# 技术报告


**夏令营-高中五组**

组员：蔡毓鑫、曹瑞翔、陈辰、陈思达、王嘉平、张嘉轩、张帅鹏

# 目录

## 机器人设计需求

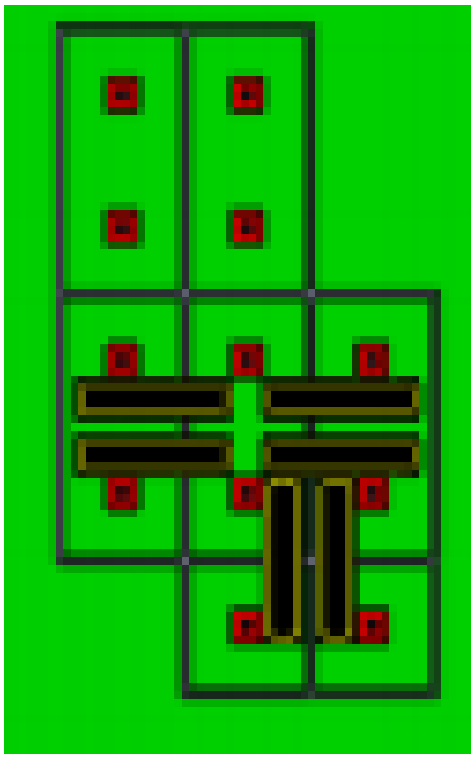
机器人的设计肯定是要贴合规则来进行设计，所以我们一上来先详读了发到每个人手里的规则介绍。读完之后我们发现这次 RM 夏令营的规则相对于 RM 比赛放宽了许多，我们可以不用考虑功率限制，装甲板的安装位置也可以在很大的程度上改动，另外还可以额外加一块电池。对于这些规则上的放宽，我们明确了我们应该展开想象力的方针，因为这样我们才可以制作出更高效，更富有创新性的结构。

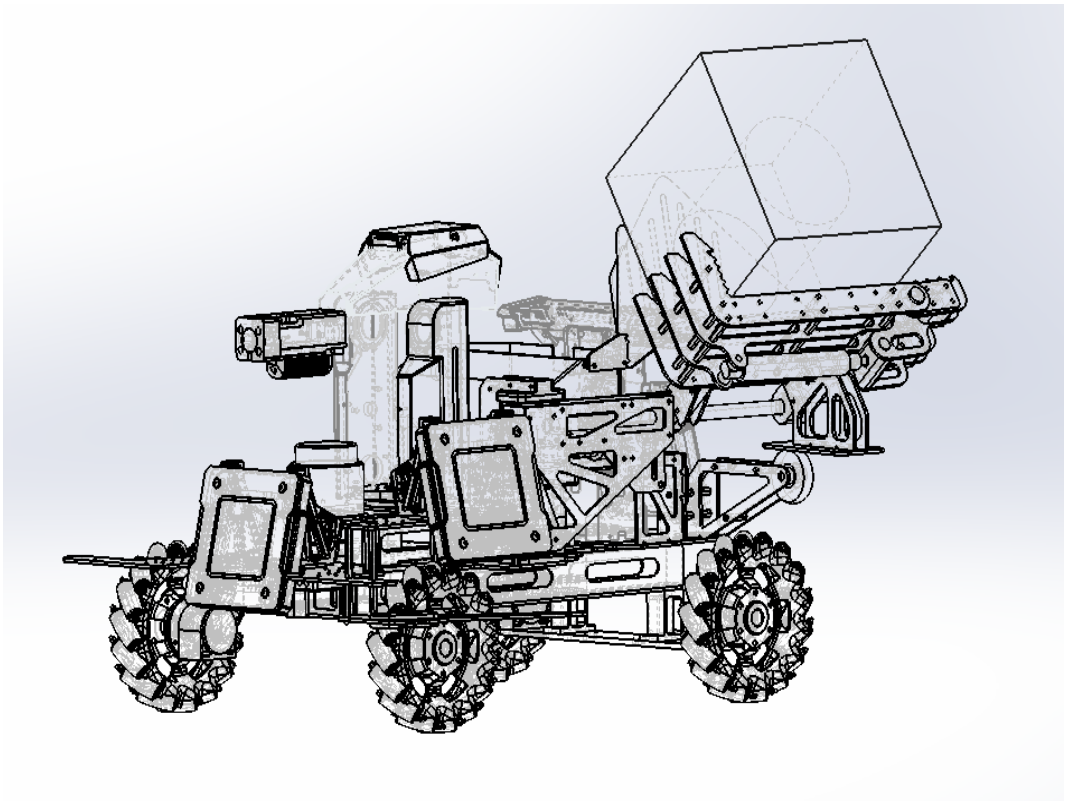
这次的比赛目的主要是拾起在障碍块里的子弹，然后利用这些子弹和敌方对打，那么这就将比赛划分为了两个部分，需要分开讨论。针对第一阶段，我们注意到了地图图纸里，位于障碍块下面的塑料托盘，分割场地的线槽减速带，和贴在地上的白色胶带。针对这些场地上的改动，我们先考虑需要改变哪些方面，又有哪些地方不用动。我们首先测试了线槽减速带，发现车底盘的高度足够垂直通过减速带，但是如果与减速带平行的话则会卡底盘。通过这次测试，我们确信底盘不需要改动，但是也需要注意在操作中出错。在图纸上，白色胶带位于两个障碍块之间，于是我们就想到了使用寻迹卡来保证车体对正两侧的障碍块。同时，由于两个障碍块之间的距离是 600 毫米，在设计的时候我们考虑到车身的宽度不能超过 600 毫米。时间方面，组委会设计了时间分，也就要求我们的收子弹机构要高效，就算是在对打阶段，高效的收子弹也可以为我们赢得先手优势。

