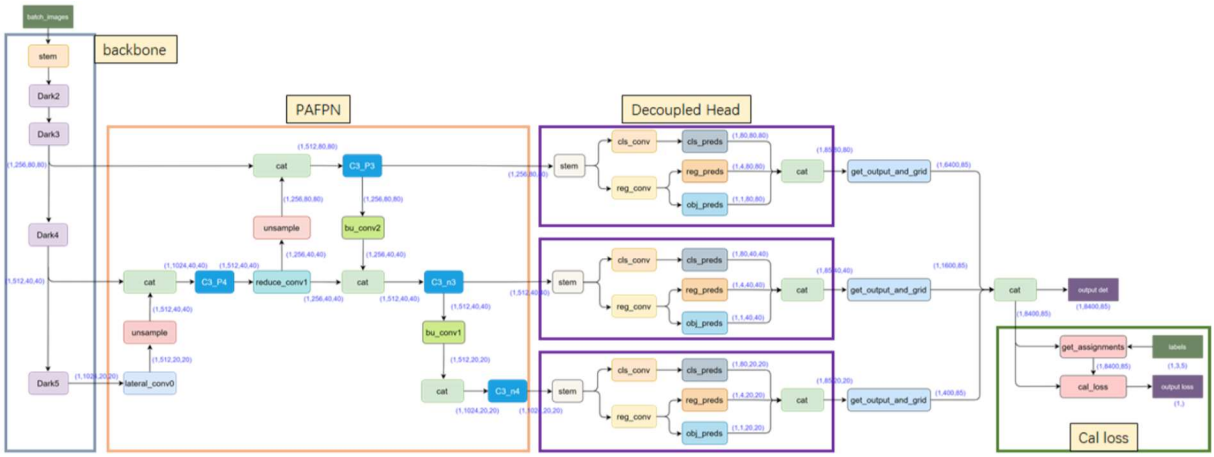


图 1-5-4-3 YoloX 网络结构

总体来说，该算法通过结合数据增强、损失函数和标签匹配选择合适的网络结构的策略等技术手段，成功提高目标检测任务的准确性和鲁棒性，为实际应用提供了更好的支持。

3、算法性能、优缺点分析、优化方案

算法性能：

系统环境	训练环境
windows11	torch 1.9.1+cu111
GPU	显存大小
NVIDIA GeForce RTX 3060 Laptop GPU	6143.5MB

表 1-5-4-1 网络训练环境表

平台	理论 FLOPS(FP32)
NVIDIA Jetson Xavier NX	844.8 G
推理线程	测试精度
单线程	FP16
网络结构	帧率
YoloX	~120

表 1-5-4-2 算法性能测试表

相关参数	网络模型：	opt-1208-001	测试目标：	R3	环境亮度：	108LUX	测试时间：	2023年5月25日	测试人员：	姚翔	测试效果：	好
镜头焦距	6mm						8mm					
曝光时间	4000		6000		8000		4000		6000		8000	
曝光增益	8	12	8	12	8	12	8	12	8	12	8	12
2m	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别
3m	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别
4m	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别
5m	点准确有1、2帧类误识	点准确有5、6帧类误识	点准确10帧以上的类误	点准确10帧以上的类误	点准确类误识别	点准确类误识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别	点和类精准的识别
6m	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点准确有1、2帧类误识	点准确有1、2帧类误识	点准确有1、2帧类误识	点准确有1、2帧类误识	点准确有5、6帧类误识	点准确有5、6帧类误识
7m	点准确类误识别	点准确类误识别	点准确类误识别	点准确类误识别	点准确类误识别	点准确类误识别	点准确有5、6帧类误识	点准确有5、6帧类误识	点准确10帧以上的类误	点准确有5、6帧类误识	点准确有5、6帧类误识	点准确有5、6帧类误识
8m	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点准确10帧以上的类误	点准确类误识别	点准确10帧以上的类误	点准确类误识别	点准确10帧以上的类误	点准确10帧以上的类误
9m	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点准确类误识别	点准确类误识别	点准确类误识别	点准确类误识别	点和类均不准确或不识	点准确类误识别
10m	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识
11m	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识	点和类均不准确或不识

图 1-5-4-4 部分算法识别效果测试结果

优点：

1) 上场需要调的参数少，减轻视觉组上场压力和人员。
2) 鲁棒性强，减少误识别的可能性，更稳定。
3) 网络的容错度高可以在赛场上续训出新的网络完成迭代。

缺点：

1) 受输入大小影响导致对远处目标的检出率较低。
2) 对数据集要求较高，需要大量人员仔细漫长地标注。
3) 对设备要求较高，需要需要英伟达的大显存边缘计算设备。

优化方案：

1）由于中、大型的输出规模也将包括小的 anchor，因为数据集不包括大的目标。这里，标签重写的问题将逐步升级，因为需要在粗网格中检测小目标。反之亦然。大目标将被检测在小和中等输出层。在这里，检测将不会是精确的，因为中小输出层有有限的感受野。

第一种方法是为三个输出尺度定义感受野，并定义两个阈值来分隔它们。然 k-means 将根据这些阈值计算聚类中心(用作 anchor)。缺点是这样把数据驱动的 anchor 转成了问题驱动，只能在固定的一些尺度上检测对应的目标，而不是全尺度，这样会浪费网络。

第二种方法是创建一个具有单个输出的体系结构，该输出将聚合来自各种 scale 的信息。然后一次性处理所有的 anchor。与第一种方法相比，anchor 大小的估计将还是保持由数据驱动。

2）修改正样本匹配规则考虑忽略一些样本，之所以我们会忽略那些和目标框的 IoU 超过了阈值的 anchor box，就是因为这些样本其实也是比较好的样本（超过了我们给定的阈值），若是就这么作为负样本，直观上就觉得不太合适，因此我们就忽略掉。

