哨兵技术文档

2018/2/2

目录

一、机械部分

1.概述

1.1 背景

Robomaster 从 2015 年发展至今，机器人种类从最初的步兵成机器人发展到现在的步兵机器人，英雄机器人，工程机器人，自动基地等多元素机器人，而今年，取消了自动基地机器人，由官方提供固定的基地机器人，转而添加了新的兵种“哨兵”。

哨兵，比赛初始拥有 500 颗 17mm 直径的小弹丸，在己方启动区设定的固定的轨道上，能够自主移动，依赖视觉，追踪敌方进攻的机器人，并自动反击，枪口热量是其他机器人的 2 到 3 倍，是哨兵的射频可达到 11 发每秒，可对敌方机器人造成致命的伤害，保护基地，并未基地提供 50%的防御加成。

从某种意义上来讲，哨兵的出现，是将基地的自动反击的功能独立出来，以另外一种形式来实现反击的功能，但其要求的机械，嵌入式，视觉条件要比以往更加苛刻，更加复杂，有以下几个新提出的几点要求：

一、让哨兵稳定倒挂行走于轨道上的夹紧行走机构。

二、云台 360 度转，且轻量化，使云台响应及时。

三、弹仓对云台的供弹结构。

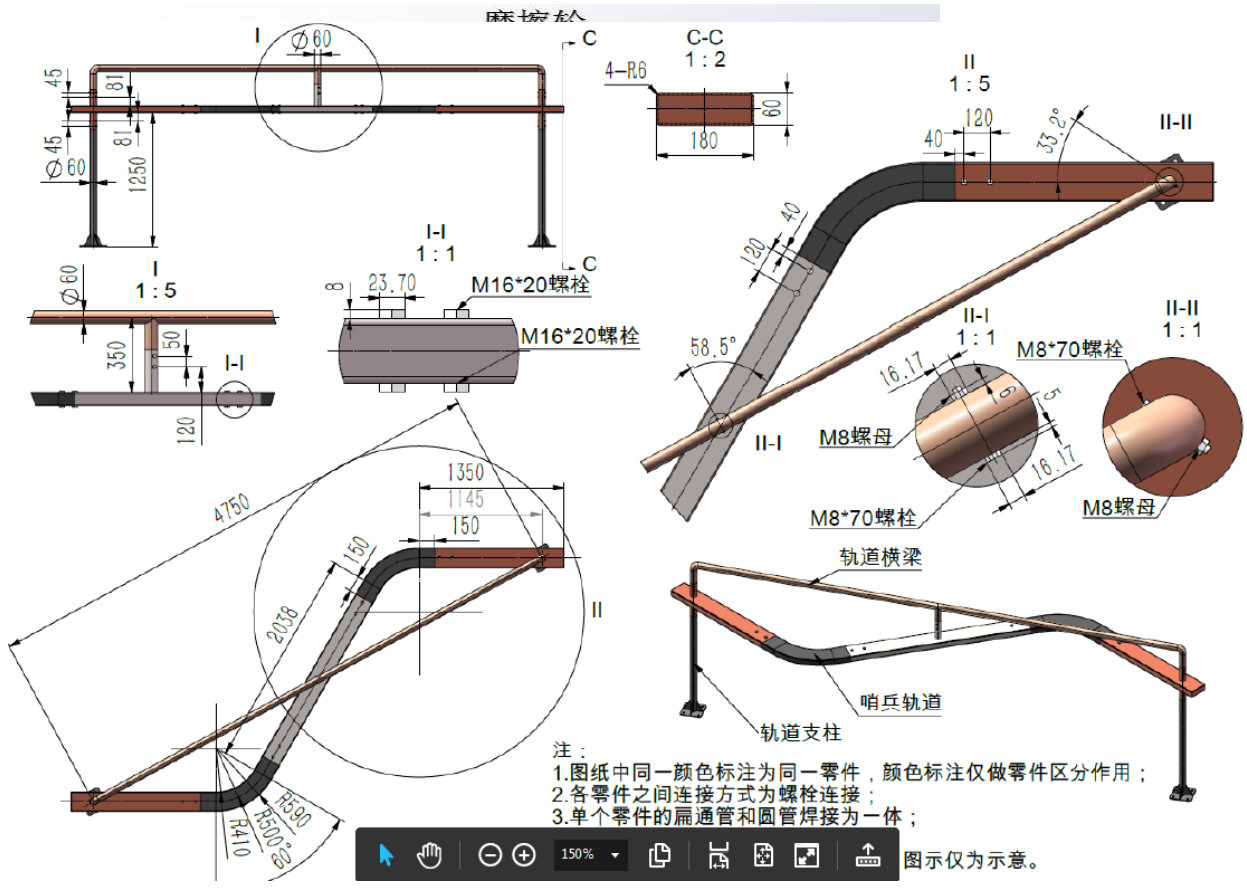
四、哨兵的上层决策，怎样让哨兵的比赛过程发挥最大的作用。

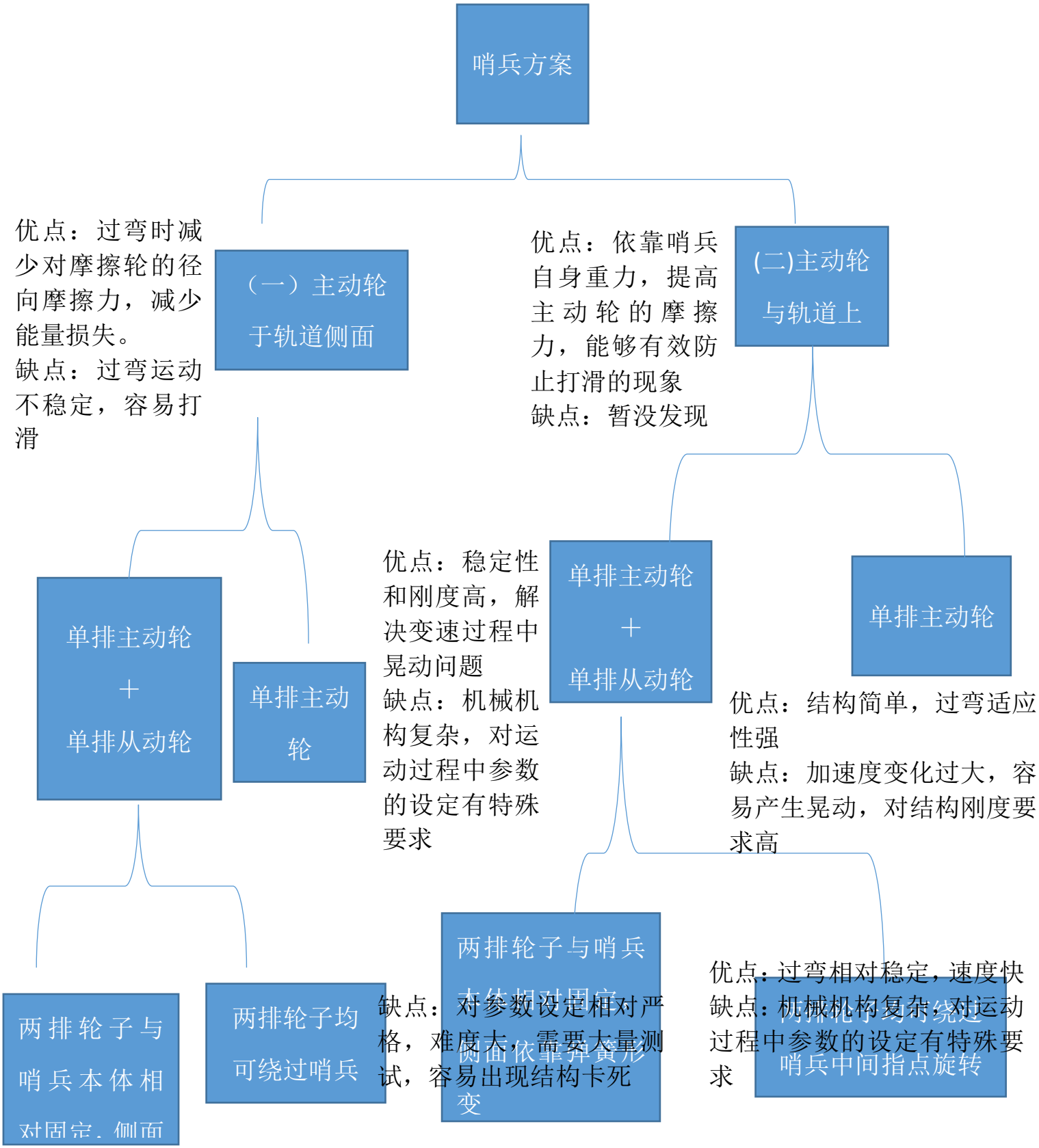
五、哨兵的位置位于启动区与荒地之间，视觉方面需要同时扫描监视荒地和启动区的情况

1.1.1 整体设计说明

轨道夹紧行走快拆结构的方案设计与分析

夹紧行走快拆结构，顾名思义，就是一种倒挂夹紧轨道，稳定行走，并可以在短时间内拆装的结构。根据官方提供的轨道，截面为长 180mm，宽为 60mm 的矩形，相交于传统的圆形管道，不管是矩形轨道的上表面，还是左右两侧平面，能够为机器提供更为稳平稳的平面，让机器的摩擦轮有更大贴近面积，针对官方的矩形轨道，拟定了夹紧行走机构的方案。




哨兵方案
（一）主动轮于轨道侧面
优点：过弯时减少对摩擦轮的径向摩擦力，减少能量损失。
缺点：过弯运动不稳定，容易打滑
(二)主动轮与轨道上
优点：依靠哨兵自身重力，提高主动轮的摩擦力，能够有效防止打滑的现象
缺点：暂没发现
单排主动轮＋单排从动轮
单排主动轮
优点：稳定性和刚度高，解决变速过程中晃动问题
缺点：机械机构复杂，对运动过程中参数的设定有特殊要求
单排主动轮
优点：结构简单，过弯适应性强
缺点：加速度变化过大，容易产生晃动，对结构刚度要求高