第二阶段的对打，相对的要求就比较少，但是是建立在子弹够多的前提下的。由于只是 2v2，在没有无人机视角的帮助下，双方都无法知道对面的位置，只能凭操作手随机应变。于是在第二阶段的比赛前，我们和盟友开了一次联合会议，讨论了我们的走位策略和进攻路线。最终确定了以盟友为进攻主力，我方从旁策应的战略方针。充分利用了盟友射手的准确性，和我们机器人庞大的体积优势。

# 总体方案

## 1) 设计





讨论方案一开始，我们就谈到了对准的问题。RM 的步兵车的云台只能随着车而转动，所以视野只有正前方的一小片，那么想要对准位于机车两侧的障碍块就是重中之重的问题。经过多个方案的讨论，最终我们确定了自动巡线和旋转摄像头的结合。采用前后两个七路灰度传感器来保证车身与障碍块的连线垂直，四个红外对管来确定两侧箱子何时到位，通过交叉定位从而确保了战车的位置是固定准确的。如果出现了差错，还可以旋转摄像头来手动对准。

前面说到效率是很重要的，所以我们毫不犹豫的选择了同时夹两侧的障碍块。但是由于没有考虑到展开超宽，最终只能两侧轮流打开，但是依然比只做一侧要高 效。为了防止两侧互不干涉，我们使用 SolidWorks 3D 建模，基本上保证了没有大块的冲突，但是由于一些小部件的不注意，最终使用角磨机对冲突的部分进行微调。强度，和重量在这里是一对反义词。本来许多需要使用 CNC 铝件加工的地方，后来由于经费和重量问题，而选取了一部分不受力的零件改用 3D 打印。最终的机器人大体是使用 3 毫米碳板组装而成，关键位置使用铝件，辅以 3D 塑料件。