第一，multi_anchor 策略，即只要 anchor box 和目标框的 IoU 超过了给定的阈值，就作为正样本，反之为负样本。没有忽略样本处在中间。

第二，Max IoU 策略，正样本就选择 IoU 最大的，其余全部作为负样本处理。同样，没有忽略样本处在中间。

1. 算法结果

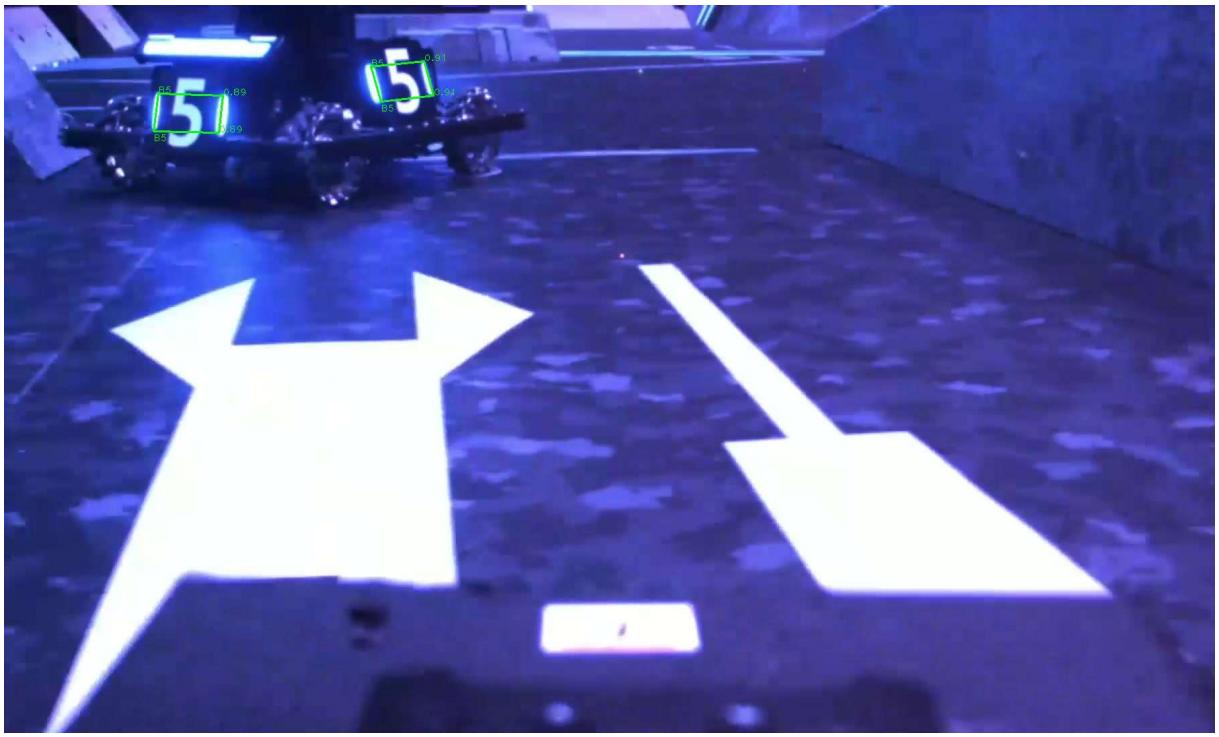


图 1-5-4-5 部分识别算法结果展示（蓝色/多目标/清晰）

图 1-5-4-6 部分识别算法结果展示（红色/单目标/模糊）

二、目标预测算法

1、功能简介

由于相机采图，神经网络推理，数据传输，电控控制等具有一定的延迟，如果直接使用识别到的点进行弹道解算，在装甲板快速移动时，自瞄对装甲板的追踪会有一定的滞后导致无法命中目标，因此需要预

测算法根据历史状态对下一时刻的装甲板板的位姿进行预测使云台在目标在快速移动时也可以快速跟随目标对目标进行精确的打击。

秉承着“整体框架不变，小步快跑迭代”的研发思路，23 赛季的目标预测算法总体上与 22 赛季保持不变，即使用拓展卡尔曼滤波和匀速运动模型对目标装甲板进行预测，但在细节上，我们进行了优化：

在数据源上，我们采用了硬触发对齐相机和陀螺仪的时间戳，减少了数据源的噪声，提高了预测算法的响应速度，大大减少了预测结果的噪声；

在算法的运行上，我们采用了多线程优化了运行速度，加快了收敛过程，提高了对移动目标的命中率。

2、重要算法原理阐述、公式推导

①硬触发

在通过 pnp 解算装甲板的位姿后，我们可以得到装甲板在相机坐标系下的三维坐标，但由于云台在不断的转动，即使装甲板不移动，由于云台的转动也会使装甲板在相机坐标系下的三维坐标发生变化，因此需要将装甲板的三维坐标从移动的坐标系转到不动的坐标系——世界坐标系（即陀螺仪坐标系）以便后面的预测算法的正常运行。这两个坐标系的变化需要装甲板在相机坐标系下的位姿加上一个平移矩阵和乘上一个旋转矩阵得到。由于相机在云台上的安装位置是固定的，因此平移矩阵是一个固定的值。而平移矩阵需要得到云台的转角来得到——即电控通过串口给视觉发送陀螺仪的数据来实时获得云台的位置。但这也同时引出了另一个问题：由于串口的发送和相机的采图不是同步的，因此装甲板的三维位置和对应的旋转矩阵可能在时间上不对应，在云台高速转动时会带来较大的误差，而通过硬触发可以解决这个问题。

硬触发就是利用 IO 或者光耦等外部控制信号触发相机快门的技术。简单来说就是把控制相机快门的方式由视觉软件发送模拟信号控制改为电控发送电流信号控制，由于电流的传输时间极短，可以认为相机采图和串口的发送是几乎同时的，从而实现串口数据和相机图片的同步。但是由于相机的传输延迟和采集延迟，以及串口发送数据的延迟，而这两个延迟不一致。所以即便电控同时发送，视觉也不能同时收到，需要进行软同步。软同步我们采用的方法是确定图片收到了并使用了再开始下一次采集（依旧是电控控制串口的频率，不同的是，不是每一次采集的命令都有效，这个采集命令是否有效由视觉来控制）。这样每一帧图像都有对应的数据，并且保证图片对齐了串口数据。从而结合硬触发实现同时发送和同时接收。