两排轮子与哨兵本体相对固定，侧面
两排轮子均可绕过哨兵
两排轮子与哨兵本体相对固定，侧面依靠弹簧形变
缺点：对参数设定相对严格，难度大，需要大量测试，容易出现结构卡死
两排轮子均可绕过哨兵中间指点旋转
优点：过弯相对稳定，速度快
缺点：机械机构复杂，对运动过程中参数的设定有特殊要求

云台 360 度旋转的方案设计与分析

2018 年 robomaster，取消了自动基地机器人，由官方提供固定的基地机器人，转而添加了新的兵种“哨兵”，前面提及到，哨兵的出现，一定意义上，在取代原来基地反击攻击基地的敌方机器人此功能的基础上，还将攻击的方位延伸至启动区门荒地，和桥上的关口位置，这就要求哨兵具备 360 无限位旋转结构或者有限制的角度行程（推荐限制 720 度，参考 osmo 云台）结构。

1.1.2 部分数据计算

1. 快拆结构的设计

G=60N，d1=55mm，d2=35mm

F1*D1=F2*D2　得 F2=47.1N

a 约等于 50 度，正交分解得 F3 约等于 30N. 单根碳板承受 15N，因为碳纤维材料为正交各向异性材料，不同方向表现力学性质也不同，没有固定的弹性模量，在软件中的仿真并不能完全反映现实情况，因此，需承受 15N 的拉力，取保守值，该碳板厚度 3mm。

✧ 该计算过程仅针对竞培哨兵结构。

2. 行走摩擦轮的选择

摩擦轮选择了包胶轮（聚氨酯），外径 55，稍大于 3508 电机的最大直径，以保证在运行中，电机不会与轨道发生干涉，厚度选择 15mm，以保证包胶轮有足够的压缩变形量，提高行走过程中的稳定性。

而在硬度上，主动摩擦轮和从动摩擦轮有区别：

在行走中，主动轮的滚动摩擦力，是提供驱动力，而从动摩擦轮的滚动摩擦力则形成阻力，滚动摩擦力实质上是静摩擦力。接触面愈软，形状变化愈大，

则滚动摩擦力就愈大，按照此准则，主动摩擦轮的硬度要比从动摩擦轮底，在一次模拟试验中，拿不同硬度的摩擦轮在轨道上，模拟行走情景，大致给定主动摩擦轮邵氏硬度 45，从动摩擦轮邵氏硬度 55.

✧ 方案确定后通过简单的建模，在 SolidWorks 插件 motion 中直接仿真行走。确定稳定性后再继续细化模型。

2.易损结构说明

1. 云台，在哨兵被拆下来的的时候，云台处于悬挂状态，放置地面时，必须侧着放 ，如果放置不好，或者侧放时被弄翻，对云台链接本体的连接处容易受损。
2. 主动轮摩擦轮，一旦哨兵暴走，撞在两侧柱子上，摩擦轮处于原地打转状态，对摩擦轮耗损很大。