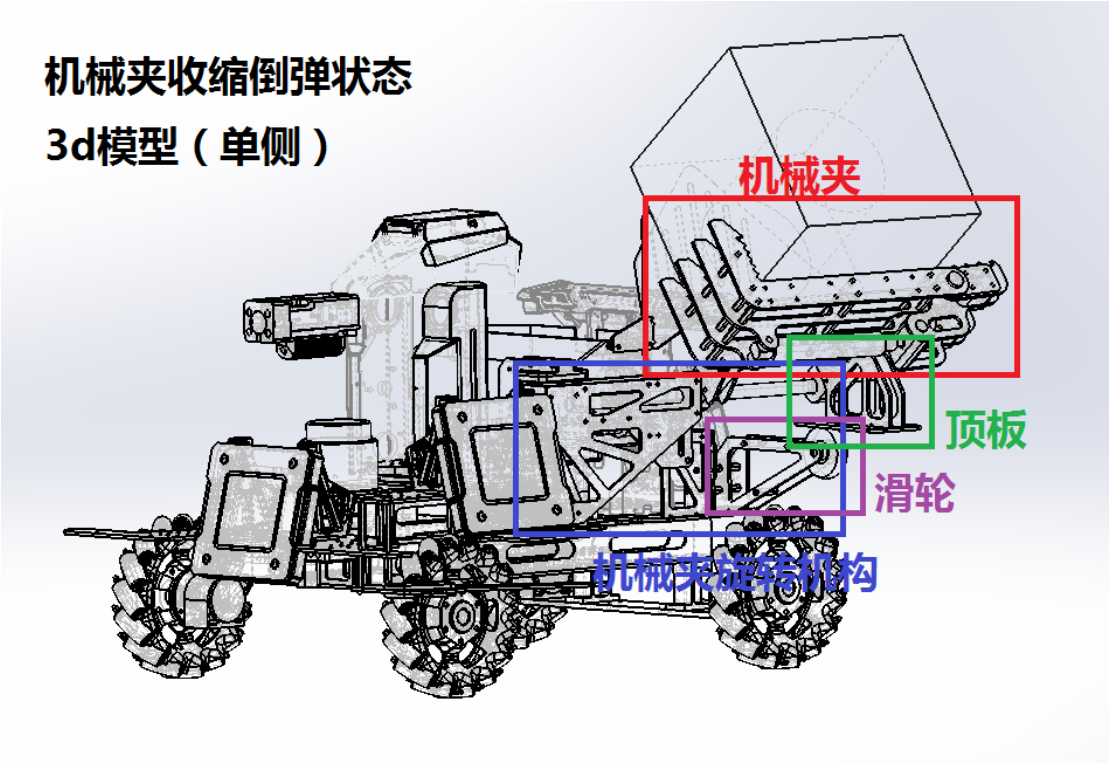
## 2) 动力

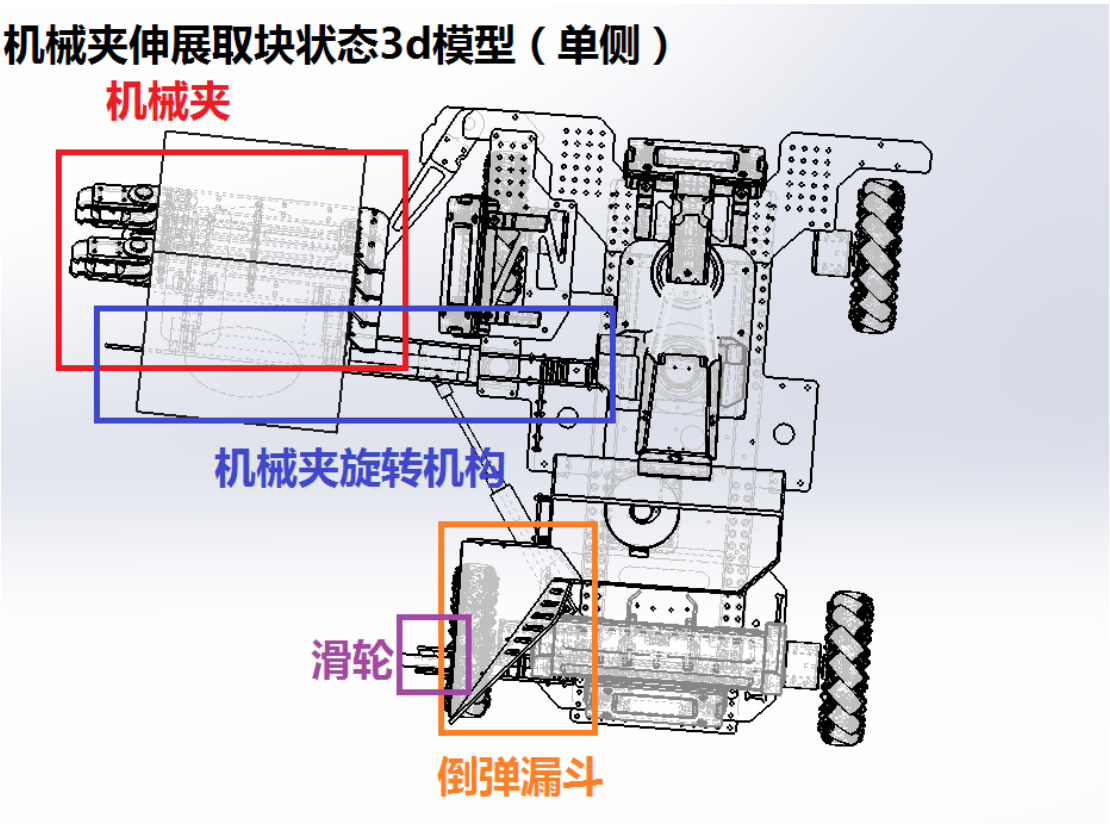
考虑到动力，我们毫不犹豫的选择了气动。因为气动行程短，效率高，特别适合在爪子上使用。而在机械臂的收缩上，我们依然选择了气动，事实证明这不是一次明智的选择。在我们明确方案之前，我们的导师建议我们对障碍块的抬升机构做一个计算。由于我们错误的计算了机械臂的力矩问题，导致我们误以为使用 16 毫米的气缸可以完成障碍块的抬升。最后，我们不得不改而使用 1:144 的 减速电机，搭配绞盘来把抬升机构拉上去。气缸则被我们充上气，当做弹簧来使用。

## 3) 传感器

巡线的传感器是比较容易选择的，因为规则里面明确有白色的线，并且在后面的规则讨论中，我们确认会在白线旁贴黑线，于是就选择了七路灰度传感器。红外对管我们选用了比较便宜的可调距离式，结果在最后发现红外对管在面对黑色的障碍块时无法正常工作的问题，最终交涉后确认使用贴白色胶布的障碍块，解决了问题。除此之外我们还用到了四个微动开关，用于确认机械臂何时放到应该的位置。而且有了这四个微动开关，我们的机车就达到了半自动，大大的提升了效率。