②多线程

多线程，是指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程，进而提升整体处理性能。我们将原来单线程运行的任务改为多线程，使多个任务在同一时刻运行，在数据延迟不变的情况下提高了程序运行的帧数，将帧率从原来的 60 帧提高到 120 帧，提高了设备 cpu 和 gpu 的利用率。

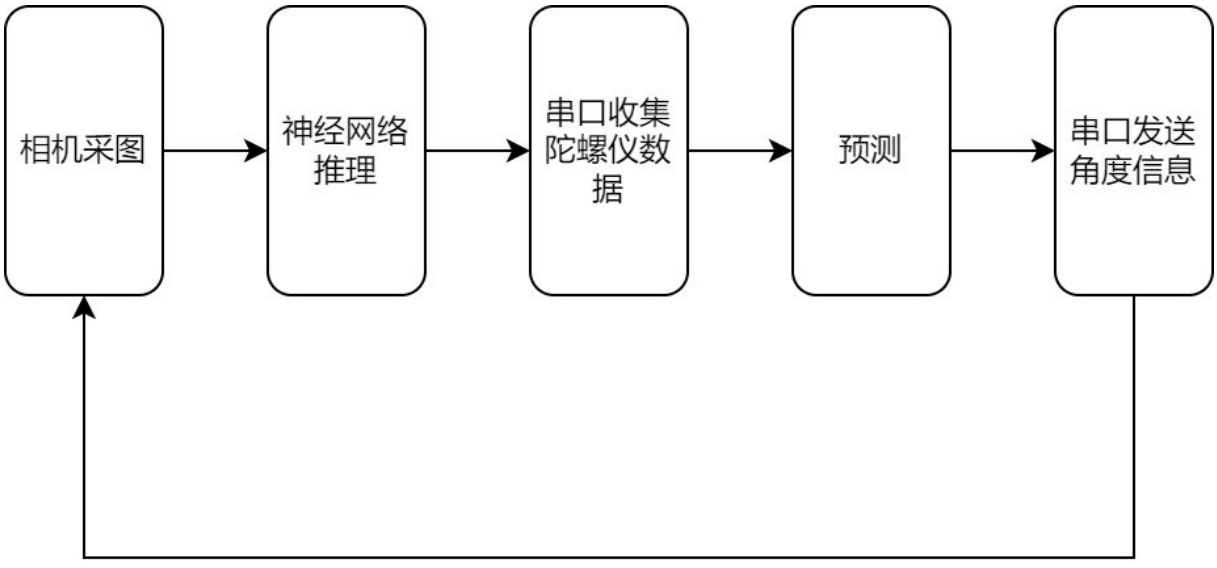


图 1-5-4-7 修改前自瞄流程

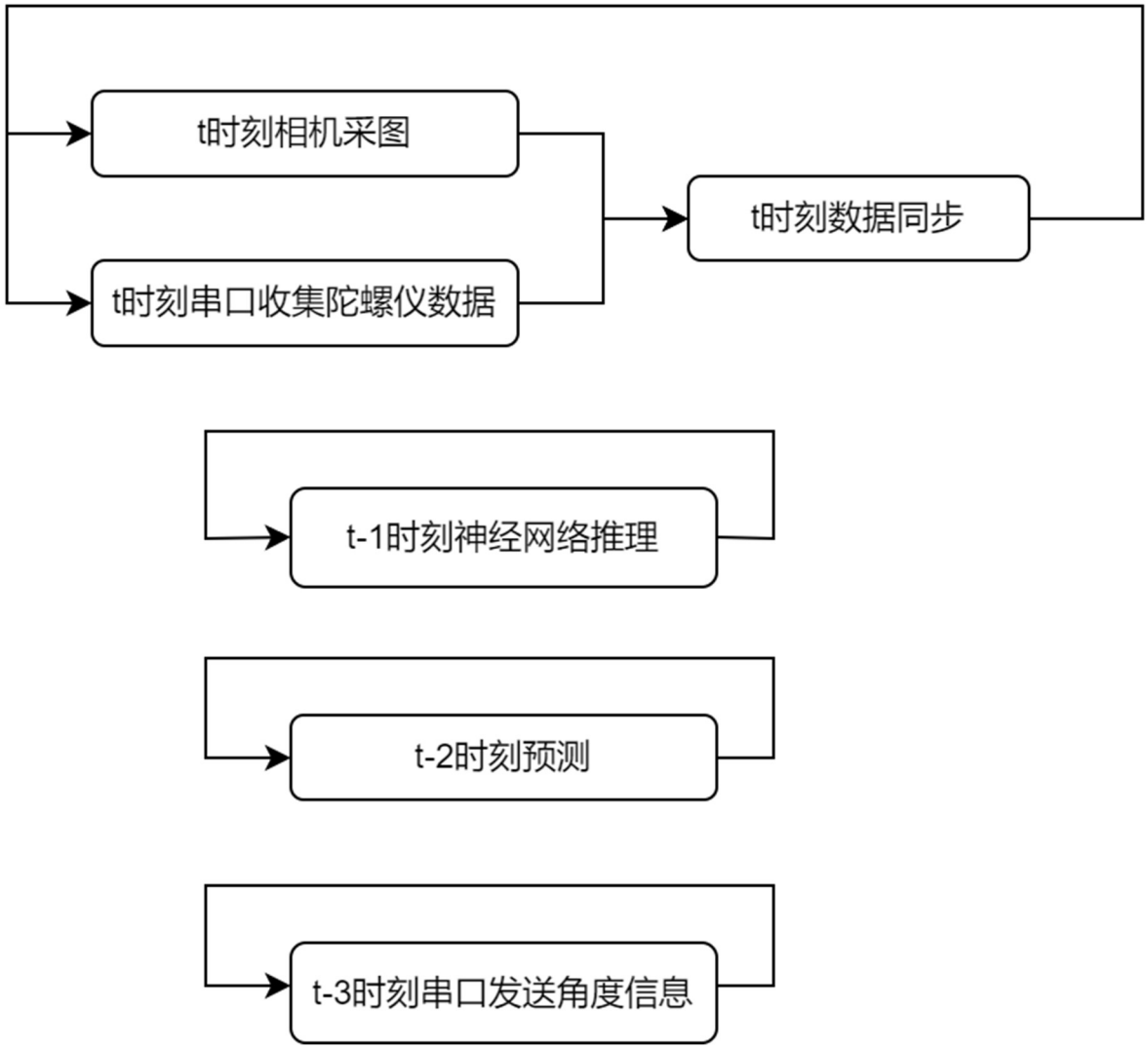


图 1-5-4-8 修改后自瞄流程

3、算法性能、优缺点分析、优化方案

算法性能：

平台	理论 FLOPS(FP32)
NVIDIA Jetson Xavier NX	844.8 G
cpu	自瞄帧率
6 核 1.4Ghz	100~130

表 1-5-4-9 算法性能测试表

优点：

在不改变运动模型的情况下提高了算法的性能

缺点：

1) 多线程由于锁的问题容易出现帧率下降的情况
2) 对快速移动和旋转的陀螺命中率较低

优化方案：

1) 优化多线程的代码，尝试无锁多线程
2) 优化预测模型，同时对敌方的四个装甲板进行预测

4、算法结果

以下为左右快速移动云台的情况下使用硬触发数据的前后对比:

图 1-5-4-10 使用硬触发前

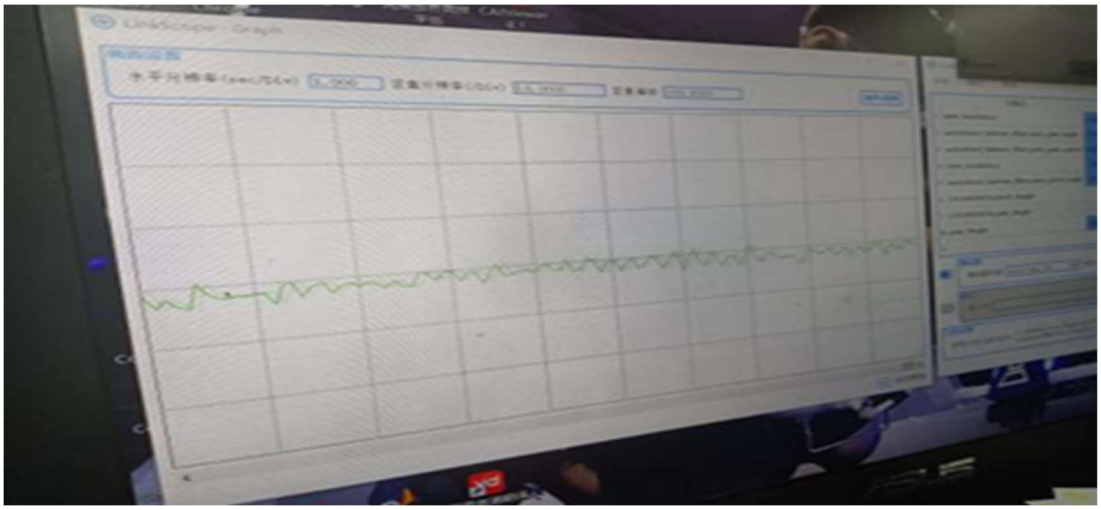


图 1-5-4-11 使用硬触发后

测试目标情况/命中率		优化前	优化后
1m	慢速移动	90	90
	快速移动	60	80

2m	慢速移动	80	80
	快速移动	50	75
3m	慢速移动	70	75
	快速移动	45	70
4m	慢速移动	60	70
	快速移动	40	65
5m	慢速移动	60	60
	快速移动	30	55
