2018 RoboMaster 夏令营

技术报告

第 一组

2018.8

目　录

1.机械部分

1.1 整体设计说明

1.1.1 第一阶段

第一阶段的任务是夹取弹药箱并抬起地面，在机械方面，有两个自由度，分别是夹取和抬起，决定方案时主要考虑了以下几点:

1. 时间很短，在满足功能的前提下，实现起来越简单越好；
2. 第一阶段的工作对后阶段有迭代效果；
3. 第一阶段对官方给的车不进行大改，在原有的基础上加上夹取和翻转机构。

1.1.2 第二阶段

第二阶段机械方面要求主要有一下几点：

1. 取弹过程夹具至少需要 3 个自由度；
2. 调整整车各个模块的位置，将整车、底盘尺寸改小，并融合夹具。
3. 融合嵌入式的一些传感器、摄像头的安装。

1.2 设计方案

1.2.1 夹具

根据任务需求，确定整个夹具至少需要 3 个自由度，分别是夹取、翻转和升降。原理及实现方式如下表所示：

表 1

自由度	夹取	翻转	升降
动力	气动	3508 电机	3508 电机
实现方式	气动配合导轨	电机直驱	配合同步带轮、滑块和导轨
优缺点	夹取速度快，力足	速度快，力足；缺点是比较重	缺点：一边驱动，滑块导轨容

			易卡主，运动不顺畅

第一阶段

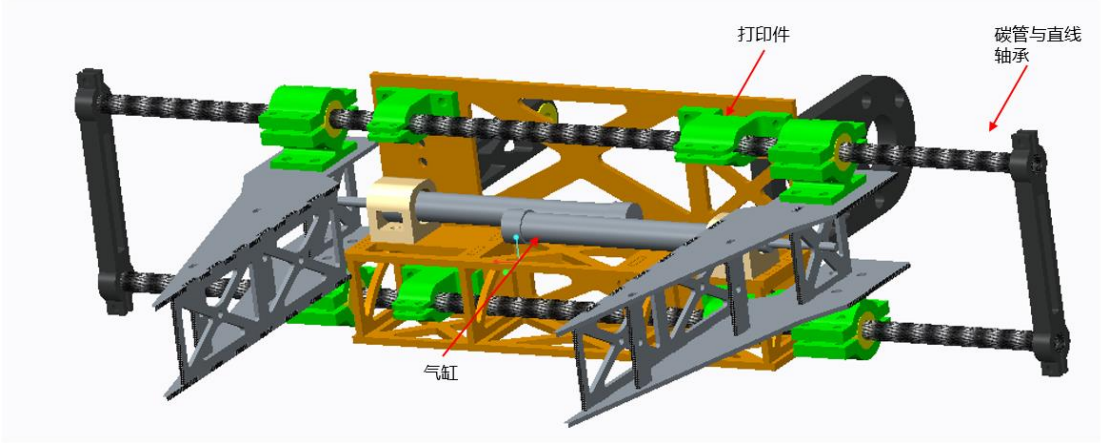


图 1

图 1 是第一阶段的夹具，该夹具的优点在于用碳管与直线轴承配合当导轨的使用，质量轻，且结构牢固，夹取效果比较好。

细节：使用过程中，打印件出现断裂破坏的情况，改变了一下打印件的结构，即改变它的传力路径，效果明显，后来没出效果断裂现象。由于本组的机器是用 3D 打印件最多的一组，之前用过打印件，大改知道其材料的力学性能，所以需要对传统结构进改变，即改变它的传力路径，或者用在不承受主力的地方，主要起链接固定的效果。3D 打印件使用的有点是方便，结构不用考虑加工，只要让其力学性能最优即可，缺点就是 3D 打印件的材料性能不够。

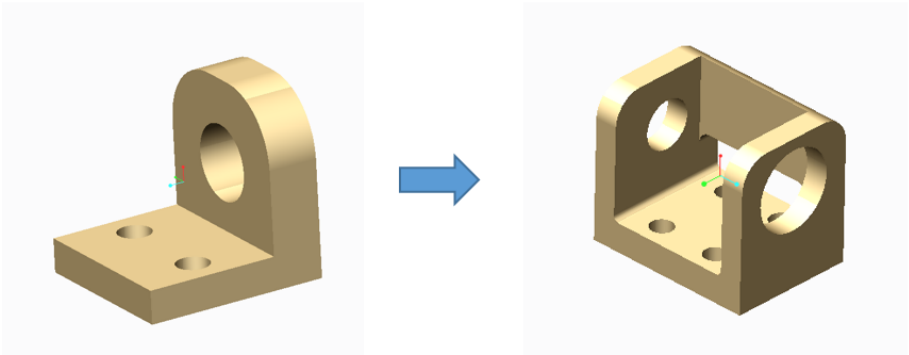


图 2

第二阶段

考虑到夹具可以加上矫正弹药箱姿态的功能，来降低摄像头对弹药箱识别的要求以及提高夹取速度，所以加上其附属机构。做了一个简单的实验，实验效果不错，如视频所示。

试验.mp4

所以第二阶段的夹具一开始加上了矫正功能，其结构图如图 3 所示。

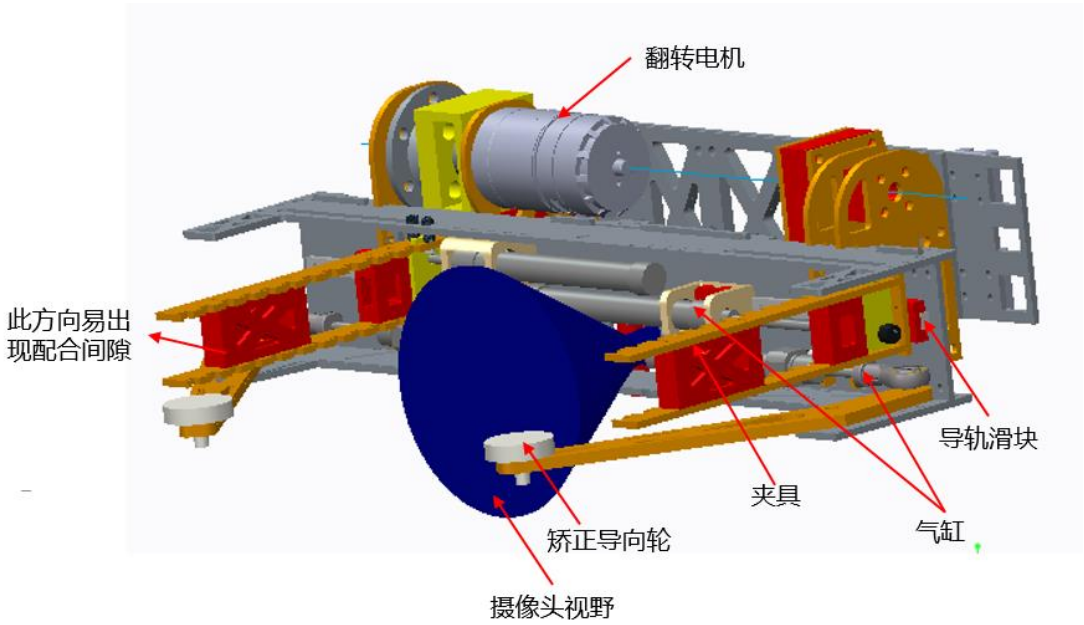


图 3

细节：第二版夹具与第一版相比较，没有第一版的使用效果好，主要原因：

1. 导轨的变化，由直线轴承套碳管改变成滑块导轨。第一种的双导轨的方式结构更牢固。
2. 第二种的结构方式容易出现间隙，使其夹取不牢。

下面是夹具改进的过程：

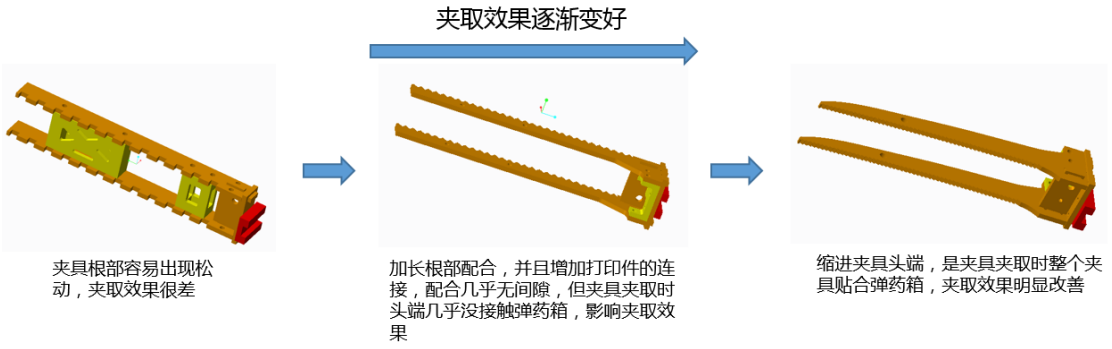


图 4

首先试用没有加上矫正姿态机构的夹具，发现不用矫正姿态的功能也能使用，后面就没有急着加上矫正姿态的机构，也由于时间关系，没有来得及实验，后面的夹具都没有把矫正姿态的模块加上去，一大遗憾!!!