# 整体布局


机械夹收缩倒弹状态
3d模型（单侧）
机械夹
顶板
滑轮
机械夹旋转机构


机械夹伸展取块状态3d模型（单侧）
机械夹
机械夹旋转机构
滑轮
倒弹漏斗

机械夹旋转机构展开、收回机构的改进过程

1、初始设计

直接控制气缸，用压缩空气把整个机械臂旋转机构弹出、收回

发现问题：气缸的力过小，不足以在收回时将整个夹着障碍块的机械爪收回至合适位置

2、改进设计

在旋转机构末端安装舵机，单独控制抬升夹着障碍块的机械夹

发现问题：由于障碍块重量过大以及舵机位置安装限制的原因，舵机无法抬升

3、最终设计

在车辆上安装绞盘，使用减速电机带动绞盘，拉动机械夹旋转机构使其收回并倾倒子弹

将气缸一端连接至储压瓶使气缸一端始终保持一定压力，让气缸始终有一定推力用于在绞盘放线时推动旋转机构展开

**PART 2** 涵道取球

机器人设计需求

任务是取出盒子中的子弹并进行射击，将管子通过机械臂伸入盒子中吸出里面的子弹，期望每个盒子的吸球能在 10 秒内完成，因为设计结构未超出整车太多，所以具体路线要与盟友共同商榷。因为该方案的取球速度极快，效率相比机械臂较高，所以能在相同时间内取得更多子弹以提高战斗持续性。取消原装弹舱，将涵道密封舱与弹舱做整合。

总体方案

用亚克力做舱体，热熔胶枪做密封。通过涵道抽气，在一个密封舱内形成负压，然后通过密封舱内外气压差在与外界接通的吸球管中形成强大气流，产生吸力，使球吸入弹舱中，密封舱与弹舱设计为整体。机械臂有两个自由度，可以在同一位置对两边的盒子内的子弹进行吸取，节省了换向时间，提高效率。在对位上使用红外对管，共有三个，当前后两个红外对管未检测到盒子，中间的红外对管能检测到盒子时停止，在车底安装七度灰度传感器，来按场地中的白线进行寻线，用以上两种传感器来实现自动实行吸取。机械臂的主要部分为 3mm 碳纤维板以保证强度，在转动处使用舵机。

涵道动力选择 70mm12 叶内转 6s 涵道，试验后测出推力达到 2kg 以上，舵机使用 RM 同一发放舵机，经测试可以轻松完成安装管后的机械臂的运作。

传感器使用红外对管和七度灰度传感器

三．方案测试改进过程

1.舱体设计与改进

1.0（SW 图）经团队讨论，考虑到涵道侧面装会对车的稳定性产生影响，涵道推力方向应向下，于是将涵道安装口从侧面改到上方，并对舱体的整体外观进行改动

2.0（SW 图）能吸起球，但吸取时间较长，效率还需提升，减小密封舱体积，使负压产生时间更快

3.0（SW 图）制作完成进行测试，无法将球吸起，经询问大学生，初步判断有可能是舱内某处形成涡流，还有管子入舱口未与密封舱口相对，两个原因导致吸力不够

4.0（SW 图）改进后，解决了之前的问题，但与云台有产生干涉，会影响到取球后战斗的阶段云台的机动性

5.0（SW 图）更改密封舱尺寸，降低与云台有干涉的部分，测试后吸力与之前几个版本的相比有很大提升

2.机械臂

每一个版本的舱体都有相应可用的机械臂，但实现功能相差不大，所以没有很大的改动。（SW 图）

3.吸球管的测试与选择

首先选择了三种管子，52mm 直径的铝箔管和软塑料管，大口径铝箔管，在测试后发现，大口径铝箔管管径太大，同一涵道装置产生的吸力太小，52mm 直径的铝箔管和软塑料管的吸力均能达到要求，在搭载机械臂经行测试后发现，铝箔管在吸球时管口会因为强气流而快速抖动，不便于吸球，最后选择使用软塑料管。

f 本方案只停留在测试阶段，在赛前几天电调因为一些意外情况损坏，没有时间购买新的并上车测试，所以未考虑 PWM 的调试和布线等具体问题

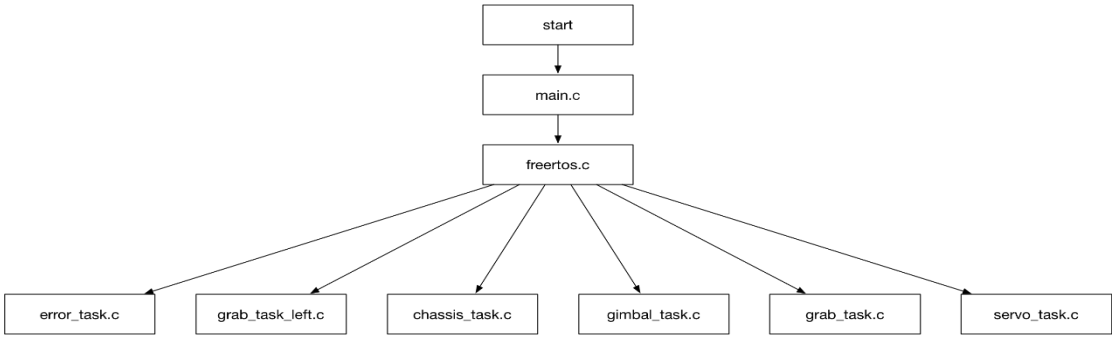
四．项目历程简介

测试时吸球最快速度能达到 6 秒左右，未能上车测试。在这个方案中在各队中首先创新使用涵道吸球，并且在这个方案中，学习到激光雕刻机的使用，气流流向的判断。今后在使用有关气流的器材上要学习一些有关空气动力学的知识才能更好地使用这些器材。接下来要向机械结构搭建与设计方面努力，之前也有学习 C++基础知识，接下来也将深入学习相关方面的知识，来进一步提升自己的能力。