3.迭代

哨兵的研发时间只有不到一个月的时间，只做了第一代，所以没有迭代，但是第一代的版本中，各个结构均在测试中达到预期期望：

1. 装上轨道后，运行稳定，不出现明显晃动。
2. 过弯流畅，稳定
3. 稳定供弹，不卡弹
4. 云台同步带松紧合适，云台转动阻力小。
5. 拆装方便。

唯一遇到的的问题就是重量，加视觉需要的 txone，布线后，超出规则中重

量上限 0.4kg，通过减去没有作用的零件，改为换其他材料，问题很快就解决。

4.优化

重量还有优化的空间，最后一次的减重，有些零件的被替换了，第二次迭代的话，在重量上把控好，以保证实物和图纸一一对应。

二、嵌入式

2.1 概述

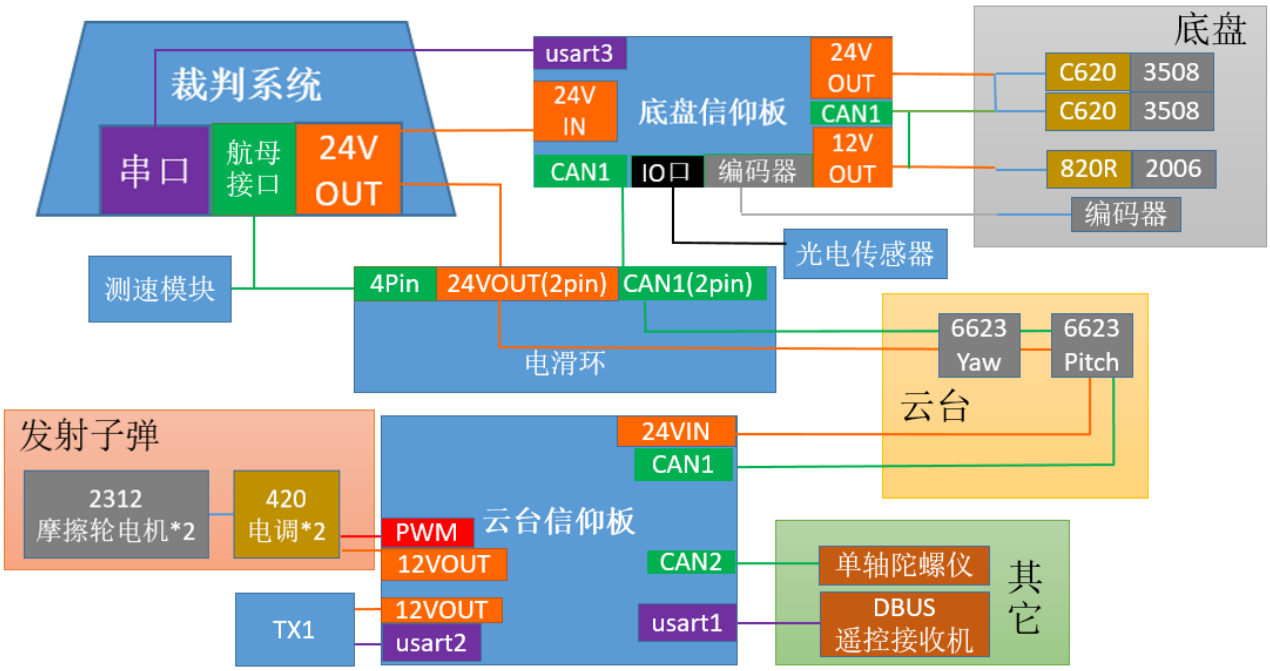
2.1.1 电子器件清单

电机	数量（个）	电路板	数量（个）
3508	2	信仰板	2
6623	2	中心转接板	1
2006	1	其它模块	数量（个）
2312	2	单轴陀螺仪	1
电机总和	7	光电传感器	1
		正交编码器	1
		遥控器接收机	1

2.1.2 嵌入式硬件、软件资源配置

云台信仰板	用途
CAN1	3508、6623、2006电机、底盘信仰板通讯
CAN2	单轴陀螺仪
USART1+DMA	遥控接收机
USART2+DMA	和TX1通讯
TIM12	2312电机PWM输出
GPIO	LED灯、激光、拨弹开关
SPI	板载IMU
TIM3、TIM4	LED灯频闪，用于显示系统运行情况
底盘信仰板	**用途**
CAN1	与云台信仰板通讯
USART3+DMA	读取裁判系统数据
GPIO	LED灯、光电传感器
TIM2	读取编码器数据