1.2.2 整车

对整车一些模块进行了位置的调整，如图 5 所示：

灯柱与电池都放在底盘下层以降低机器的重心，由于去除悬挂，所以底盘的安装板尽量镂空以增加底盘的柔度。

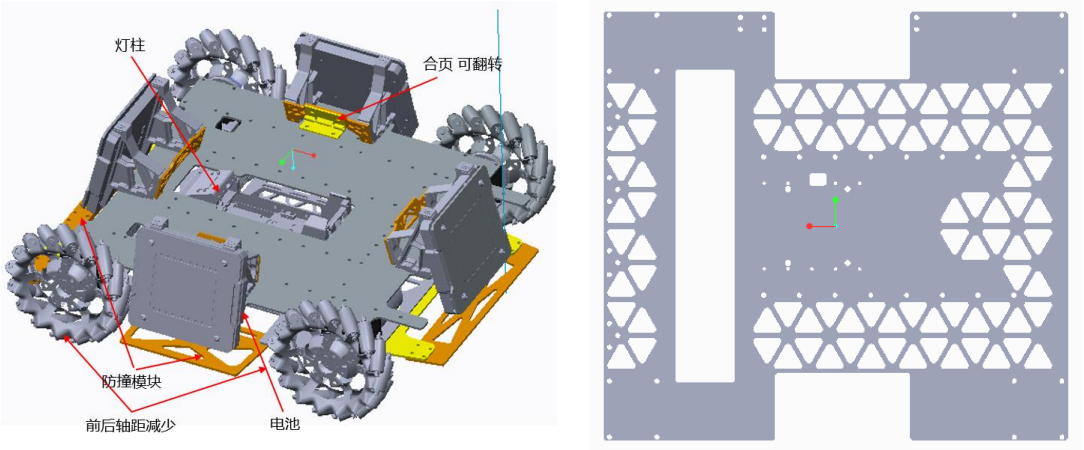


图 5

整车的上层是由铝方管构成的框架，包括升降，都是在框架的基础上搭建起来。升降采用 9mm 的滑块导轨，夹具采用 7mm 的滑块导轨。升降则选用齿距 5mm 的同步带及同步带轮。整个框架采用铆钉连接。

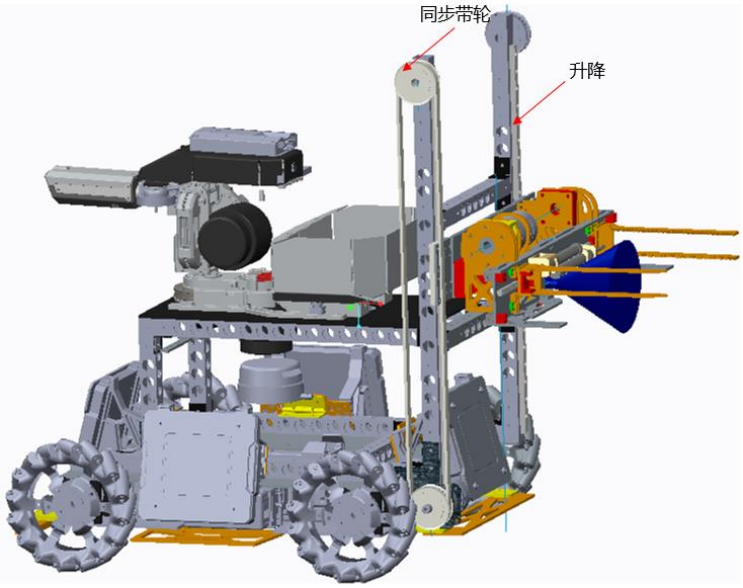


图 6

减重孔细节：本人研究生的研究是结构优化方向，所以看到有个结构都有减重的欲望。本阶段车的减重孔用了最常见也是比较有效的较重方式—交叉型。减重孔的前提是不能切断或者尽量不减弱该结构的传力路径，下面是一些结构件的减重孔，现在看看感觉还有很大的提高空间，因为这些都是凭经验设计的，由于时间紧迫，没有来得及用相关软件进行拓扑优化。

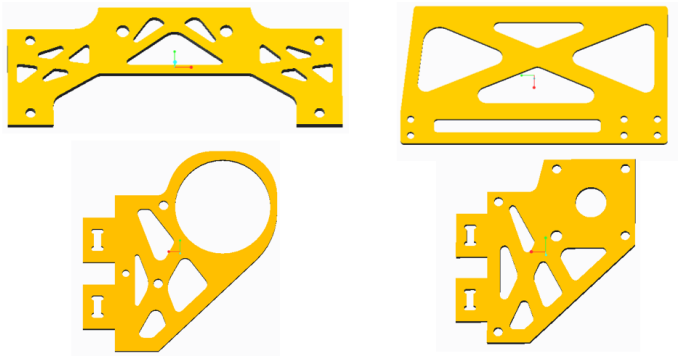


图 7

1.3 最新装配图

1.3.1 总装配图

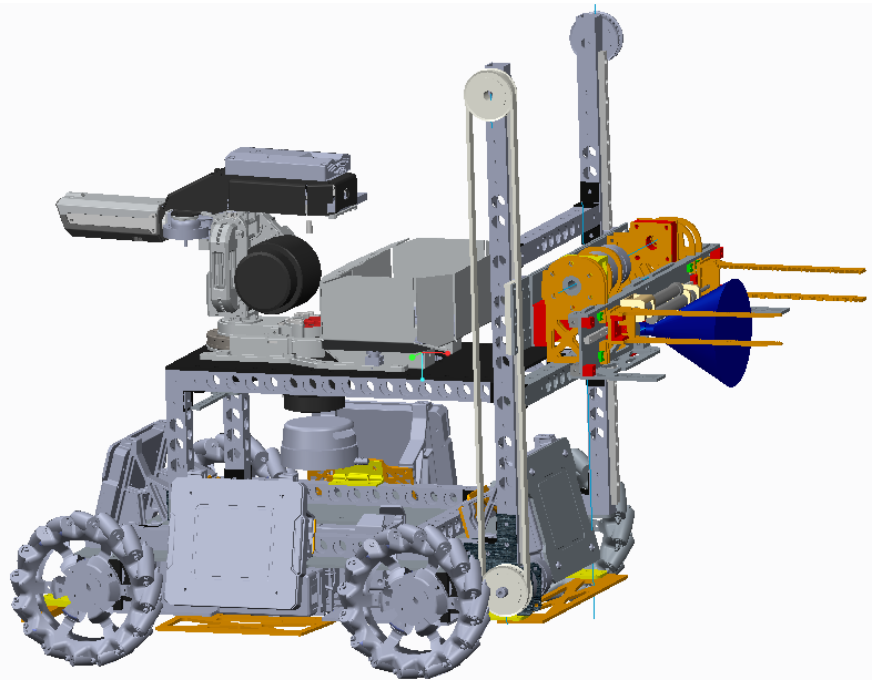


图 8

1.3.2 零件清单

由于之前一直使用 proe5.0，不能中文命名，所以在建模过程中习惯用拼音命名。

材料清单.html

1.4 还可以优化的方向

1. 底盘还是稍微有点打滑，可能和没有悬挂有些关系，还有就是升降的夹具那个模块较重，所以需要在车的前端加一些配重，由于时间关系，没有来得及加上合适的配重。
2. 很期待加上矫正弹药箱姿态模块。
3. 弹仓一直没有时间好好的做，影响整个车的颜值。