技术难点：①如何保证舱体的密封性

②舱体内产生的气流流动的判断

# 程序

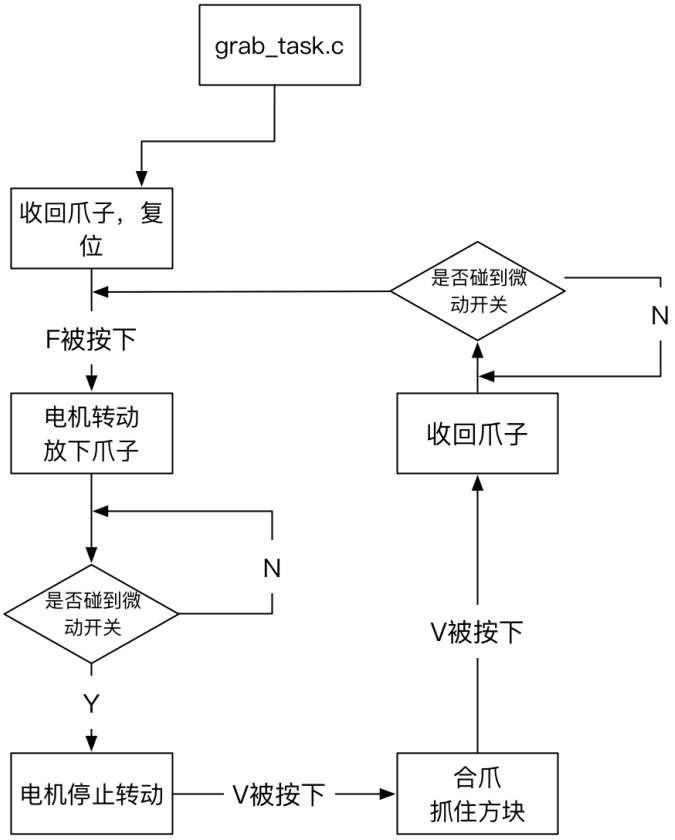



stm32 控制板接电复位后进入 main.c, 在 main.c 中进行完 io 口初始化后调用 freertos, 开始运行自定义的六个线程。

线程作用：

| | |
|---|---|
| error_task.c | 检测模块是否掉线 |
| grab_task_left.c & grab_task.c | 左右两边机械爪的抓取线程 |
| chassis_task.c | 底盘控制线程 |
| gimbal_task.c | 云台控制线程 |
| servo_task.c | 使用舵机控制摄像头转动 |