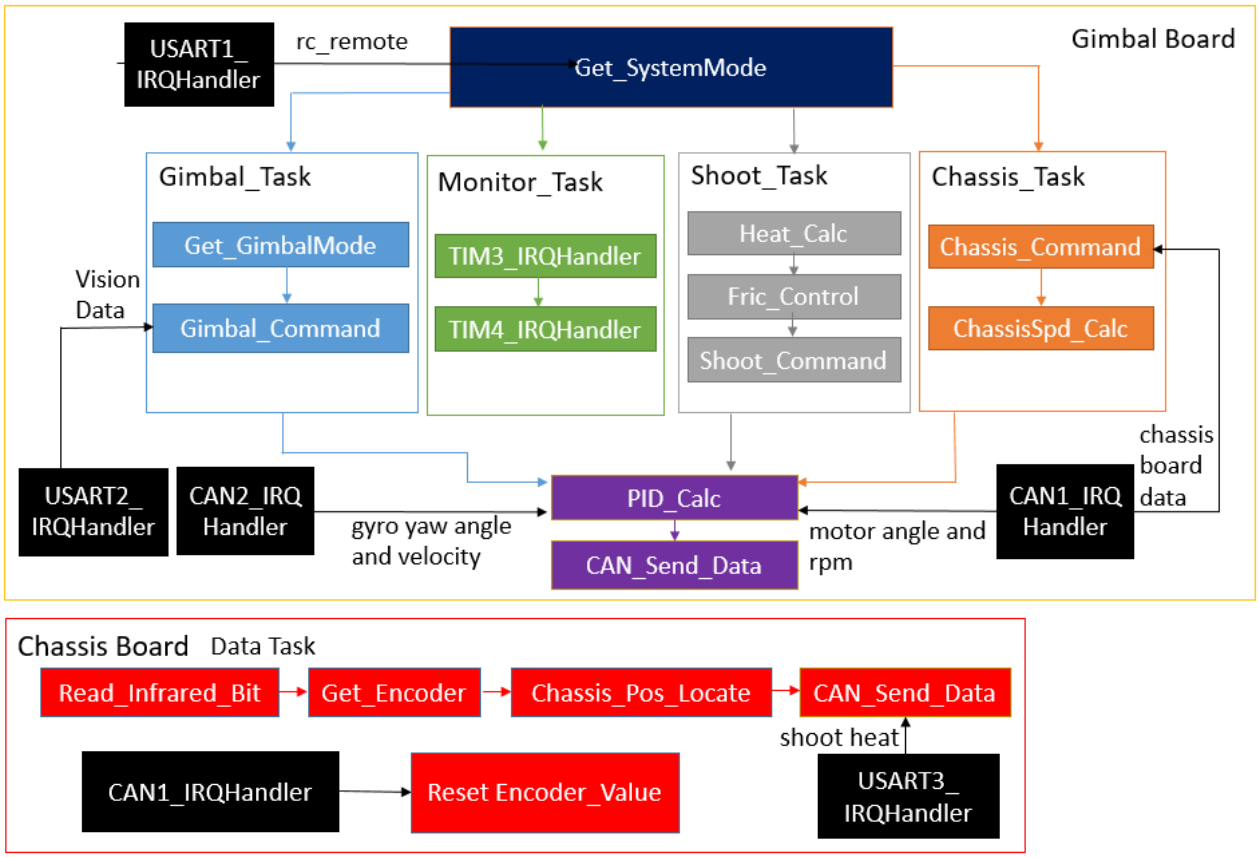
2.1.3 嵌入式系统框图



云台信仰板控制所有电机，与底盘信仰板通讯，串口接单轴陀螺仪、TX1。

底盘信仰板读取编码器、裁判系统以及光电传感器的数据，进行底盘定位计算，并用 CAN 总线将哨兵定位信息、裁判系统数据发送给云台信仰板。

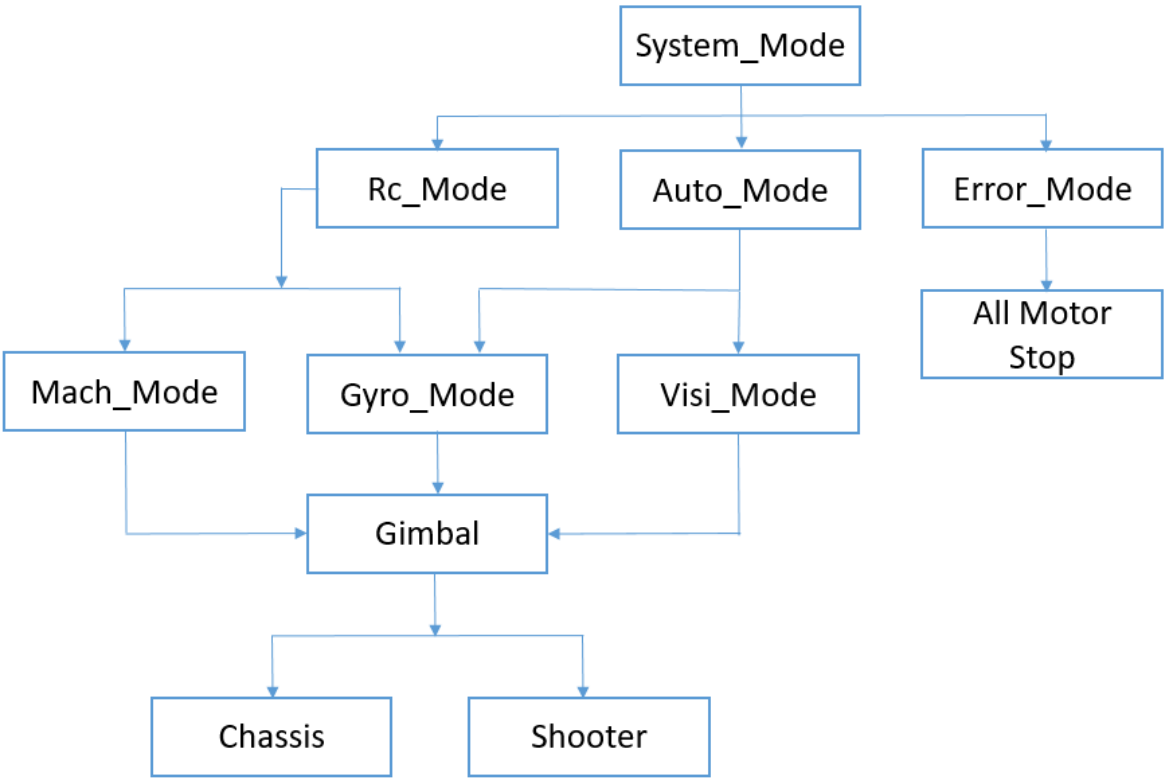
2.1.4 软件架构



本哨兵嵌入式控制系统由上到下分为 3 层：指令层、控制层、驱动层

指令层

最上层为指令层，首先是通过读取遥控器数据，获取当前哨兵系统运行模式。其次是获取云台模式，在遥控模式下，由遥控器拨码开关切换云台模式，自动模式下，通过读取 tx1 数据，自动切换相应模式，三种云台模式对应三套 PID 参数。获取了运行模式后，再发送相应指令分别控制各电机运转。



控制层

中间层为控制层，包括云台、底盘、射击、系统监视四个部分，控制层接收命令层的命令和驱动层反馈的数据，对云台、底盘、射击等电机进行 PID 计算，并将计算结果发给驱动层。系统监视负责监视整个系统所有串口和 CAN 总线的通讯状况，一旦某个模块掉线，板载 LED 作出与之对应的闪烁，同时停止该模块的运作，防止暴走。

驱动层

底层为驱动层，包括串口、定时器、CAN 总线、GPIO 等外设的初始化与驱动，并读取电机、传感器、裁判系统等模块数据，反馈给控制层，同时将控制层的计算结果发送给电调，实现电机 PID 控制。

2.2 细节

2.2.1 方案优化流程

①底盘：定位一开始只用了一个编码器来计算哨兵走过的路程，从而实现定位，后来果然和预期一样出现了累计误差，因此加多一个光电传感器，当哨兵经过轨道中间的柱子时，光电传感器电平跳变，从而校准编码器数值。此外一开始，底盘电机和云台电机分别是两块板控制，但是后面调试的时候，debug 太麻烦了，所以改成所有电机由云台主控控制。

②云台巡视监测：yaw 轴云台需要来回扫描监测，这就需要一个绝对或者相对底盘绝对的角度以确保每次都能扫描到同个地方，而陀螺仪 yaw 轴数据会漂，yaw 轴云台电机采用 2:3 的同步轮驱动方式，每转一圈，云台相对底盘的绝对角度会变化，这个问题不解决，后面的决策将难以进行，最后是用电机编码器数值自己计算出一个相对底盘的绝对角度，精度足以满足需求。

③拨盘的卡弹问题：由于弹仓采用大拨盘，而且弹量有 500 发，有时候存在卡弹现象，增加卡弹电机反转的功能，彻底解决了卡弹问题。

④识别射击最短距离决策：刚开始，哨兵识别到目标后，若云台不处于中心而是转向底盘一侧射击目标，则底盘也往这一侧移动（类似步兵的底盘跟随），直到目标进入射击范围或到达轨

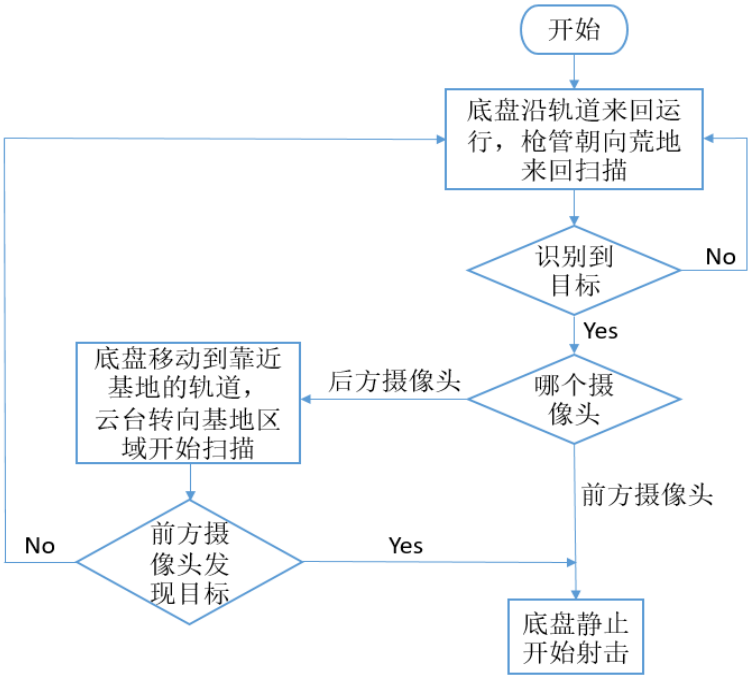
道边缘。后来发现了两个问题，第一个是当云台处于底盘跟随的角度临界时，云台左右转动，会使底盘的速度忽左忽右，产生抖动。第二个是对于运动的目标，接受到的目标距离数值跳变过大，以此为依据判定是否进入射击范围显然是不靠谱的。综上，为保证识别射击稳定不丢帧，识别到目标后，底盘保持静止。

⑤调试复位功能：由于信仰板没有复位按键，而哨兵定位方案需要在特定的地方上电启动才能正确运行，且 TX1 由主控供电，为了方便调试，避免反复开关电池，增加底盘主控复位功能，当哨兵进入失控保护状态，云台主控通过 CAN 总线发送命令让底盘主控复位定位数据，同时底盘主控 LED 灯由绿变红。