2.嵌入式部分

2,1 整体方案

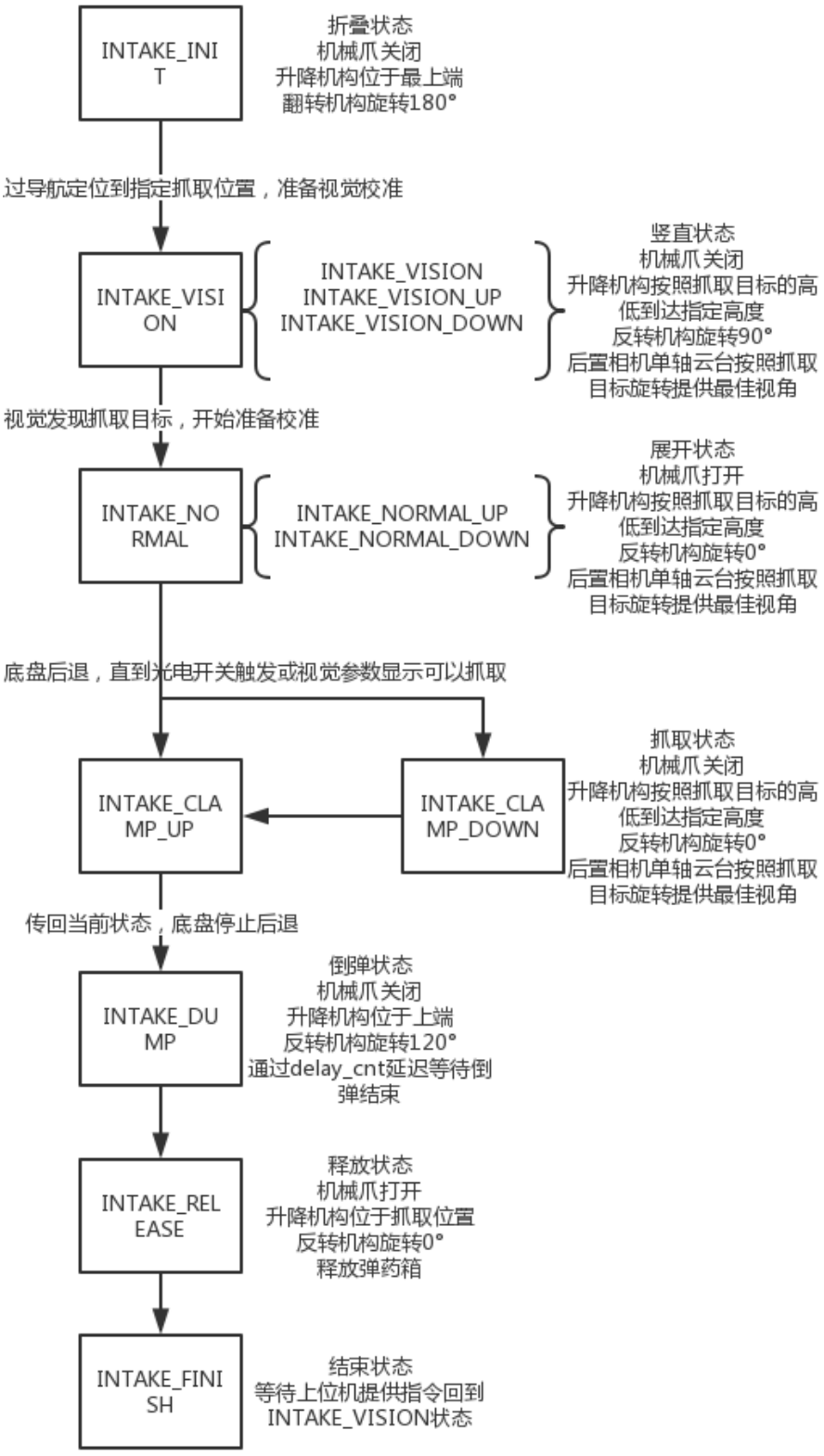
基于官方的 RoboRTS-Firmware 架构进行修改和增加，加入了 intake_task 作为抓取机构的控制，在 pc_send_mesg 及 pc_recv_mesg 中加入相应的结构体并在 ROS 加入所对应的 msg 文件作为上位机与下位机之间的通信链路。读取 judge_recv_mesg 中枪口热量的信息，并根据该信息进行枪口热量控制。修改底盘 DODGE 模式相关代码将单侧扭腰改为双侧扭腰以降低被击中概率。

2.2 抓取模块

抓取弹药箱是整个流程中的重点，也是 ICRA 比赛中没有涉及到的新功能，所以我们把嵌入式的重点放在了抓取机构的设计和控制上。

硬件方面，通过控制两个 3508 电机来控制抓取机构的升降和翻转，通过控制电磁阀来控制机械爪的打开和闭合，通过 PWM 信号控制 EMAX-ES08MD 舵机带动后置相机旋转，以确保在抓取不同位置时相机处于最佳视角状态。

软件方面，加入了 intake_task 作为抓取机构的控制，在 pc_send_mesg 中加入了 intake_info 相应的在 ROS 中加入了 IntakeInfo.msg 作为下位机向上位机提供抓取状态的通信链路，在 pc_recv_mesg 中加入 intake_control_data 相应的在 ROS 中加入 IntakeControl.msg 作为上位机向下位机提供抓取指令的通信链路。底盘控制及抓取指令由上位机视觉模块提供，抓取、升降及后置相机云台由下位机控制。控制逻辑如下图。


INTAKE_INIT
折叠状态
机械爪关闭
升降机构位于最上端
翻转机构旋转180°
过导航定位到指定抓取位置，准备视觉校准
INTAKE_VISION
INTAKE_VISION_UP
INTAKE_VISION_DOWN
竖直状态
机械爪关闭
升降机构按照抓取目标的高低到达指定高度
反转机构旋转90°
后置相机单轴云台按照抓取目标旋转提供最佳视角
视觉发现抓取目标，开始准备校准
INTAKE_NORMAL_UP
INTAKE_NORMAL_DOWN
展开状态
机械爪打开
升降机构按照抓取目标的高低到达指定高度
反转机构旋转0°
后置相机单轴云台按照抓取目标旋转提供最佳视角
底盘后退，直到光电开关触发或视觉参数显示可以抓取
INTAKE_CLAMP_UP
INTAKE_CLAMP_DOWN
抓取状态
机械爪关闭
升降机构按照抓取目标的高低到达指定高度
反转机构旋转0°
后置相机单轴云台按照抓取目标旋转提供最佳视角
传回当前状态，底盘停止后退
INTAKE_DUMP
倒弹状态
机械爪关闭
升降机构位于上端
反转机构旋转120°
通过delay_cnt延迟等待倒弹结束
INTAKE_RELEASE
释放状态
机械爪打开
升降机构位于抓取位置
反转机构旋转0°
释放弹药箱
INTAKE_FINISH
结束状态
等待上位机提供指令回到INTAKE_VISION状态

2.3 梯形曲线轨迹规划

为了更好地控制爪子升降、旋转以及底盘的运动，我们使用了梯形曲线的轨迹规划(trapezoidal motion profile)。轨迹规划有如下优点：

1. 相比于单独使用 PID 反馈控制，被动输出信号，使用轨迹规划提供了前馈速度与位置数据，使用主动方式输出信号，尽可能的减小了扰动的影响。将轨迹规划前馈和 PID 反馈结合让控制系统得到大量优化，PID 反馈只需弥补前馈位置与目标位置之间的误差。

2. 加减速度和最高速度都可以受到指令的限制。麦克拉姆轮在高加速度情况下容易打滑，会影响底盘 Odom 定位的准确性。对加减速度的控制使我们能让麦克拉姆轮在不打滑最大加减速运行，在高速度下达到 Odom 定位的准确性。加减速限制也能对爪子升降旋转机械结构做到保护。

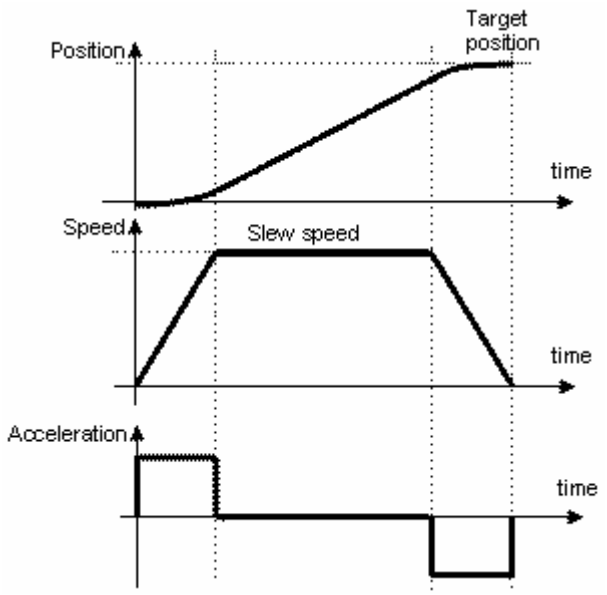
3. 对运动时间的准确把握。Motion Profile 使机械结构运动时间与位置得到了一一对应，方便在多个结构运动有相对限制的情况下规划最短运行时间，可以有效防止机械结构运动中可能的碰撞。

梯形曲线有三个阶段，

t0—t1 为加速阶段，从起点开始位置以设定的加速度运行到最高速度

t1—t2 为匀速阶段，保持最高速运行

t2—t3 为减速阶段，以设定减速度运行到最终速度为零，到达设定位置



https://www.technosoftmotion.com/ESM-um-html/index.html?tml_trapezoidal_position_profiles.htm

计算输入值为: 初始位置(pi), 最终位置(pf), 加速度(acc), 减速度(dcc), 最高速度(vmax)
算法:

1. 确定是否需要匀速运行。计算加速到最高速度立刻减速到速度为零的距离(d1)，比较距离是否大于设定距离(d2)
2. 如果 d1 大于等于 d2，则不需匀速运行 (t1 = t2) ，我们可以通过 pi, pf, acc, dcc 计算出 t1 与 t3 值
3. 如果 d1 小于 d2，则需匀速运行 (t2 > t1) ， 我们可以通过 d1,d2 差值与 vmax 计算出匀速阶段运行时间(t2 – t1)，最终算出 t1 与 t3 值