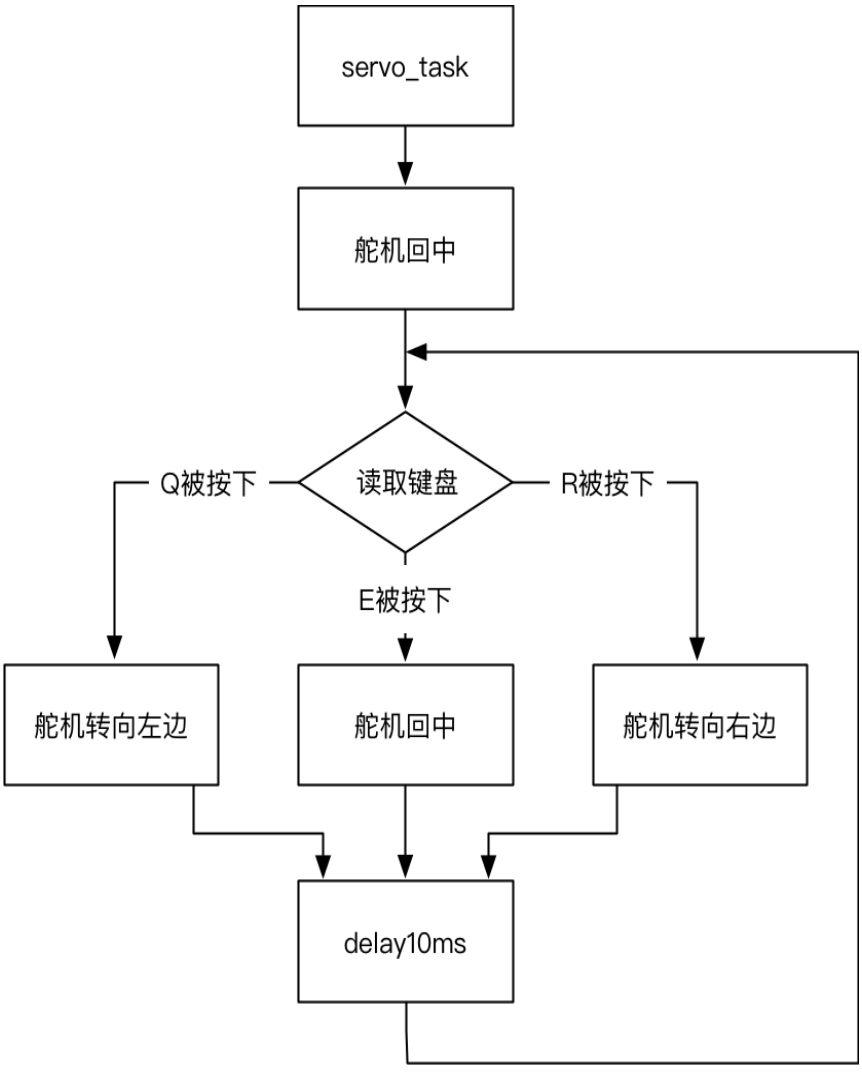
1.grab_task_left.c & grab_task.c



如果 F 被按下，爪子被绕线电机放下
然后用 switch-case 写出一个状态机，控制机械爪的状态（因为若使用 while()一直循环的话线程会死掉）
如果微动开关被按下状态机就跳到下一个状态
这样就实现了对机械爪的键盘控制
这个流程图只列出了一部分状态机的控制逻辑，完整的控制逻辑请阅读 grab_task.c。

2.servo_task.c

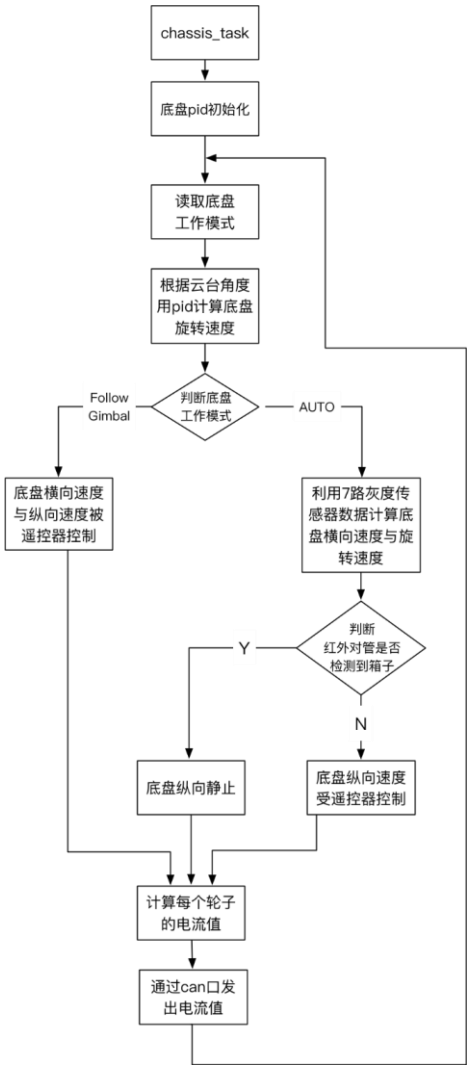
使用 switch-case 结构，判断目前键盘的状态和上一次的状态，实现按下 Q，E，R 键后舵机与云台上摄像头一起转动的功能。



```c
int kb_stat = 2;
int servo_pwm = 1500;
void servo_task(const void *argu)
{
    servo_pwm = 1500;
    while(1)
    {
        if (rc.kb.bit.Q)
            kb_stat = 1;
        if (rc.kb.bit.E)
            kb_stat = 2;
        if (rc.kb.bit.R)
            kb_stat = 3;

        switch(kb_stat)
        {
            case 1:
                servo_pwm = 920;
                break;
            case 2:
                servo_pwm = 1500;
                break;
            case 3:
                servo_pwm = 2080;
                break;
        }
        TIM2->CCR1 = servo_pwm;
        osDelay(10);
    }
}
```