⑥基地区域防御决策：哨兵后方摄像头负责监视基地区域，且打击优先级高于前方摄像头，当后方摄像头发现目标时，云台立即掉转 180 度。这种决策有两个问题，第一，目标往往是运动的，等云台转过 180 度后，目标已经跟丢了；第二，由于轨道较长，云台扫描角度较大，有时候后方摄像头发现的目标不一定在基地区域，而等云台转过去后，目标可能已经跑到了基地区域，但由于哨兵转向后看的是非基地区域，跟丢目标后，哨兵将会复位云台，从而错失了战机。因此，决策改成，当后方摄像头发现目标，底盘运动到靠近基地区域的轨道位置，云台固定转向基地方向，开始扫描基地区域，并且在没有发现目标后，在云台复位之前会再次扫描一遍基地区域。

⑦哨兵识别到目标后将切换攻击模式，判断依据是 TX1 的标志位，但由于识别中存在个别空帧，导致发来的标志位存在跳变，哨兵的模式也因此在攻击和巡视之间跳变，极大地影响了后续的决策以及射击，因此在接收标志位后，以自己的标志位为依据切换模式，只有视觉标志位累计到一定数值后，才会退出攻击模式。

2.2.2 哨兵运行流程



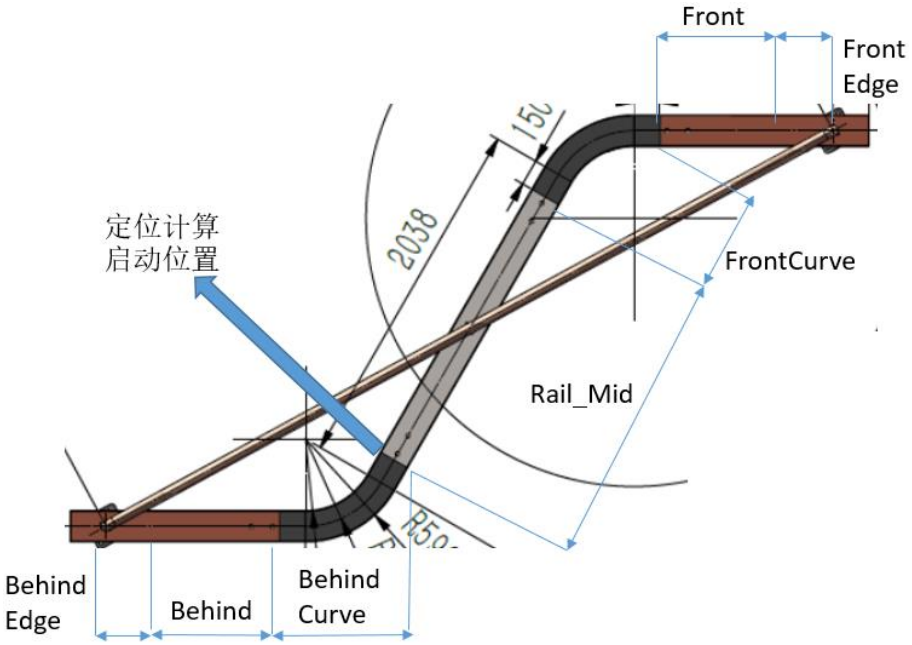
2.2.3 关键算法

①利用电机编码器计算相对底盘的绝对角度，记录单位时间内，云台编码器转过的角度，得出一个角度数值，当哨兵云台旋转一圈，角度数值变化范围 0~12258，类似云台编码器。

```
if((int32_t)(currentTime - loopTime_Gimbal) >= 0 )//gimbal abs origination
{
    loopTime_Gimbal = currentTime + 3000;
    if( !Gimbal[YAW].Init && M6623_Gimbal_Data[YAW].Angle!= 0 )
    {
        Gimbal[YAW].Agl = Gimbal_MaxLimit/2;
        Gimbal[YAW].Last_Agl =  M6623_Gimbal_Data[YAW].Angle;
        Gimbal[YAW].Init = 1;
    }
    Gimbal[YAW].Agl_Error = M6623_Gimbal_Data[YAW].Angle - Gimbal[YAW].Last_Agl;
    if( abs(Gimbal[YAW].Agl_Error) <= 1 )//忽略数值跳变
    {
        Gimbal[YAW].Agl_Error = 0;
    }
    else if( Gimbal[YAW].Agl_Error > 7000)//云台电机编码器临界处理
    {
        Gimbal[YAW].Agl_Error = Gimbal[YAW].Agl_Error - 8191;
    }
    else if( Gimbal[YAW].Agl_Error < -7000 )
    {
        Gimbal[YAW].Agl_Error = -8191 - Gimbal[YAW].Agl_Error;
    }
    Gimbal[YAW].Last_Agl = M6623_Gimbal_Data[YAW].Angle;
    Gimbal[YAW].Agl += Gimbal[YAW].Agl_Error;
    if( Gimbal[YAW].Agl > Gimbal_MaxLimit )
    {
        Gimbal[YAW].Agl = 0;
    }
    else if(  Gimbal[YAW].Agl < 0 )
    {
        Gimbal[YAW].Agl = Gimbal_MaxLimit;
    }
    Gimbal[YAW].Agl_Error = 0;
}
```

②哨兵定位方案

底盘主控需要在特定位置上电启动，定位计算才能正确运行。将轨道分为 7 段。调试时事先用编码器读取每一段轨道的长度，另外当哨兵路过 Rail_Mid 段中心的柱子时，光电传感器电平跳变从而校准编码器数值。比如哨兵从启动位置开始运行，当编码器数值大于 Rail_Mid 轨道值，当前位置变为 FrontCurve，上一次位置变为 Rail_Mid，当编码器数值大于 Rail_Mid+FrontCurve 轨道距离，当前位置变为 Front，上一次位置变为 FrontCurve，同时清零里程数，从位置段位临界处重新计数，防止数值溢出。



定位部分代码

```
Data.RailDis += Get_Encoder_Dis();
Data.Infrared = GPIO_ReadInputDataBit(GPIOD,GPIO_Pin_14);
if( Data.ChassisPos == Rail_Mid )
{
    if( Data.Last_ChassisPos == Rail_Mid || Data.Last_ChassisPos == Rail_BehindCurve )
    {
        if( Origin == Rail_MidBehind )
        {
            if( Data.RailDis > Rail_Mid_Dis )
            {
                Data.ChassisPos = Rail_FrontCurve;
                Data.Last_ChassisPos = Rail_Mid;
            }
            else if( Data.RailDis < 0 )
            {
                Data.ChassisPos = Rail_BehindCurve;
                Data.Last_ChassisPos = Rail_Mid;
            }
        }
        else if( Origin == Rail_MidFront )
        {
            if( Data.RailDis < -Rail_Mid_Dis )
            {
                Data.ChassisPos = Rail_BehindCurve;
                Data.Last_ChassisPos = Rail_Mid;
            }
            else if( Data.RailDis > 0 )
            {
                Data.ChassisPos = Rail_FrontCurve;
                Data.Last_ChassisPos = Rail_Mid;
            }
        }
    }
```