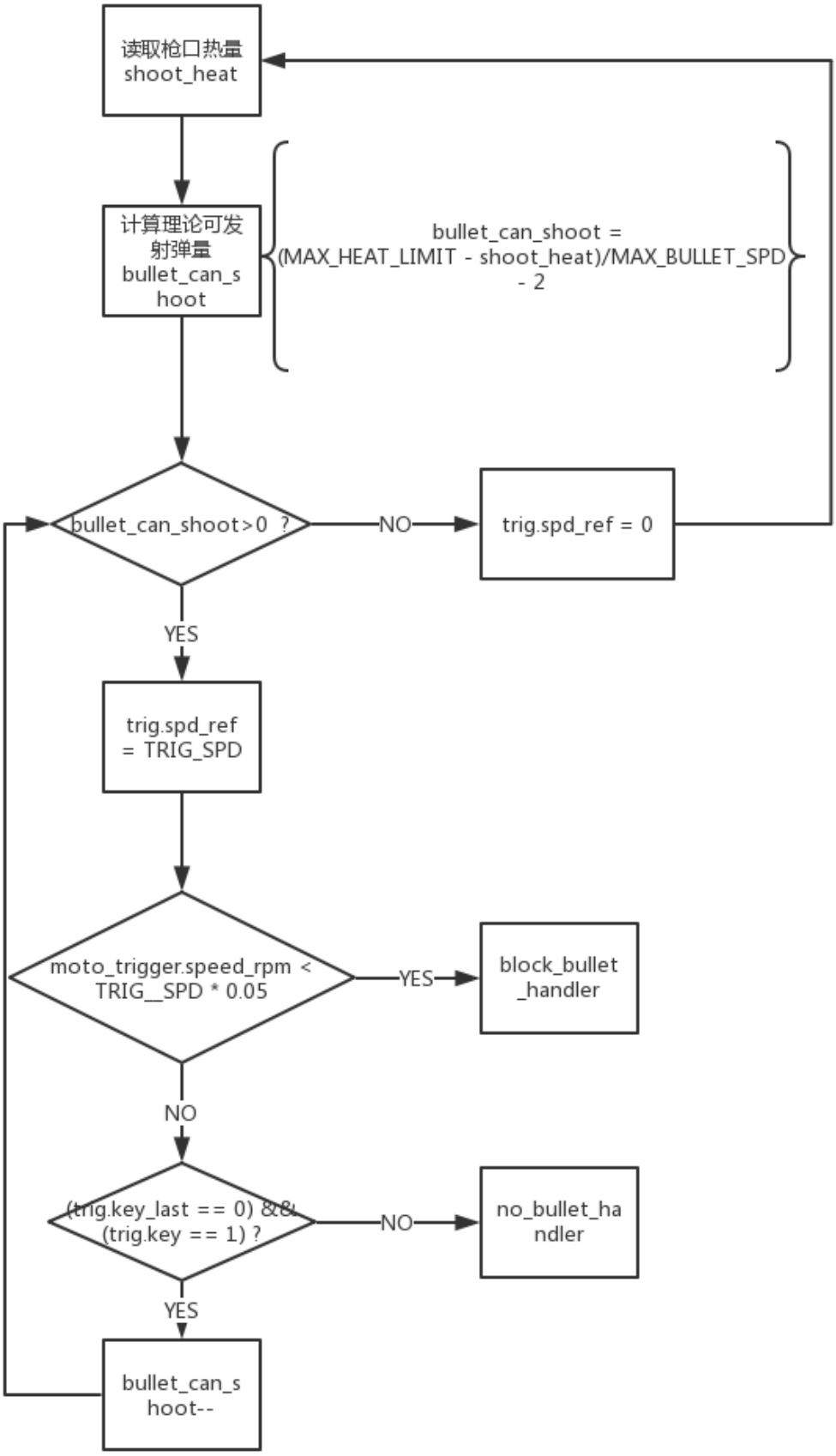
运算得到 t1, t2, t3 值后，运动时间 t 的位置和速度便可被推导出。

2.4 枪口热量模块

针对官方规则中枪口热量的限制，我们在射击模块中加入了枪口热量的控制以确保不会因为超出枪口热量而受到伤害。

由于与较高的射频相比，裁判系统始数据时性较差，又加上论坛中多人反应裁判系统热量数值延迟的问题吗，无法在线实时读取枪口热量以确认当前是否有热量进行发弹。只能采取单次读取热量后离线处理的方式，基本思路是通过读取裁判系统中枪口热量信息，结合测试时弹速的最大值得出当前可发射的子弹数量，每检测到一次发弹，当前可发射子弹数量-1，直至可发射子弹数量为 0，拨弹电

机停转，再次读取热量进行判断，同时为了处理热量数值延迟及拨弹电机控制延迟的问题，留有部分控制余量。关于如何检测发弹，最初考虑使用裁判系统的射速信息始时更新发弹数并准确离线计算热量，但效果并不理想，最终选择通过发弹开关来记录发弹量。具体控制逻辑如下图。


读取枪口热量
shoot_heat
计算理论可发射弹量
bullet_can_shoot
bullet_can_shoot =
(MAX_HEAT_LIMIT - shoot_heat)/MAX_BULLET_SPD
- 2
bullet_can_shoot>0 ?
NO
trig.spd_ref = 0
YES
trig.spd_ref
= TRIG_SPD
moto_trigger.speed_rpm <
TRIG__SPD * 0.05
YES
block_bullet
_handler
NO
(trig.key_last == 0) &&
(trig.key == 1) ?
NO
no_bullet_ha
ndler
YES
bullet_can_s
hoot--

2.5 难点与不足

1. 关于弹舱内剩余弹量的估计，以每次抓取动作作为信号，剩余弹量+50，这样可能会出现有抓取动作但没有抓到弹药箱或弹药箱中无弹的情况，导致对剩余弹量误判影响决策，本考虑在弹舱侧壁加入光电开关检测是否有弹药通过，但因时间原因未能完成。
2. 轨迹规划常见的有 S 曲线(S-curve profile)以及梯形曲线(trapezoidal profile)。对于这次比赛，我们选择了梯形曲线应为其计算的方便，占用很短时间实现突出的效果。梯形曲线的加速度并不是连贯的，S 曲线可以更有效的控制加加速.

3 算法部分

3.1 开发环境介绍

视觉部分

Sdk:　OpenCV 3.2.0
相机配置：
　　弹药箱识别相机：200W 高清 USB 摄像头模组 模块 SONY IMX322
　　链接 http://m.tb.cn/h.32i2jel

3.2 整体技术方案概述

弹药箱视觉识别

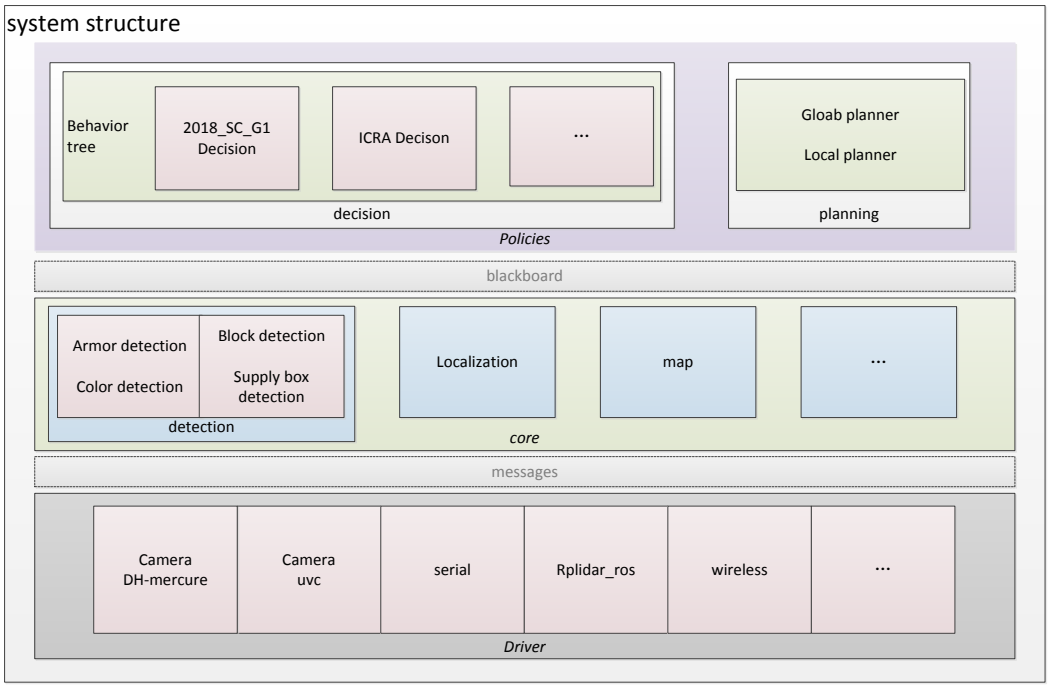
（1） 通过 YUV 通道提取和筛选出弹药箱上最稳定的特征——红色半圆弧（该弹药箱中的蓝色圆弧表现效果不好，不易提取）。
（2） 将所有 YUV 筛选出来的连通域转化为点集，用最小二乘法拟合出点集的三阶拟合曲线。
（3） 将图像中所有拟合出的曲线与特征曲线——半圆圆弧的数学参数进行对比，筛选出符合的曲线
（4） 将筛选出的符合的曲线与曲线的原图像求取标准差，筛选出最匹配的连通区域。即为真实的红色半圆弧所在区域。
（5） 事先在比赛场中用该算法进行不加特征的环境标定，求取出最低的信度筛选阈值。