3.chassis_task.c



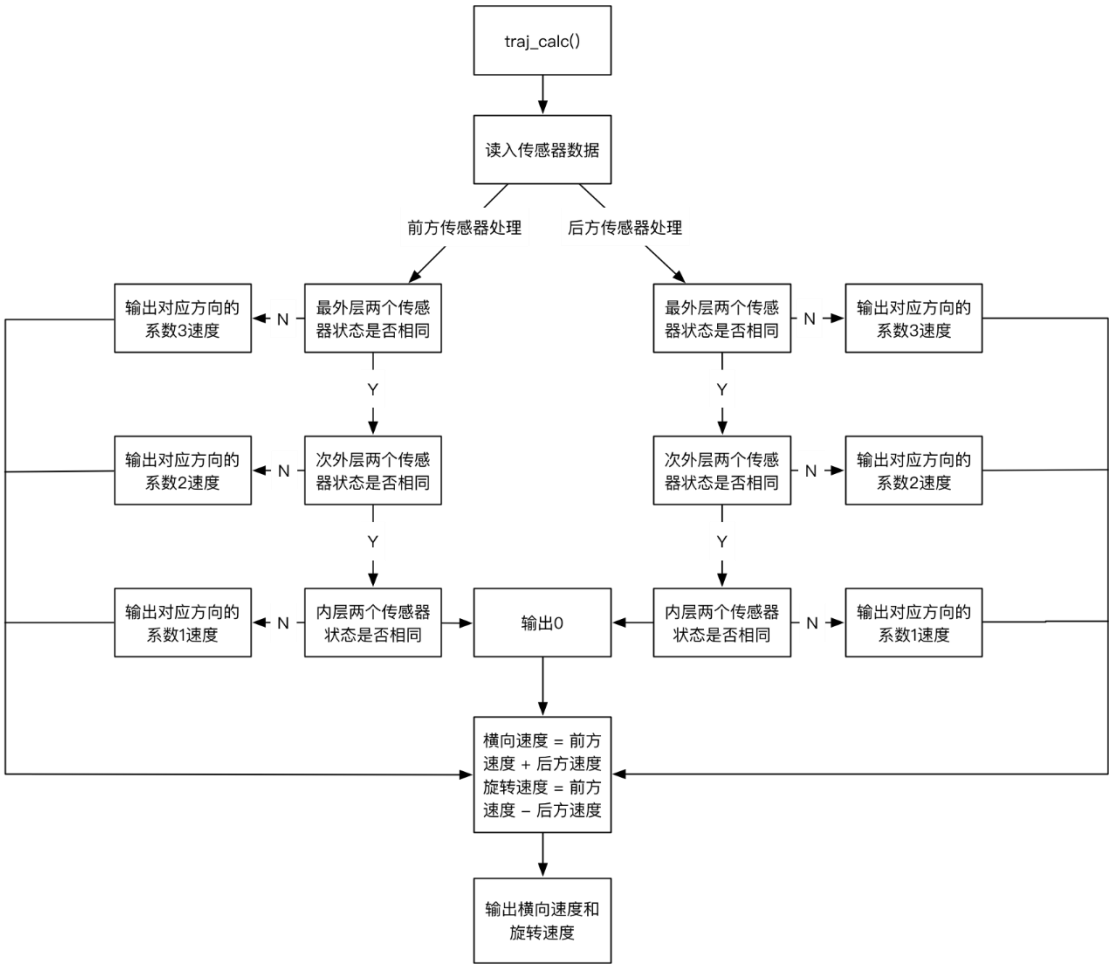
底盘线程在原有程序的基础上增加了一个自动定位模式，通过遥控器的 sw2 拨杆可以调出。

在自动定位（AUTO）模式中，车的前后运动速度是遥控器右手前后遥感的映射，左右和旋转使用前后的 7 路灰度传感器计算控制。当红外对管检测到箱子的时候车将静止，移动完全不受遥控器控制，此时就可以抓取方块。

实现代码如下
其中 traj_calc()为速度计算函数
具体见下

```
412             case CHASSIS_AUTO:
413             {
414                 chassis.vy = rc.ch2 * CHASSIS_RC_MOVE_RATIO_Y + km.vy * CHASSIS_PC_MOVE_RATIO_Y; // vy USE REMOTE CTRL
415                 get_sensor_data(sensor_data);       //GET SENSOR DATA
416                 traj_calc(sensor_data, &chassis);   //GET VX&VW
417                 if((sensor_data[14] == 1 && sensor_data[15] == 1 )||( sensor_data[16] == 1 && sensor_data[17] == 1))
418                     //if all IR detector at one side detected the box   (the car is at the right place)
419                     chassis.vy = 0; //STOP
420             }break;
421
422             default:
423             {
424                 chassis.vy = 0;
425                 chassis.vx = 0;
426             }break;
427         }
```

使用 7 路灰度传感器通过计算输出横向速度与纵向速度的逻辑如



```
void traj_calc(int sensor[], chassis_t *chassis_data)
{
    float f_v = 0;
    float b_v = 0;
    //front wheel vx calculate
    if ((sensor[0] + sensor[6]) != 0)
    {
        f_v = sensor[0] * COEF_3 - sensor[6] * COEF_3;
    }
    else if ((sensor[1] + sensor[5]) != 0 && f_v == 0)
    {
        f_v = sensor[1] * COEF_2 - sensor[5] * COEF_2;
    }
    else if ((sensor[2] + sensor[4]) != 0 && f_v == 0)
    {
        f_v = sensor[2] * COEF_1 - sensor[5] * COEF_1;
    }
    else
    {
        f_v = 0;
    }
    //rear wheel vx calculate
    if ((sensor[7] + sensor[13]) != 0)
    {
        b_v = sensor[13] * COEF_3 - sensor[7] * COEF_3;
    }
    else if ((sensor[8] + sensor[12]) != 0 && f_v == 0)
    {
        b_v = sensor[12] * COEF_2 - sensor[8] * COEF_2;
    }
    else if ((sensor[9] + sensor[11]) != 0 && f_v == 0)
    {
        b_v = sensor[11] * COEF_1 - sensor[9] * COEF_1;
    }
    else
    {
        b_v = 0;
    }
    //vx&vw calculate
    chassis_data->vx = (f_v + b_v);
    chassis_data->vw = (f_v - b_v);
}
```