2.3 总结

预期效果：1.可以丝滑地通过弯道

2.哨兵可以准确知道自己在轨道上的具体位置

3.在 5m 内可以轻易命中静止或者摇摆的目标

4.云台扫描范围覆盖基地区、桥洞顶以及荒地，并对这些地方的目标具有一定的战术威慑力

最终效果：相比预期效果，没有做到的地方是云台扫描范围没有覆盖到桥洞顶，难以命中荒地上移动的目标以及云台扫描对于基地区域存在一定的盲区。

方案创新点：1.利用自行计算相对底盘的绝对角度作为云台闭环的反馈

2.目前的定位方案其实已经超出了哨兵底盘单纯的自动运行需求，但是具体完善的定位信息对于后续的决策时很有必要的。

3.当后方摄像头识别到目标后，底盘无论在哪里，都会运动到靠近基地区域的指定位置停下。

4.对于基地区域，云台在开始和结束前都会扫描多一次，确保不会遗漏。

5.在没有识别到目标时，云台将根据定位信息，调整不同区域扫描的角度，确保有效扫描效率的最大化。

6.视觉识别有时候存在误识别情况，会抬高云台识别到场地外的灯光，哨兵会根据情况判定自身是否误识别，并将云台低下，避免一直抬高误识别。

存在的坑：自动运行下，偶尔存在跑到轨道边缘没有减速直接撞边缘的情况，看了 debug 是定位信息出现了跳变，原因不明。

可以改进的地方：

1. 舍弃用光电传感器检测轨道中间柱子的方法，改用两个光电传感器装在哨兵两侧，用来检测是否到达轨道边缘，也可防止哨兵暴走怼边缘的情况。

2. 目前基于各种条件限制，难以命中荒地上快速移动的目标，并且由于需要扫描荒地，导致哨兵怼基地区域的监视存在一定的盲区，因此可以放弃对荒地部分区域的监视，以确保可以百分百监视基地区域。

2.4 附录

操作手册

若开启遥控器，哨兵为遥控模式

S1 发射控制与步兵相同，

S2，上：失控保护，复位位置信息和云台自计算角度，底盘主控 LED 变红

中：云台以陀螺仪作反馈

下：云台以自行计算的角度作反馈

遥杆（日本手）：ch2 控制底盘，ch0 控制 yaw 轴，ch1 控制 pitch 轴

若关闭遥控器或者关闭接收机，哨兵进入自动运行模式，哨兵底盘和云台的启动位置必须如图所示。

哨兵上电后将在 180s 后启动，同时摩擦轮开启，若上电后进入过遥控模式，再进入自动运行模式，则不需要等待 180s。

云台主控 LED 灯显示系统运行状况：

LED显示	事件
绿灯常亮	系统运行正常
绿灯一闪	没有收到过TX1发来的数据
红灯常亮	遥控器掉线
红灯一闪	底盘主控掉线
红灯二闪	外部陀螺仪掉线
红灯三闪	底盘电机掉线
红灯四闪	pitch轴云台电机掉线
红灯五闪	yaw轴云台电机掉线
红灯六闪	拨弹电机掉线

三、算法部分

1.整体

1.1 功能需求

多摄像头装甲板检测：

功能：用多摄像头检测哨兵周围地区敌人

- 优先锁定英雄车
- 清楚知道敌人在哨兵的前方，桥头或者基地附近
- 多摄像头轮流读图，节省 CPU 负载

1.2 硬件环境：

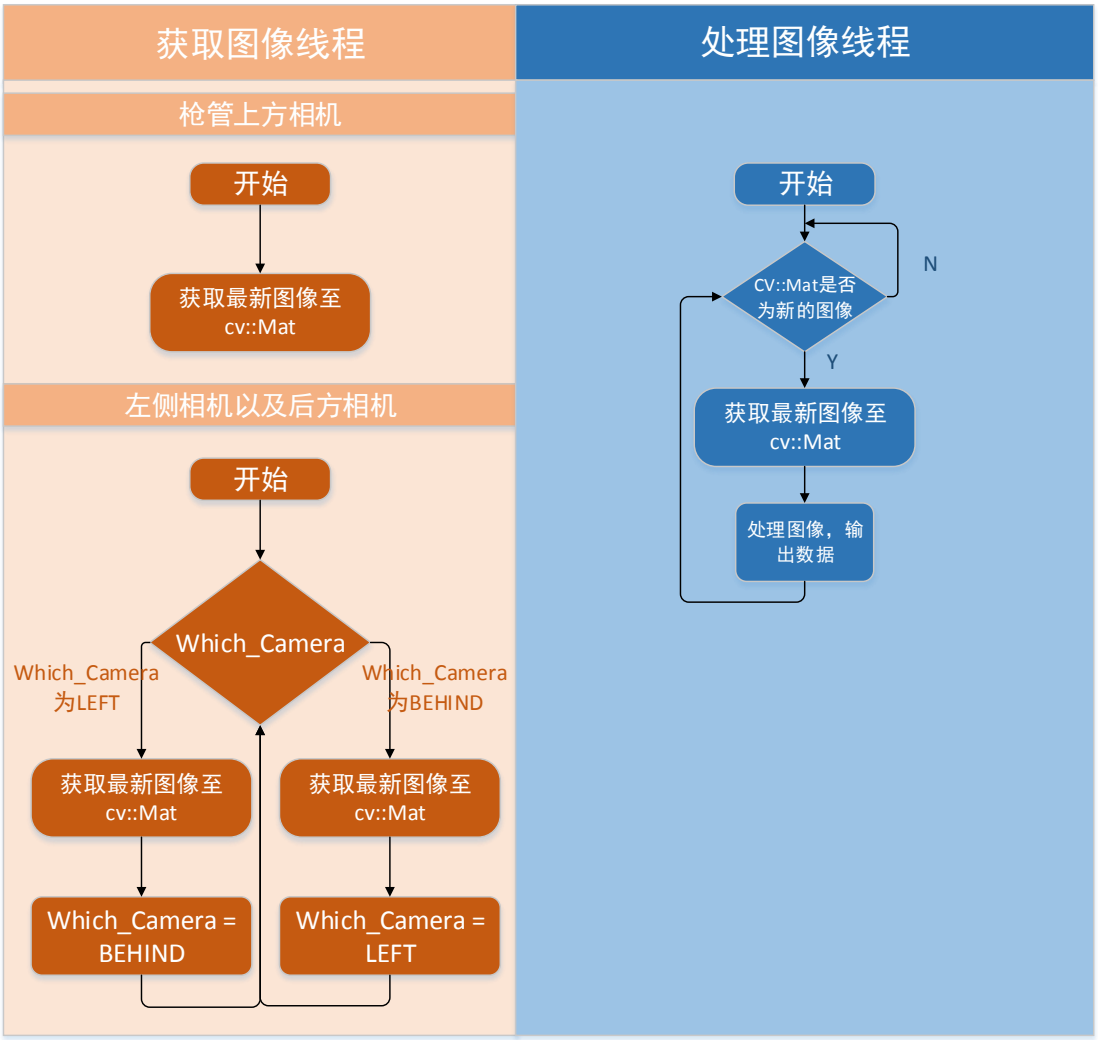
- 一组：NUC + 工业相机 + 2 个普通 USB 相机（总决赛和半决赛一组决定只用一个摄像头）
- 二组：TX1 + 2 个普通 USB 相机