行为树

在 RoboRTS 的基础上，分别将比赛行为分为:detect block; search buff ; auto supply ; offensive

3.3 算法整体框架设计

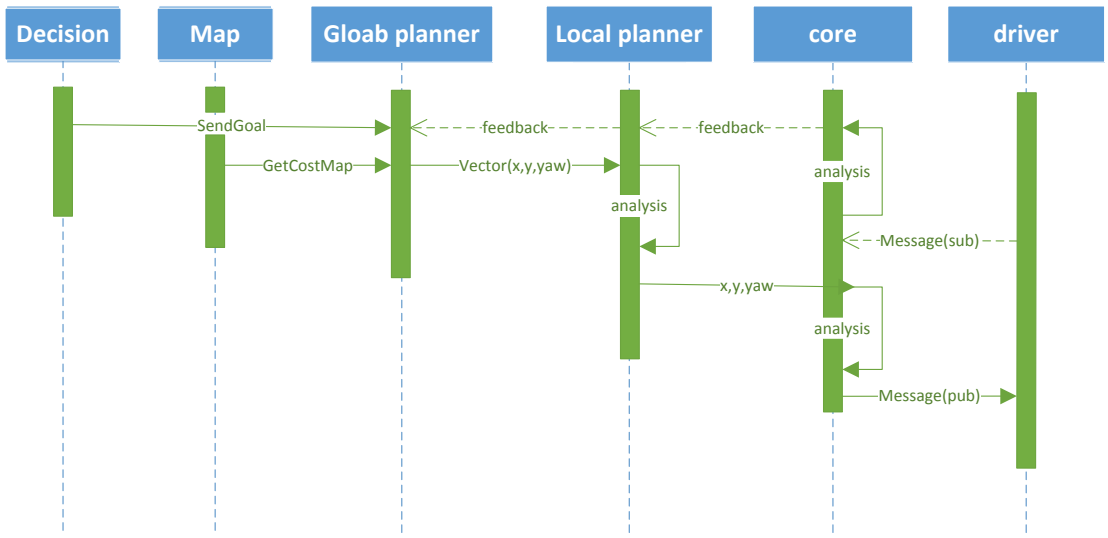
3.3.1 软件结构图



如上图所示，我们队伍将该 RTS 框架理解为如上软件结构图所示。将整个软件层次划分为三层：Driver Core Policies 层级之间双向通信。

3.3.2 系统时序图

系统时序图以到达某一个坐标点为例，如图所示。



3.4 算法功能模块说明

3. 4. 1 定位算法

3. 4. 2 导航算法

3. 4. 3 跟踪射击

3. 4. 4 弹药箱识别

3.4.4.1 弹药箱识别算法简述

（1） 通过 YUV 通道提取和筛选出弹药箱上最稳定的特征——红色半圆弧（该弹药箱中的蓝色圆弧表现效果不好，不易提取）。

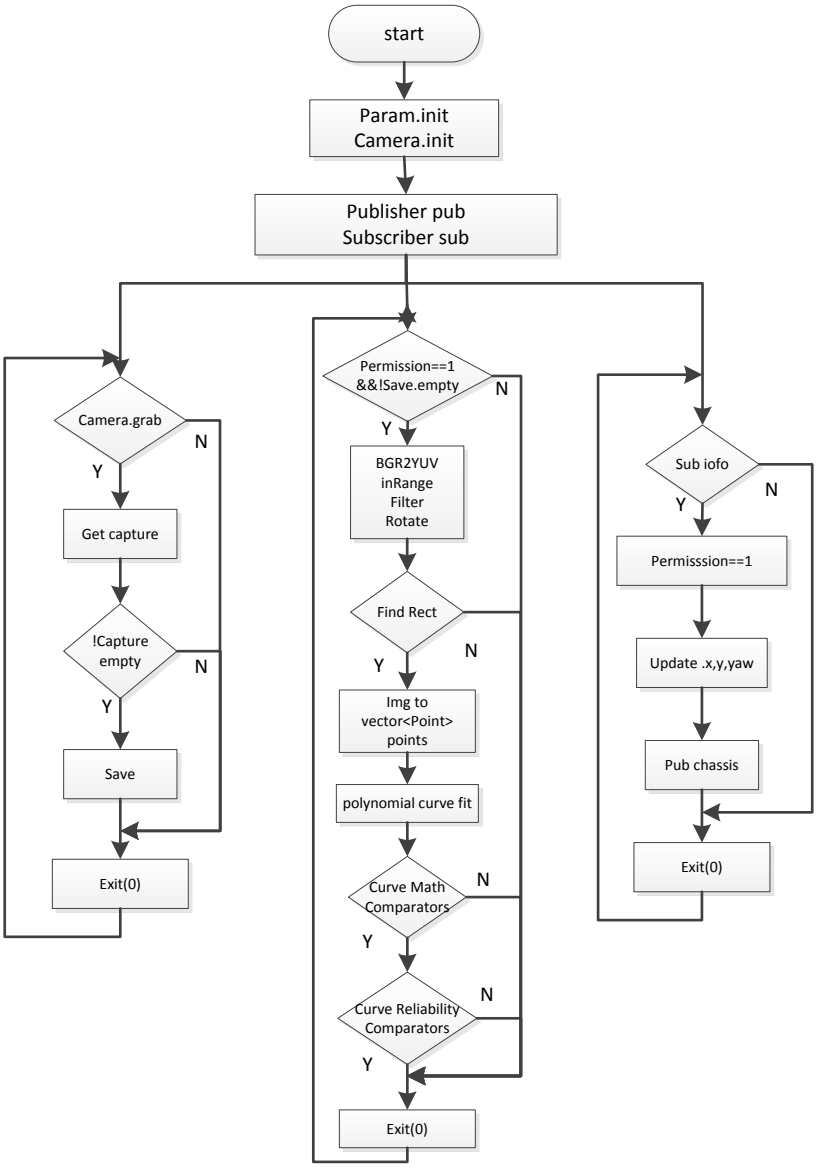
（2） 将所有 YUV 筛选出来的连通域转化为点集，用最小二乘法拟合出点集的三阶拟合曲线。

（3） 将图像中所有拟合出的曲线与特征曲线——半圆圆弧的数学参数进行对比，筛选出符合的曲线

（4） 将筛选出的符合的曲线与曲线的原图像求取标准差，筛选出最匹配的连通区域。即为真实的红色半圆弧所在区域。

（5） 事先在比赛场中用该算法进行不加特征的环境标定，求取出最低的信度筛选阈值。

其流程图如图所示：



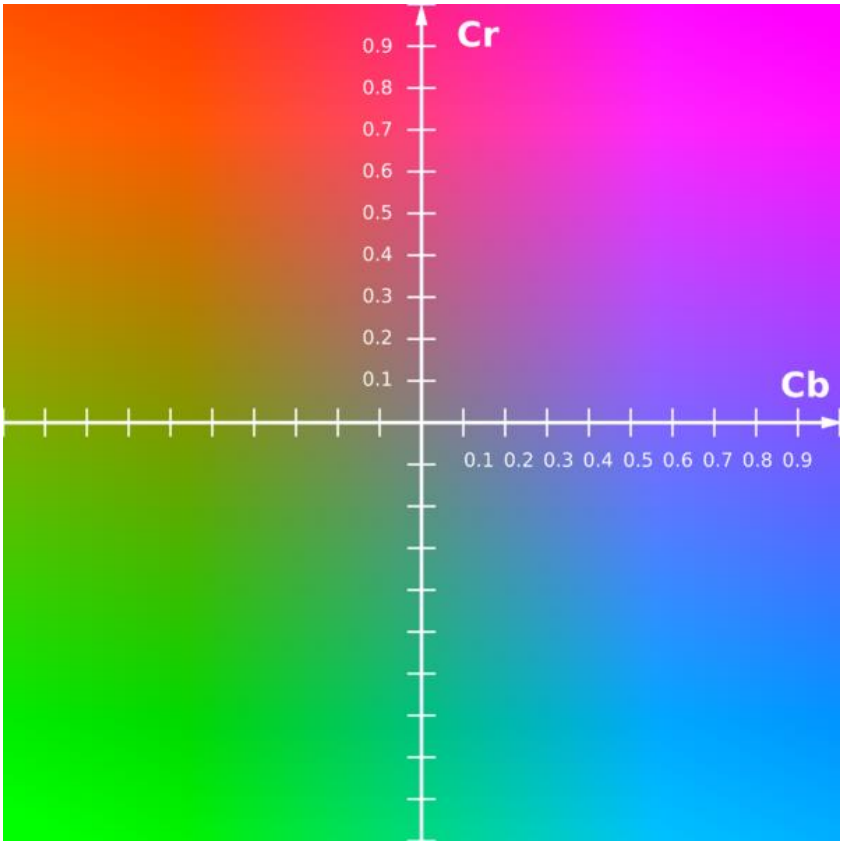
3.4.4.2 颜色预处理

（1） BGR TO YUV

YUV 是指亮度参量和色度参量分开表示的像素格式，而这样分开的好处就是不但可以避免相互干扰，还可以降低色度的采样率而不会对图像质

量影响太大。YUV 是一个比较笼统地说法，针对它的具体排列方式，可以分为很多种具体的格式。对于 YUV 格式，比较原始的讲解是 MPEG-2 VIDEO 部分的解释，当然后来微软有一个比较经典的解释，中文的大多是翻译这篇文章的。文章来源：http://msdn.microsoft.com/en-us/library/aa904813(VS.80).aspx

好了，以上都是我复制粘贴的。总的来理解，yuv 相对于 hsv 和 rgb 的好处在于，它将颜色空间的亮度单独抽取出来，置于 Y 通道，也就是亮度通道，而 u、v 通道（压缩版本为 Cb、Cr 通道）分别代表当前颜色对蓝色和红色的偏移程度。如图所示。



图

说到这里，搞过 RM 的估计都会心一笑了，这不就是 Robomaster 比赛专业颜色空间嘛？是不是 dji 搞出来的？答案当然是否定的，不过 YUV 颜色空间用在 R(red)o(or)B(blue)oMaster 的比赛赛场上再适合不过了。据可靠消息：某帝都的工业大学从 17 年起第一次使用该颜色通道在装甲板识别上，便成功的在 RM 变幻多端的赛场上打出不俗的效果，并据说该队视觉组成员到了现场看了下 debug 窗口一眼，便再没调过参数。