6m	慢速移动	50	50
	快速移动	20	40

表 1-5-4-12 自瞄命中率测试表

三、能量机关激活算法

1、功能介绍

在上赛季队伍能够激活能量机关的背景之下，本赛季队员面对能量机关较大改动的挑战，积极投入新算法的研究当中。

鉴于上赛季传统视觉使用的成功经验积累，队员延续着这一方向的研究。本赛季的工作主要针对能量机关的特征分析与识别算法的研发，对大能量机关转速的拟合速度优化和准确度提高，还有云台控制方法的更迭。

2、重要算法原理阐述、公式推导

① 识别算法

在分区赛后，根据相机内录分析，修改了图像预处理方法。如上图所示，可以发现能量机关的外部轮廓与击打的圆盘的色相、饱和度、亮度都有较大区别，非常契合在 HSV 颜色空间中的颜色分离的方法，于是我们选择先将击打圆盘剔除，通过对外轮廓的处理先找到应该击打的扇叶，再通过圆盘上的标靶找到应击打的圆心。

将外轮廓分离还有一个好处，能在外轮廓上更靠准地找到特征点，便于后续 pnp 测距的使用。

②预测算法

扩展卡尔曼滤波在 RoboMaster 的比赛中已经得到了较为广泛应用和检验，我们学习借鉴此算法并成功应用实践，在此不再赘述，仅列出推导的状态转移方程如下：

$$
\begin{aligned}
a_{k+1} &= a_k \\
w_{k+1} &= w_k \\
spd_{k+1} &= a_k * sin(w_k * t_k) + (2.090 - a_k) \\
t_{k+1} &= t_k + \Delta t \\
re_{k+1} &= re_k * cos(spd_k * \Delta t) - im_k * sin(spd_k * \Delta t) \\
im_{k+1} &= re_k * sin(spd_k * \Delta t) - im_k * cos(spd_k * \Delta t)
\end{aligned}
$$