1.3 软件环境：

- Ubuntu16.04 系统+ROS Kinetic

1.4 总体系统框图：

因为需要用到多摄像头进行对装甲板进行检测识别，一个摄像头放在枪管上面，用于检测哨兵前方荒地区域有无敌方，还有用于枪管的自动跟踪瞄准；左方的摄像头专门用于检测桥头方向有无敌人；后方的摄像头专门用于扫描基地区域有无敌人。用了两个线程，一个线程用于获取图片，另外一个线程用于处理图像，获取装甲板相对于相机的位置。

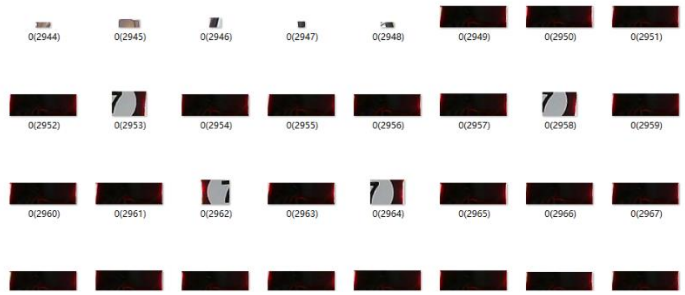


1.5 算法方案:

- 对目标颜色通道进行阈值二值化，同时加上滤掉白色灯光的部分，然后进行一个开操作
- 提取所有的灯柱，通过对所有灯柱进行模板匹配进行筛选出左右灯柱
- 根据每个灯柱偏离垂直方向的角度以及水平方向角度等阈值排除灯柱，并左右灯柱包围成矩形组合成待挑选装甲板，每个矩形区域都有根据以下先验知识进行筛选和评分：
  - 左右灯柱角度差绝对值
  - 左右灯柱偏离垂直方向的角度的绝对值
  - 左右灯柱中心点的连线与水平线之间的角度值
- 对目标区域进行透视变换，变为正视图，利用以下先验知识对目标区域进行筛选：
  - 根据大小装甲长宽比筛选目标区域
  - 若存在上一次的检测目标，则当前目标与上一次目标的宽度和高度差不能超过一定阈值
  - 灯柱周围区域的目标通道灰度均值需大于其他通道的灰度均值
  - 黑色装甲中间区域绿色通道的梯度值较大的比例小于一定阈值

- 因为 Hu 矩阵是不变矩，所以用 SVM+Hu 矩来做一个分类器，把误识别部分和正确部分的 Hu 矩进行一个 SVM 训练，然后剔除 SVM 预测出是误识别部分的目标区域；

  - 误识别部分：



  - 正确部分：

  - 然后进行训练和保存：

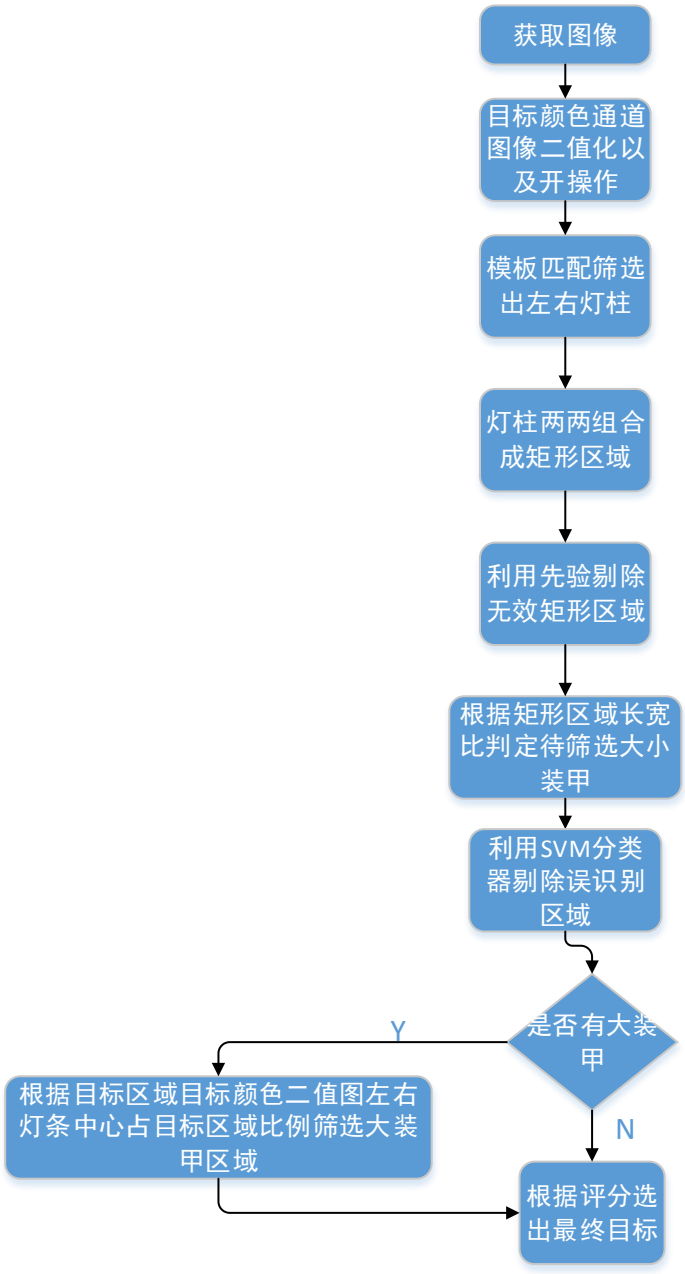
```
svm->train(trainingData, cv::ml::SampleTypes::ROW_SAMPLE, trainingLabel);
svm->save("/home/mcjun/Armor/src/SVM.xml");
```

  - 然后在预选目标区域中进行筛选：

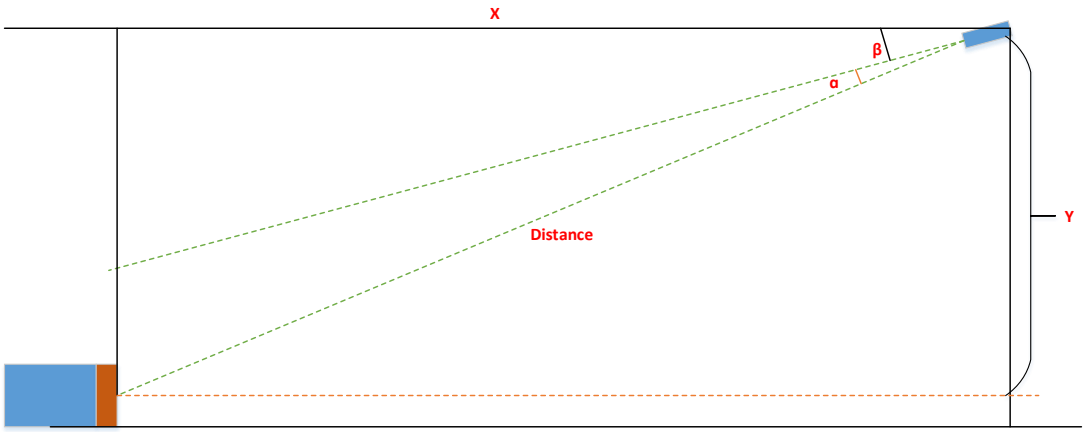
```
Moments mts;
mts = moments(black_part_gray);
float Hu[7];
HuMoments(mts, Hu);
Mat result = Mat::zeros(1, 7, CV_32FC1);
for (int i = 0; i<7; i++)
{
    result.at<float>(0, i) =  (float)Hu[i];
}
int re = _svm->predict(result);
if(re == 0) continue;
```