（2） inRange

这里的阈值分割不再多提，我个人比较喜欢用滑动条调参，到比赛现场打开 debug 窗口，用滑动条拖一下，参数就调好了。

Img before inrange

Img after inrange

（3） Filter Rotate and Find Rect

Filter 还是用我习惯的组合：
中值滤波去噪点
膨胀形成更多连通域

```
medianBlur(output, output, 3);
dilate(output, output, element55);
```

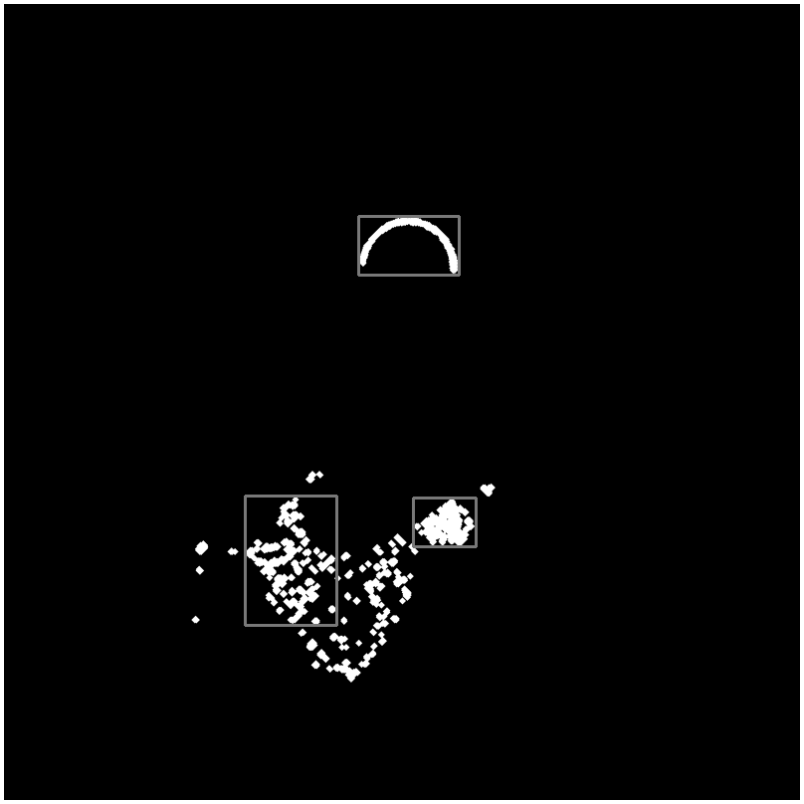
Rotate

```
//基于仿射变换的图像旋转
void    ImgRotate(const Mat &srcImg, Mat &rotatedImg, double degree)
{

    int h = srcImg.rows;
    int w = srcImg.cols;

    //求对角线的长度，做一个以对角线为边长的正方形图像
    int diaLength = int(sqrt((h*h + w*w)));
    Mat tempImg = Mat::zeros(diaLength, diaLength, srcImg.type());
    int tx = diaLength / 2 - w / 2;//原图左上角在新图上的 x 坐标
    frist_tl.x = (float)tx;
    int ty = diaLength / 2 - h / 2;//原图左上角在新图上的 y 坐标
    frist_tl.y = (float)ty;
    srcImg.copyTo(tempImg(Range(ty, ty + h), Range(tx, tx + w)));//把原图先复制到新的临时
图上。

                                                                    //以新的临时图
的中心点为旋转点
    Point rotatepoint;
    rotatepoint.x = rotatepoint.y = diaLength / 2;
    Mat rotaMat = getRotationMatrix2D(rotatepoint, degree, 1); //  获取二维旋转的仿射变换
矩阵
    warpAffine(tempImg, rotatedImg, rotaMat, Size(diaLength, diaLength));//进行仿射变换。
    return;
}
```

FindRect

```
//build the rect for all connected region
            std::vector< std::vector<Point> >    contours;
            std::vector<Vec4i> hierarchy;
            findContours(rotate_output, contours, hierarchy, RETR_EXTERNAL,
        CHAIN_APPROX_SIMPLE, Point(0, 0));

            std::vector<std::vector<Point> > conPoint(contours.size());
            std::vector<Rect> boundRect(contours.size());
            int number = 0;
            for (int i = 0;i < contours.size();i++)
```

```
{

        approxPolyDP(Mat(contours[i]), conPoint[i], 3, true);

        boundRect[i - number] = boundingRect(Mat(conPoint[i]));
        float rect_long_0 = 0, rect_wide_0 = 0, rect_shape_0 = 0;
        Point rect_tl_0 = boundRect[i - number].tl();
        Point rect_br_0 = boundRect[i - number].br();
        rect_wide_0 = (float)abs(rect_br_0.x - rect_tl_0.x);
        rect_long_0 = (float)abs(rect_br_0.y - rect_tl_0.y);
        rect_shape_0 = rect_wide_0/rect_long_0;// wide /high
        //std::cout<<rect_shape_0<<std::endl;
        //select the right size
        if ((rect_wide_0 < RECT_WIDE_MIN) || (rect_long_0 < RECT_HEIGHT_MIN) ||
(rect_shape_0 >RECT_SHAPE_MAX) || (rect_shape_0 <RECT_SHAPE_MIN))
        {
            boundRect.pop_back();
            number++;
        }
        else
        {
                cv::rectangle(rotate_output, rect_tl_0, rect_br_0, Scalar(120), 2,
2, 0);//draw the rect        }
  }
```



3.4.4.3 拟合曲线

将连通域中的点集输入，并拟合为高阶多项式曲线。

```
bool polynomial_curve_fit(std::vector<cv::Point>& key_point, int n, cv::Mat& A)
{
    //Number of key points
    int N = key_point.size();

    //构造矩阵 X
    cv::Mat X = cv::Mat::zeros(n + 1, n + 1, CV_64FC1);
    for (int i = 0; i < n + 1; i++)
    {
        for (int j = 0; j < n + 1; j++)
        {
            for (int k = 0; k < N; k++)
            {
                X.at<double>(i, j) = X.at<double>(i, j) +
                    std::pow(key_point[k].x, i + j);
            }
        }
    }

    //构造矩阵 Y
    cv::Mat Y = cv::Mat::zeros(n + 1, 1, CV_64FC1);
    for (int i = 0; i < n + 1; i++)
    {
        for (int k = 0; k < N; k++)
        {
            Y.at<double>(i, 0) = Y.at<double>(i, 0) +
                std::pow(key_point[k].x, i) * key_point[k].y;
        }
    }

    A = cv::Mat::zeros(n + 1, 1, CV_64FC1);
    //求解矩阵 A
    cv::solve(X, Y, A, cv::DECOMP_LU);
    return true;
}
```

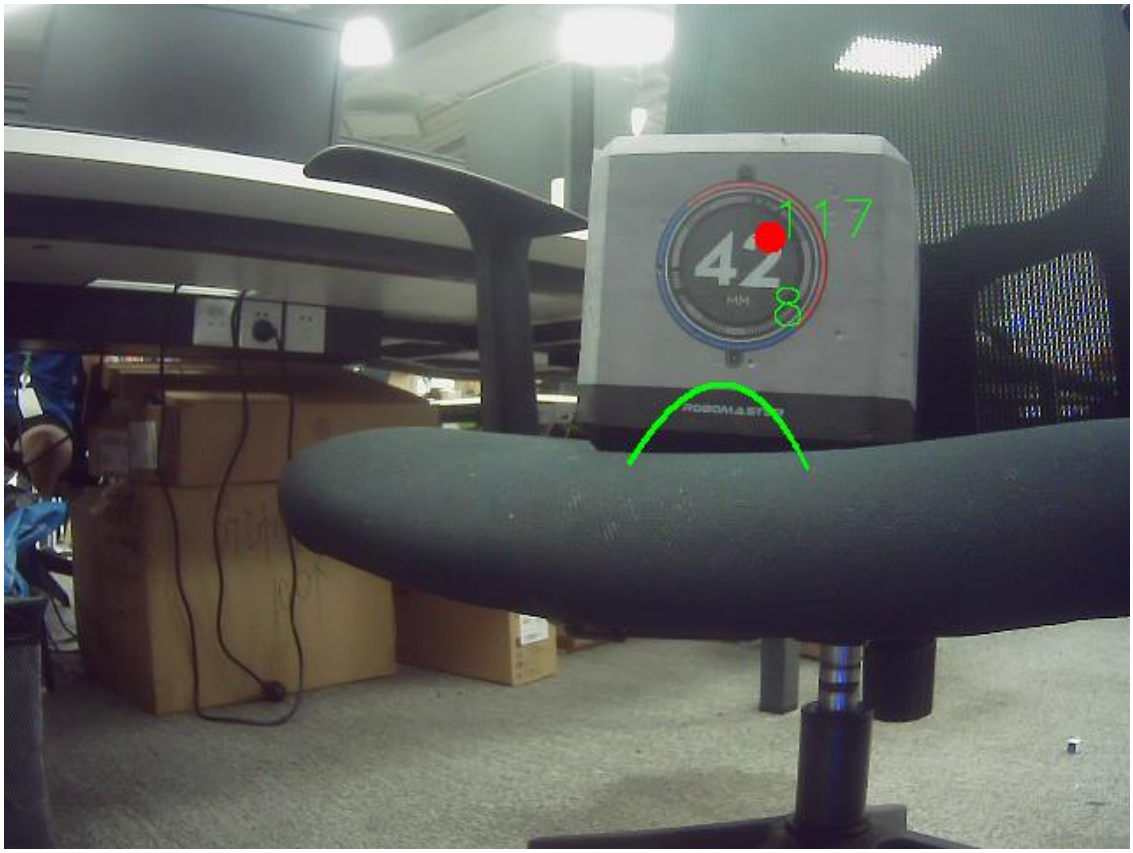
3.4.4.4 曲线数学关系比较器

求出了 A 矩阵，就相当于知道了三阶多项式各项系数，经过比较这些系数的

关系，便可以求解出最接近于半圆弧的曲线。

这里由于比赛时间紧张，选择提取了半圆弧最鲜明的特征，对曲线左半部求导，导数均大于 0；对曲线右半部取导，导数均小于 0。实时证明，只有图像旋转 45°之后拟合曲线为抛物线的连通域，才符合该项特征，因此，该方法简单有效。而后，又加上了曲线左右半部近似对称的数学关系。几乎可以筛除所有特征外的连通域。