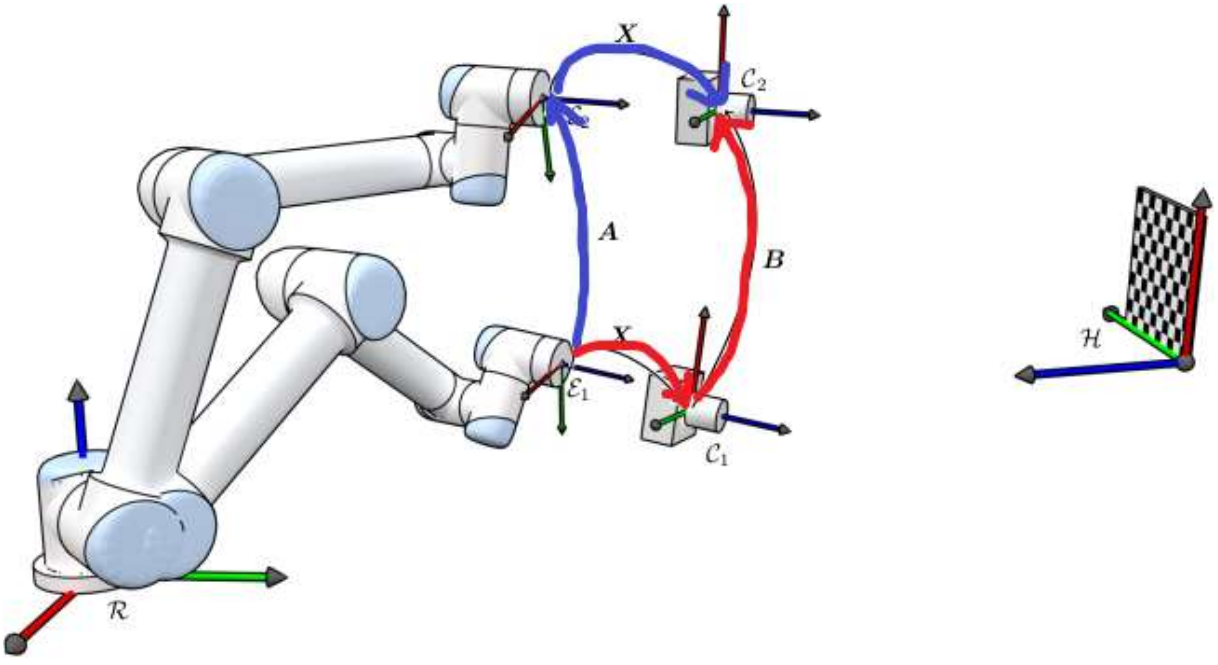
本赛季还使用了多线程预测的算法，单独开一个线程用来接收拟合结果，在此基础上进行预测的计算，将此线程帧率控制在 300 帧，向云台发送 300 帧的指令，实现更加精准的控制。

③云台控制

本赛季的规则对能量机关击打的精度提出了更高的要求，为了实现精准打击，我们设想能直接掌握能量机关在三维世界坐标系下的位姿和运动状态。

我们知道能量机关在一个平面上转动，我们将待击打的扇叶进行 pnp 计算， 即可得到 扇叶平面的位姿和待击打点的位置，通过拟合就可以得到运动状态 ，预测得到需要打击的点，再通过弹道解算得到云台的 pitch 和 yaw 轴偏角，再预测 0.001 秒后的 pitch 和 yaw 轴偏角计算加速度，对云台实现更精准的控制。

需要得到准确的世界坐标，需要更精确的标定，因此进行了陀螺仪与相机的联合标定，如下图所示。



我们把云台当成机械臂使用手眼标定，相当于“眼在手上”，相机和陀螺仪部分当作手的末端，通过使用标定板反算出来的位姿和陀螺仪的位姿进行计算，得到坐标转换计算需要的选择矩阵和平移矩阵。