代码

4.gimbal_task.c

这个线程我没有做控制逻辑上的改动，只是调整了一些控制方式

```
if (shoot_cmd)
  {
      trigger_pos_ref = moto_trigger.total_ecd;
      trigger_pos_ref += 130922 * 2 * trigger_dir;    //shot two bullets at once
      shoot_cmd = 0;
  }
```

二连发

1） pitch 轴运动直接被遥控器摇杆映射而不是累加

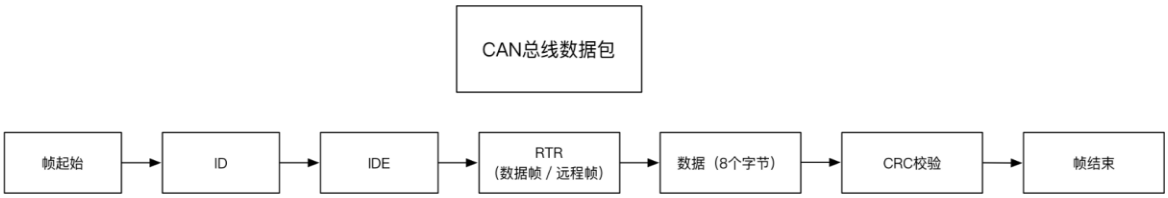
```
case GIMBAL_CLOSE_LOOP_ZGYRO:
{
    pit_angle_fdb = pit_relative_angle;
    pit_angle_ref = rc.ch4 * 35 / 1320 + 7.5;

    yaw_angle_fdb = yaw_zgyro_angle - gYaw.zgyro_offset;
    yaw_angle_ref += -rc.ch3 * 0.0015f
                                - rc.mouse.x * 0.01f;
}break;

default:
    break;
```

5.通信协议

这个工程只用到了 CAN 通信，驱动气缸控制板，车轮和云台




CAN 数据帧的数据结构如下

其中可以发送 8 个字节的数据

```
/**
    *   @brief  send cylinder data to cylinder controller through can
  *     @param  input : switch_data[4]
  *             output: NULL
    *   @note
            cylinder controller ID: 0x501
            data1~data4: cylinder data
            data5~data8: 0
  */

void can_send_cylinder_data()
{
    CYLINDER_CAN.pTxMsg->StdId   = 0x501;
    CYLINDER_CAN.pTxMsg->IDE     = CAN_ID_STD;
    CYLINDER_CAN.pTxMsg->RTR     = CAN_RTR_DATA;
    CYLINDER_CAN.pTxMsg->DLC     = 8;
    CYLINDER_CAN.pTxMsg->Data[0] = cylinder_data[0];
    CYLINDER_CAN.pTxMsg->Data[1] = cylinder_data[1];
    CYLINDER_CAN.pTxMsg->Data[2] = cylinder_data[2];
    CYLINDER_CAN.pTxMsg->Data[3] = cylinder_data[3];
    CYLINDER_CAN.pTxMsg->Data[4] = 0;
    CYLINDER_CAN.pTxMsg->Data[5] = 0;
    CYLINDER_CAN.pTxMsg->Data[6] = 0;
    CYLINDER_CAN.pTxMsg->Data[7] = 0;
    HAL_CAN_Transmit(&CYLINDER_CAN, 1000);
}
```

比如与气缸控制板通信的代码：

其中 1，2，3，4 字节控制一个气缸板通道的开关

# 总结

我们的项目，从组建队伍，讨论方案，到规划设计，加工打造，最终导入程序，实地测试，共历时三个星期。我们的开始，有一个宏伟的蓝图，想要打造 6 支队伍中最强的机器人战车。可惜结尾确实有些差强人意，由于最终留给我们优化的时间不多，即使想做的都能做出来，但是在效率上却是大大的不达标。最终我们实现了巡线，也实现了红外对管的对位。我们两侧都可以抓到障碍块，但是由于缺少优化，车身在宽度和高度上还都超过了规定的尺寸，重量也是所有机车中最重的，达到了 21.1kg。在第二轮的比赛中，我们的机器人需要平均 2 分钟的时间来抓取两侧的两个障碍块，可以说是非常低效的。我们的团队，有一种高开低走，虎头蛇尾的感觉，我们追求处处都做到完美，最中却因为时间不够导致完成度不够，从而落败。综合来说，我们的机器人在技术含量上不输任何一支队伍，却输在了整体进度的规划上，究其原因，是各个部分的成员沟通不到位。

陈思达：
曹瑞翔：
王嘉平：
张帅鹏：
蔡毓鑫：
张嘉轩：
陈辰：