用 cv 函数 polylines 将筛选出的曲线绘制在原图像上，由于原图像和处理图像存在围绕中心点的 45°夹角，固绘制在原图的曲线并不与原红色圆弧区域重合。但为了方便 Debug，依旧选择将其绘制在原图上，三阶多项式曲线如图弹药箱正下方绿色曲线所示。



3.4.4.5 信度比较器

但难免会有一些干扰项也能够拟合出类似的 45°旋转后的抛物线。事实正面，只要图像中存在完全对称的类似的点集便可拟合出该种抛物线。因此还要对其图像拟合的信度进行取优。将原图像最接近于该拟合曲线的区域视为输出值。

关于拟合信度，选用了标准差作为判断依据，将原图像所有点集于拟合曲线作差，并求取差值于连通域的高的比例进行累加，累加的结果再除于连通域的宽。以该累加结果作为信度，数值越低，说明原图像连通域于拟合曲线重合度越高。

这里为了方便 Debug 将该信度用 cv 的函数 cvPutText 直接显示在原图像的输出坐标点上。如上图所示数值 8 所示。

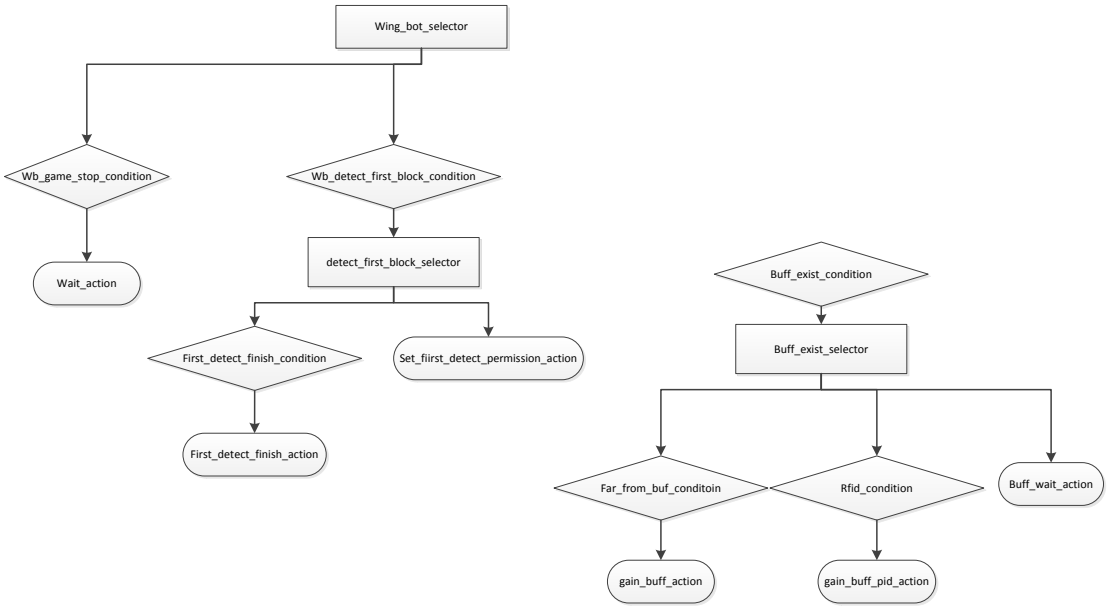
3.4.4.6 环境标定（todo）

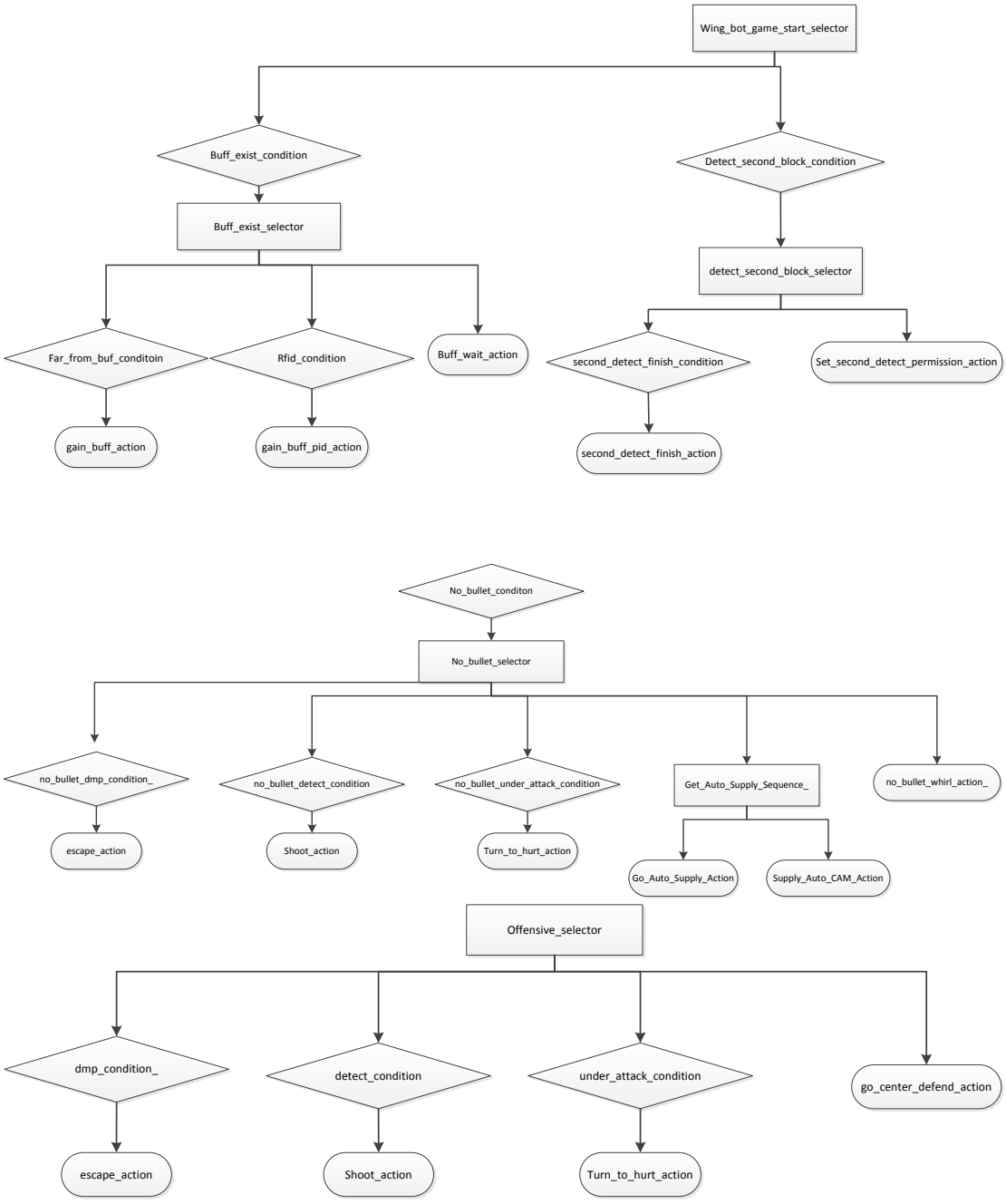
这算是一个比较有效的阈值选取方法吧，事先在比赛前用自己的算法在未加特征的环境中进行标定，算法会跑出一个在该环境中最低匹配的信度（因为该算法中的信度值是越低越好）。之后在比赛过程中，便以该阈值稍微低一点点的参数作为算法的阈值。这样就基本上自己的算法会高枕无忧了。由于本次比赛中的环境比较好，在实际测试过程中，并没有任何干扰项能满足曲线数学关系比较和曲线信度比较的条件。固没有采用该方法作为阈值设定的标准。这里提出该方法，以供为读者提供思路。

3. 4. 5 单兵逻辑

3.4.5.1 单兵逻辑行为树

行为树有点宽，因此只能用片段的方法展示。
主要是围绕四个主要行为去做的 detect block; search buff ; auto supply ; offensive