弹道计算我们使用四阶龙格库塔来迭代，具体参数都是实验所得，该方法只是引用，不再赘述。

3、算法性能、优缺点分析、优化方案

算法性能：

在 NVIDIA Jetson Xavier NX 上 识别算法稳定在 140 帧左右，预测控制帧率限制在 300 帧。

优点：

相对神经网络识别帧率较高；

控制频率高，云台响应快；

缺点：

传统视觉需要现场调参，很不稳定；

多线程之间通讯数据多，代码冗余繁杂，且不稳定；

优化方案：

识别的鲁棒性，需要加强对神经网络的研究，开发神经网络的识别；

更改代码框架，使用 ROS 开发系统；

4、算法结果

在复活赛的适应性训练已经能几乎每次都完成击打大符和小符，但可惜车的稳定性太差，在热身赛的时候车烧冒烟了，缺少场地调试的机会，没能在赛场完成击打。

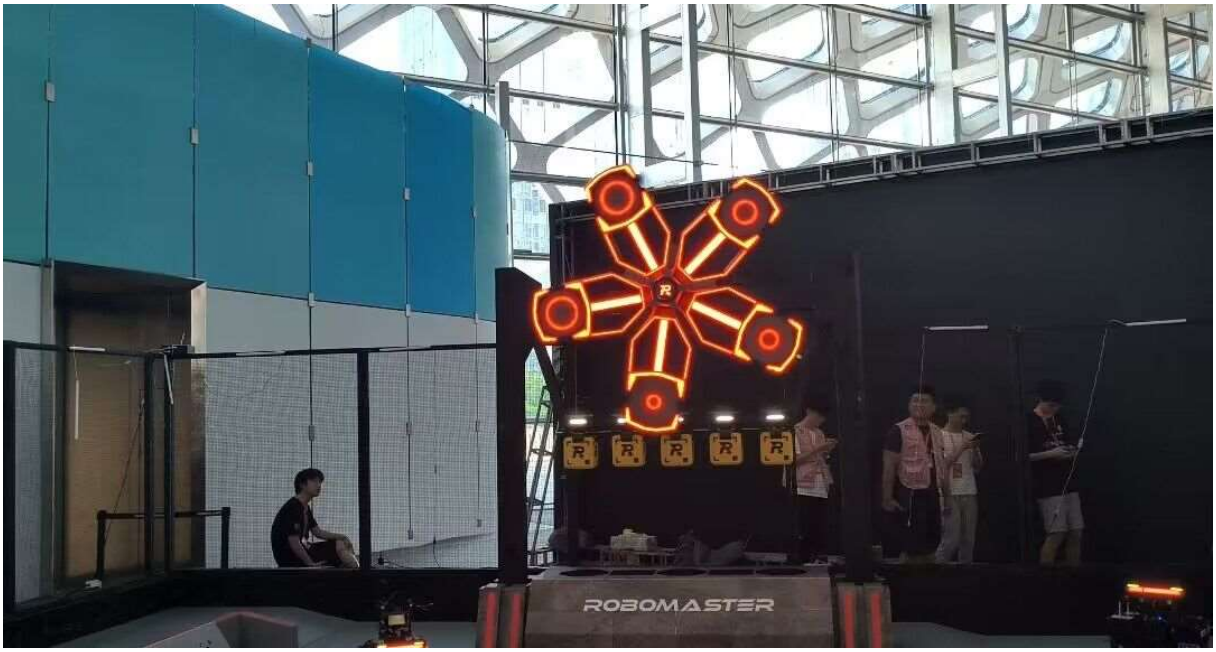


图 1-5-4-13 场地道具训练效果

1.6 研发迭代过程

1.6.1 测试记录

机械*关于弹丸发射速度波动对弹丸散步的影响

步兵机器人作为比赛中唯一可以击打能量机关的机器人，其弹丸散步数据对视觉识别自动击打有较大影响，同时，结合今年能量机关的改动，对于步兵的散步要求就提高了一个档次。经过查阅相关开源资料与同其他学校交流后，我们确定影响弹丸散步的因素包括：机械限位、弹丸初速度，枪管安装方式。结合我们机器人在上赛季的表现与机器人本身的结构限制，我们决定从弹丸发射速度测试对散步的影响

测试内容与测试目标：

本次测试主要是确定弹速波动对于弹丸散步的影响。通过控制机器人的弹速波动范围来确定弹丸散步分布范围。

测试内容为标准舵轮步兵机器人在 30m/s 的弹速限制下发射 17mm 标准荧光弹丸 200 发，击中约 7.5m 外的散步测试模块（约 A4 纸大小），重复 3 次。