- 若剩下的目标区域根据长宽比阈值判定是大装甲板，则对目标区域目标颜色二值图左右灯条中心矩形区域占目标区域的比例一定阈值进行筛选，若筛选出多于或等于一个大装甲板目标区域则选取评分最高的为目标；若都没有大装甲板目标区域，则根据选取评分最高的小装甲板区域为目标；
- 若剩下的目标区域根据长宽比阈值判定是小装甲板，则根据选取评分最高的小装甲板区域为目标；
- 每次识别到相应的装甲板后，只在其所在的一小部分进行识别运算，从而减小运算量，当目标丢失超过 10 帧时时，扩大搜索范围，不断累积，直到遍历全图；
- 当连续识别到目标连续超过 5 帧才判定为识别到目标，才进行位姿的输出。

实现的总体方案如下面流程图所示：


获取图像
目标颜色通道图像二值化以及开操作
模板匹配筛选出左右灯柱
灯柱两两组合成矩形区域
利用先验剔除无效矩形区域
根据矩形区域长宽比判定待筛选大小装甲
利用SVM分类器剔除误识别区域
是否有大装甲
Y
N
根据目标区域目标颜色二值图左右灯条中心占目标区域比例筛选大装甲区域
根据评分选出最终目标

1.6 枪管偏移



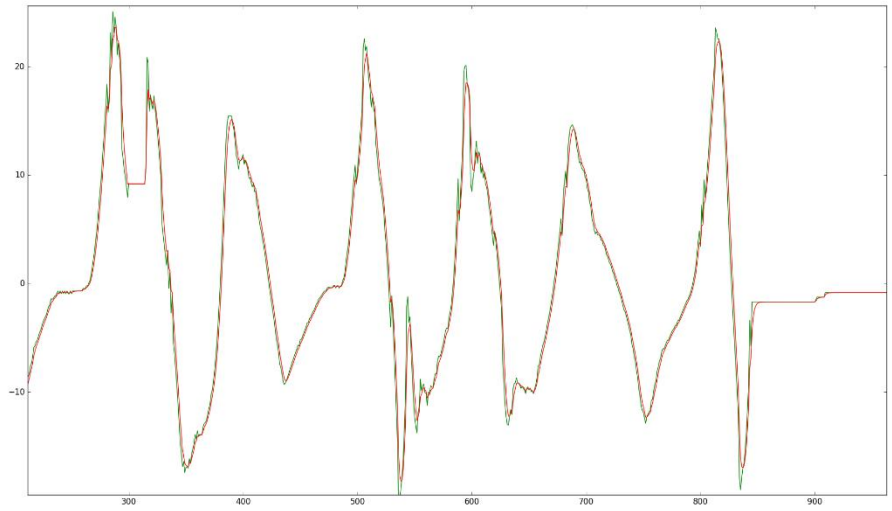
$$\begin{cases} X = V\cos\theta * t \\ Y = V\sin\theta * t - \frac{1}{2}gt^2 \end{cases}$$

$$\mathrm{X} = \mathrm{Distance} * \cos(\alpha + \beta)$$

在近似情况下，α为通过 PnP 计算出来的角度，β为云台当前 Pitch 轴角度，θ为需要计算出来云台的 pitch 角度

根据以上公式就可以计算出云台 pitch 角度进行有效打击。

1.7 平滑滤波平滑输出数据



拟合前五帧的数据进行平滑滤波。

1.8 多摄像头处理图像

因为云台摄像头不仅用于搜索敌人，还用于瞄准跟踪射击敌人，所以云台的摄像头用一个节点读图处理，不进行轮流读图；

左方和后方的摄像头用另外一个节点，进行隔帧轮流读图处理图像，当轮到某一个摄像头发现敌方时会发送对应该摄像头的标志位给 stm32 进行转向瞄准射击。

1.9 串口通信

byte0	byte1	byte2-byte5	byte6-byte9	byte10-byte13	byte14-byte17	byte18-byte21	byte22-byte25	byte26
8bit Int8_t	8bit Int8_t	32bit float	32bit float	32bit float	32bit float	32bit float	32bit float	8bit Int8_t
帧头	云台相机是否看到敌方的标志位	装甲板与目标点水平方向的偏差	云台需要抬高的角度	装甲板与相机的距离	后方相机是否看到敌方的标志位	左方相机是否看到敌方的标志位	输出的帧数	帧尾

1.10 开机自启动配置

运用了 ROS 包中的 robot_upstart 包

2.总结

2.1 预期效果与最终结果

- 预期效果：前方云台 100 度缓慢旋转扫描能迅速发现过往敌方 4m 以内车辆，其他相机能灵敏发现敌方；当发现敌方时云台会转到相应的位置迅速锁定敌方进行打击。
- 最终效果：在比赛中前方摄像头基本能迅速发现过往敌方 4m 以内车辆，其他摄像头在近距离可以迅速发现敌方，但距离变远时而且哨兵运动过快时就不能很好的发现敌方。

2.2 方案的创新点

- 在一个处理器里连接了三个摄像头，采用了隔帧轮流读图并处理图像，这样

子处理使得 cpu 的负载不会太大，减少程序崩溃的可能。

- 多摄像头互相配合可以使哨兵几乎无死角的发现敌方，做到迅速响应并且打击敌方。

2.3 存在的坑

- 视觉算法有时候会误识别到场地上方的灯柱和两个装甲板之间的区域

解决方法：

- 用 SVM+Hu 矩来做一个分类器，把误识别部分和正确部分的 Hu 矩进行一个 SVM 训练，然后剔除 SVM 预测出是误识别部分的目标区域；
- 对目标颜色通道进行阈值二值化时加入对白色的阈值筛选，然后再进行一个开操作，这样子误识别就会减少；
- 当得知云台的 pitch 轴的时候，当误识别到上方白灯时，结合 pitch 轴角度与 PnP 计算出来目标区域相对于镜头的角度即可剔除误识别区域。

2.4 可以改进的地方

- 当左方摄像头或者后方摄像头识别到敌方时，可以发出敌方相对于左方或者后方摄像头的相对位置，使云台可以根据当前 yaw 轴的角度快速知道敌方相对于当前的位置并快速转到该位置进行锁定攻击，由于后期的大部分时间都

花在调试上，这个细节就没有去做；

- 预测敌方的运动，因为相机只有 30 帧，如果帧与帧之间物体运动速度特别快，整个控制系统的延时不能满足跟踪需求。因此需要结合陀螺仪返回的 pitch 角和 yaw 角信息做一个预测。如果出现丢帧情况，该帧的目标数据能通过历史帧的图像像素坐标进行二次拟合得到修正。在之前的手自一体的测试中，我尝试结合前 5 帧对运动敌方进行预测，当敌方小车在运动时效果还行，但是当速度有急剧突变时，云台就会抖，呈一个发散的状态，越来越抖，后来如果放在嵌入式上做预测控制上会好很多；

  这一次我只预测 1 帧的数据，该方法会减少掉帧对控制系统的影响。

3.元件清单

组别	一组	二组
处理器	NUC	TX1
云台相机	CMOS迷你工业相机90°镜头	1920x1080的90°USB摄像头
辅佐相机	罗技C270摄像头 KS2A17	KS2A17