根据实际测试，散步测试模块在连续命中约 100 发后会出现损毁现象，认为若测试模块某一区域出现损毁现象，说明该区域被弹丸集中击打，可以确定弹丸的有效击打，

在实际测试中，由于弹速波动有一定幅度，同时左右摩擦轮转速有微小速度差，所有弹丸在

测试数据：

第一次测试为 2023 年 4 月中旬，测试用的步兵机器人设定弹速 27m/s，根据现场连续测试结果得出弹速波动范围

次数/波动范围	最大值	最小值
1	29	26.3
2	29.2	24.5
3	28.8	25.2

得出测试结果综合如图

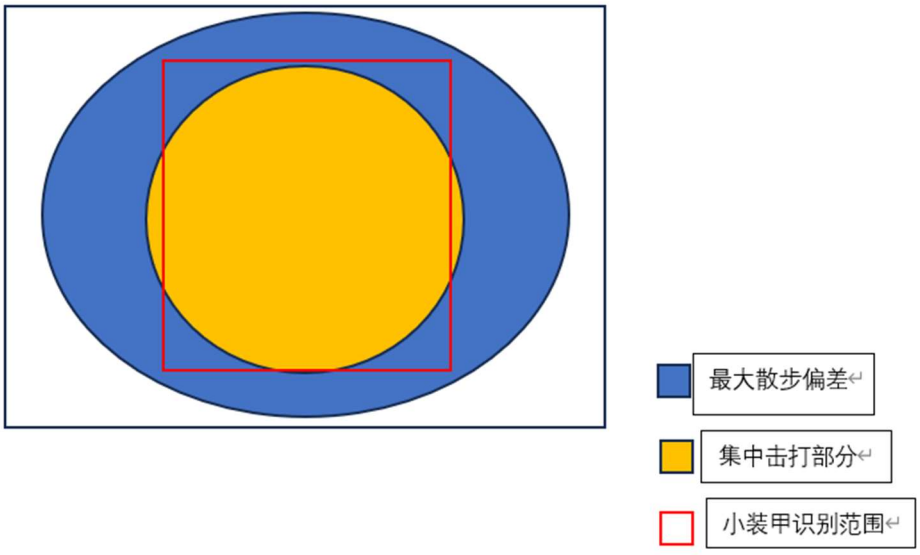


图 1-6-1-1 第一次测试效果

第二次测试为 2023 年 5 月下旬，测试用的步兵机器人设定弹速 27m/s，根据现场连续测试结果得出弹速波动范围

次数/波动范围	最大值	最小值
1	27.6	26.5
2	28.3	26.3.
3	29	25.9

得出测试结果综合如图

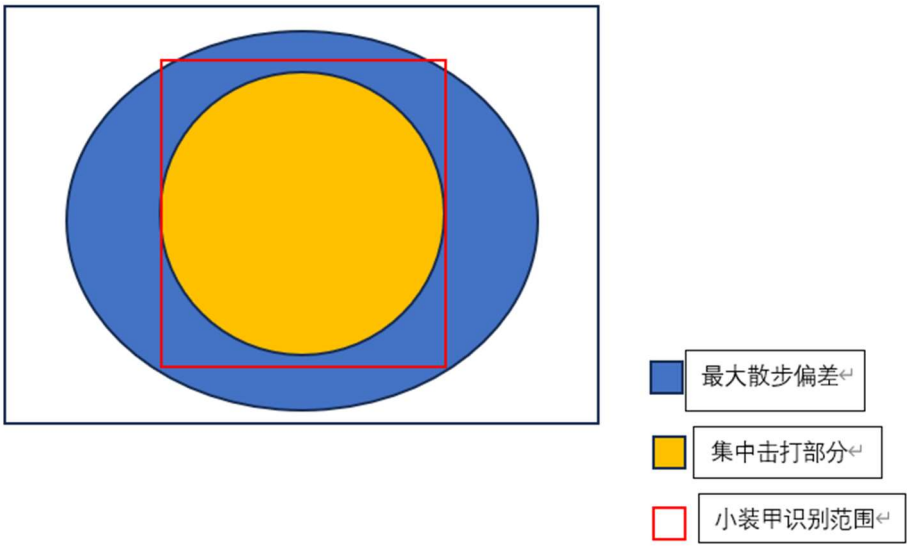


图 1-6-1-2 第二次测试效果

通过与第一次实验作对照，可以看出在限定发射初速后，弹丸最大散步偏差与集中击打部分都有明显缩小，初步可以证明弹速限制与机器人散步之间有关系

第三次测试用的步兵机器人设定弹速 27m/s，根据现场连续测试结果得出弹速波动范围

次数/波动范围	最大值	最小值
1	28.5	26.3
2	29.2	25.9
3	28.3	26.1

得出测试结果综合如图

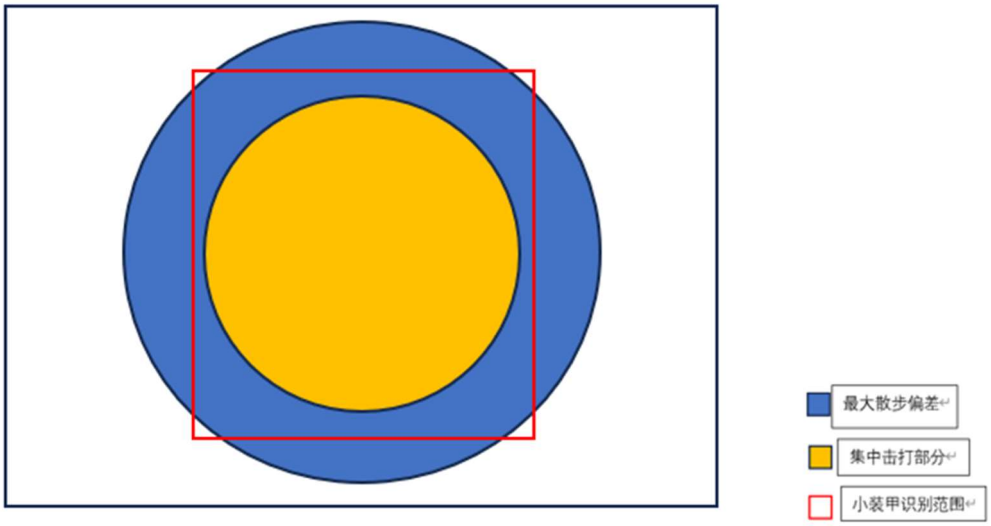
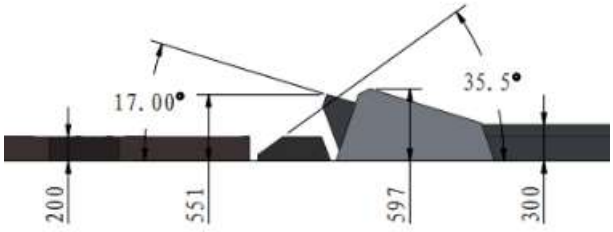


图 1-6-1-3 第三次测试效果

可以看出，随着弹速限定的精确，集中击打部分有效缩小至小装甲范围内，可以认为在视觉算法正常工作的情况下，步兵机器人可以正常激活大能量机关。

机械*悬挂弹簧的选择与·飞坡的关系

步兵机器人作为场上最常飞坡的兵种，飞坡后如何保持姿态稳定是个很关键的问题。就本赛季步兵机器人的飞坡数据来看，机器人在飞坡触地时约在公路缓冲区 1/3 处触地.



从比赛地图中可以的到步兵机器人的落地高度约为$h = 0.35m$

通过实际测验，可以得出从车辆离开 17° 坡到触地时间$t_1 = 0.3s$

可以得出落地速度$v = gt_1 = 2.94m/s$(g 取值 9.8)

步兵机器人自重$m = 20kg$

可得步兵机器人落地时动量$p = mv = 58.8kg * m/s$

步兵机器人采用弹簧为溪地标准全金属弹簧，技术参数如下

全金属避震器	85	5mm 通孔+ 3mm 球头孔	18	3	20	15	1	30g	KJ85T53	79
							4			89
	103				35		1	35g	KJ100T53	79
							4			89

可得其弹性系数$k = 1N/mm$

当车辆飞坡触地时，设其为理想状态下的四轮同时触地。根据实际测试，飞坡后弹簧将瞬间压缩至最大压缩量，设$x = 20mm$

由胡克定律，可得单根弹簧提供的弹力为$F = kx = 20N$

步兵机器人设计上共有 8 根相同型号的弹簧为其提供缓冲,设弹簧作用时间为一瞬间，可得$t_2 = 0.36s$

则总冲量为$I = 8\int F\,dt = 8Ft = 57.6kg * m/s$

由计算结果可知$p > I$，则在每次飞坡后，都会有部分多余的动量转移到避震器支柱上吸收，长久使用后容易导致避震器损坏

在后续方案中，我们重新设计了弹簧，在原基础是将弹簧的线径提升至原线径的 1.5 倍

由弹簧弹力系数计算公式$k = \frac{Gd^4}{8nD^3}$，当$d_{、} = 1.5d$时，$k_{、} = 5.06k$

很明显，新弹簧的最大冲量将大于原弹簧，因此，在新版的步兵中，悬挂将有效保护飞坡后步兵的机械结构

视觉*自瞄

刚开始整个代码运行的时候帧率特别低，然后更换了神经网络把帧率提上去了，云台跟随装甲板变得丝滑很多。但是上车测试打弹的时候依旧命中率极低，怀疑是弹道解算有问题，因此更换了弹道解算算法，采用龙格库塔迭代法计算 pitch 补偿，坐标系转换部分也进行了很久的计算检验，更换后命中率立马提升了，效果显著。然后不断的细调参数，如发弹延迟，根据弹丸落点调，比如装甲板往某个方向移动，弹丸一直在装甲板移动方向的前方，就知道是预测时间给多了，预测时间可以调小一点。还有卡尔曼滤波的一系列参数也需要细调，如观测噪声和测量噪声。还有串口也很重要，我们在串口部分耗费了大量时间，有时候突然出现数据抖动，云台疯转，或者甩头，大概率都是串口的问题，如串口堵塞，数据延迟大，检测不到串口。数据不对齐也会导致云台抖动，所以后面给代码加入了硬触发和软同步，以实现图片数据和串口数据的同时发送和同时接收，然后通过左右快速摆动云台测试，发现抖动减少了好几倍。然后就是大的逻辑问题，如误识别，这个主要运用于哨兵自瞄，所以利用了裁判系统的机器人血量来进行辅助判断，若机器人血量为 0，则不击打，加入后，误识别概率大大降低。还有击打优先级，优先击打英雄然后是步兵，经过测试，当步兵和英雄同时出现时优先击打英雄，通过实战也发现该功能没问题。至于工程，我们加入的功能是是否击打工程交给云台手来判断，若云台手决定要击打工程，则识别工程，否则不识别，经过测试，该功能也没问题。还有曝光增益等参数需要根据实际情况调，在实验室适用的参数在比赛场地不一定适用，因为灯光条件不一样，若调的不对，会直接导致无法识别装甲板，自瞄失效。

文本来源：周结

1.6.2 版本迭代过程记录

版本号或阶段	功能或性能详细说明	完成时间
V1.0	视觉*将网络从 yolov5 改为 yolox 机械*实现飞坡的稳定化和耐久性 电控*步兵优化功率限制	2023.1.5
V1.1	视觉*将自瞄从单线程修改为多线程 电控*步兵自瞄完成插帧	2023.3.12
V1.2	视觉*修改自瞄优先级逻辑 电控*步兵完成底盘缓启动	2023.4.22

版本号或阶段	功能或性能详细说明	完成时间
V1.3	视觉*添加完善弹道解算 机械*实现 7m 射击散步控制在小装甲范围内 电控*步兵完成热量限制优化 电控*优化超级电容控制逻辑	2023.5.6
V1.4	视觉*添加硬触发 电控*实现车辆稳定的单发功能	2023.7.18

1.6.3 重点问题解决记录

序号	问题描述	问题产生原因	问题解决方案&实际解决效果	机器人版本号或阶段	解决人员
1	**步兵起步时或加速时有额外的功率消耗**	**加速度过大导致车辆打滑**	**通过缓起步实现较低的加速度从而减少打滑的功率消耗**	**V1.2**	嵌入式软件工程师：赵栋林
2	**步兵开启摩擦轮后云台会上下飘**	**开启摩擦轮后陀螺仪受到影响**	**通过更换摩擦轮来减小摩擦轮的振动对陀螺仪的影响**	**V1.1**	机械工程师: 徐羽成，张博洋 硬件工程师：邢可 嵌入式软件工程师：赵栋林，陶承誉
3	**步兵打大幅时需要的单发模式不稳定**	**拨盘电机控制精度不够**	**通过给电机闭双环与细调 PID 参数实现单发模式的稳定**	**V1.3**	嵌入式软件工程师：陶承誉
4	**自瞄发送串口信息和接受陀螺仪数据的延迟波动较大**	**串口硬件不稳定**	**更换串口硬件，延迟变小且较为稳定**	**V1.2**	硬件工程师：邢可 嵌入式软件工程师：赵栋林，陶承誉 算法工程师：叶裕杰，高源
5	**对快速移动问题命中率较差**	**用于预测的滤波收敛较慢**	**修改协方差矩阵，对移动目标的命中率提高**	**V1.3**	算法工程师：程天乐，高源

1.7 团队成员贡献

姓名	基本信息 （专业、年级、队内角色）	主要负责工作内容描述	贡献度 （所有成员贡献度合计为 100%）
徐羽成	机械本研一体班、大二、步兵组机械	负责整个机器人的机械设计与组装相关工作	20%
张博洋	测控技术与仪器、大一、步兵组机械	负责整个机器人的机械组装与维护相关工作	10%
邢可	电子信息工程、大二、硬件开发负责人	负责整个机器人的主控设计与焊接以及超级电容的控制板开发等	10%
赵栋林	电气工程及其自动化、大二、软件开发负责人	负责整个机器人的嵌入式开发，包括底盘控制、云台控制开发等	20%
陶承誉	智能感知工程、大一、步兵组电控	负责整个机器人的嵌入式开发，包括云台控制、视觉系统的嵌入式环境开发等	10%
高源	测控技术与仪器、大二、算法开发负责人	负责整个机器人的视觉框架设计和神经网络部署等	10%
叶裕杰	地理信息科学、大一、步兵组视觉	负责整个机器人的能量机关部署和框架等	20%

1.8 参考文献

https://github.com/tup-robomaster/TUP-InfantryVision-2022

https://github.com/Harry-hhj/CVRM2021-sjtu

1.9 技术方案复盘

1.9.1 赛场性能表现情况分析

（1）缺点：

(1)云台未配平，yaw 轴电机使用时间长发烫，影响云台响应。

(2)底盘悬挂受力不均，容易断。

（2）优点：

（1）舵轮底盘增强了移动速度，转小陀螺速度加快。

（2）重心较低，飞坡稳定。

（3）云台 yaw 轴空间扩大，利于走线。

（4）全向运动正常未出现超功率和突发超热量现象功率限制及热量限制达到了其效果飞坡落地平稳。

1.9.2 赛场性能表现与规划对比分析

本赛季对步兵发射机构进行了优化，解决了发弹不连续的问题，优化了 yaw 轴云台走线，目前仍存在 yaw 轴走线空间还是不够的问题，pitch 轴电机过热影响 pitch 轴响应的问题，和底盘悬挂结构不合理，使避震器易损坏的问题。

1.9.3 经验总结

经验总结--步兵预算

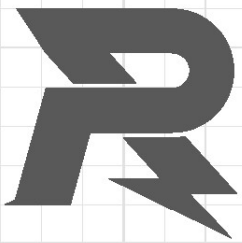
新车每辆 2 万元（机械部分预算 1 万，电机及硬件部分 1 万），旧车翻新每辆 2500 元；预计本赛季制作新车 2 辆，翻新旧车 2 辆，总共步兵预算 4 万 5 千元。

步兵实际情况总结

依据本赛季情况总结，我队共制作新车 3 辆，翻新旧车 1 辆。每辆新车制作共花费 22240 元，旧车翻新 2 千元（只涉及机械翻新），总花费 68720 元。

主要预算偏差为电控部分，考虑到实际情况下电机数量、价格以及硬件损耗问题，再次制作步兵预算方案时应给予电控部分 1.2 万元资金预算。

根据总体情况而言，前期预算制定较为合理，同时考虑到战损情况，我队为每辆步兵提前预留了 1 千元备用资金，作为准备备用耗材使用。



邮箱：robomaster@dji.com
论坛：http://bbs.robomaster.com
官网：http://www.robomaster.com
电话：0755-36383255（周一至周五10:30-19:30）
地址：广东省深圳市南山区西丽街道仙茶路与兴科路交叉口大疆天空之城T2 22F