3.4.5.2 识别障碍块

首先感谢帆神提供的开源，能让我们在识别障碍块的任务上简单高效。其次，为什么要识别障碍块，因为，据我所知，有的队伍没有去识别障碍块依旧能够正常比赛。

在这里，我谈我们队伍的两点理由：

（1） 需要知道障碍块的位置来判断哪些弹药箱的抓取位置是不可取的。障碍块的存在往往会占据 1~3 个弹药箱抓取的位置，我们需要在弹药箱管理系统

中将这些不可达位置可擦除。

（2） 在路径规划中，gloab planner 与 local planner 生成的路径往往不是完全重合的，甚至有可能完全相反，这两者的矛盾点在于，gloab planner 不会永久记忆未事先在地图上标记的障碍物，因此 gloab planner 可能会和 local planner 形成一个死循环。简单来说,gloab planner 认为第一条路可以过，因为它信息来源于初始地图，当机器人听从 gloab planner 的路径并自动导航的时候，发现这条路上有一个随机障碍物，且不可通过。因此它会立刻折返选择第二条路。当 local planner 根据自己看到的障碍物导航到远离随机障碍物时，又再次听从 gloab planner 的建议认为第一条路比较近，从而形成导航死循环。

因此，机器人需要在开场的时候便知道本次比赛中的随机障碍物生成在地图的哪些地方，并重新绘制一遍地图。

此处代码由于主要算法在帆神的代码上，便不在此处贴出。

3.4.5.3“抢”buff

由于 buff 需要连续激活 5 秒的时间才可以真正触发，比赛中两支队伍到达 buff 的时间往往只相差一两秒，如果当机器人发现对方已经提前抢占了 buff 激活区，机器人是选择撤退还是选择“抢”buff 便显得至关重要。

显然，我们选择了后者。然而如果用 sendGoal 去让机器人去 buff 点的时候显然是会自动避开对方机器人的，这便不可能实现“抢”buff 这个核心战略点。因此我们在到达 buff 半径一米范围内选择无视路径规划，通过反馈的机器人的 pose，再根据输入目标点的位置，通过 PID 自动抢占 buff，并将 buff 点上的机器人挤出去。

更新自身位置的算法：

```
//call this function to update the real-time pose of the robot TODO::version for wingbot
void updateMyCurrentPose(){
    tf::StampedTransform transform;
    try{
        master_tf_listener_->lookupTransform("/map", "/base_link", ros::Time(0), transform);
    }
    catch (tf::TransformException ex){
        LOG_ERROR << __FUNCTION__ << ex.what();
        ros::Duration(1.0).sleep();
    }

    my_current_pose.header.frame_id = "map";
    my_current_pose.header.stamp = ros::Time::now();
    my_current_pose.pose.position.x = transform.getOrigin().x();
    my_current_pose.pose.position.y = transform.getOrigin().y();
    tf::quaternionTFToMsg(transform.getRotation(), my_current_pose.pose.orientation);
}
```

判断是否距离 buff 点一米半径的算法：

```
bool GetTheFarFromBuffStatus()
{
    updateMyCurrentPose();
    double temp=sqrt(pow(my_current_pose.pose.position.x-4.15,2)+pow(my_current_pose.pose.position.y-2.85,2));
    if(temp>1)return true;
    else return false;

}
```

通过 PID 控制获得 buff 的代码：

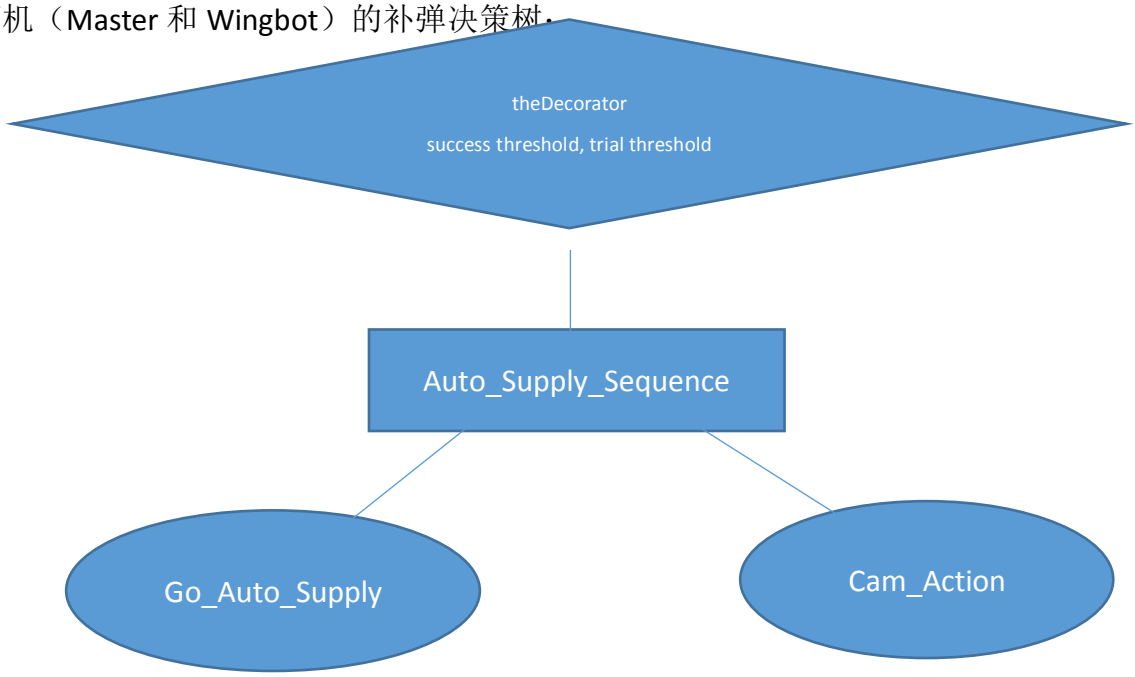
```
void GainBuffPidControl(geometry_msgs::PoseStamped tar_pose)
{
    float x_temp,y_temp,yaw_temp;
    geometry_msgs::Twist buff_chassis_;
    geometry_msgs::PoseStamped buff_pose,standard_pose;
    updateMyCurrentPose();
    x_temp=pid_buff_x(tar_pose.pose.position.x,my_current_pose.pose.position.x);
    y_temp=pid_buff_y(tar_pose.pose.position.y,my_current_pose.pose.position.y);

    buff_pose.pose.orientation.w = my_current_pose.pose.orientation.w;
    buff_pose.pose.orientation.x = my_current_pose.pose.orientation.x;
    buff_pose.pose.orientation.y = my_current_pose.pose.orientation.y;
    buff_pose.pose.orientation.z = my_current_pose.pose.orientation.z;
    standard_pose.pose.orientation.w=1;
    standard_pose.pose.orientation.x=0;
    standard_pose.pose.orientation.y=0;
    standard_pose.pose.orientation.z=0;
    yaw_temp=GetAngle(buff_pose,standard_pose);
    LOG_INFO<<"yaw_temp==";
    LOG_INFO<<yaw_temp;
    LOG_INFO<<my_current_pose;
   // poseStampedMsgToTF(camera_pose_msg, buff_pose);
    buff_chassis_.linear.x=(float)(x_temp*cos((double)yaw_temp)-y_temp
    *sin((double)yaw_temp));

    buff_chassis_.linear.y=(float)(x_temp*sin((double)yaw_temp)+y_temp*cos((double)ya
    w_temp));
    LOG_INFO<<"outchassis==";
    LOG_INFO<<buff_chassis_;
    LOG_INFO<<"pidout==";
    LOG_INFO<<x_temp;
    LOG_INFO<<y_temp;
    chassis_cantrol_pub_.publish(buff_chassis_);
```

}

3.4.5.4Auto Supply

根据自身坐标于弹药箱识别位置坐标的距离，来自动选择弹药箱补给顺序，并具备一套弹药箱管理系统，实时更新弹药箱的状态，并于同队机器人共享。根据自身坐标于弹药箱识别位置坐标的距离，来自动选择弹药箱补给顺序，并具备一套弹药箱管理系统，实时更新弹药箱的状态，并于同队机器人共享。

两机（Master 和 Wingbot）的补弹决策树：



进入补弹树之后，theDecorator 负责决定何时结束补弹。theDecorator 记录 Auto_Supply_Sequnce 返回的 BehaviorState，当成功次数达到 Success Threshold 时返回 BehaviorState::SUCCESS，表示成功补弹 n 箱，n 为设置好的 Success Threshold。如果 Sequence 返回总次数达到 Trial Threshold，返回 BehaviorState:: FAILURE，表示没能完成补弹 n 箱。

Go_Auto_Supply 负责判断前往哪个弹药箱位置补弹。每个弹药箱有四种状态（0=unknown，1=not available， 2=available， 3=taken）。只有在弹药箱状态为 unknown 或者 available 时会选择。在这些弹药箱中，Go_Auto_Supply 计算直线距离（机器人当前位置通过读取 map 到 base_link 的 tf， 弹药箱位置用预先设置的取弹位姿），并前往“最近”的弹药箱。因为计算的是直线距离而不是 a star algorithm 计算出的距离，取弹的决策不是最优的。

到达取弹位姿之后 Go_Auto_Supply 返回 SUCCESS，进入 Cam_Action。Cam_Action 发布一个 permission flag 给 vision_demo 节点。操纵电机靠近弹药箱，控制爪子交由 vision_demo 负责。Vision_demo 返回抓取是否成功，由 Cam_Action 更新本机的弹药箱状态，并且发布/shared_fromwing（frommaster）/CamFeedBack 通知另一台机器。

3.4.6 多兵作战

两机（Master 和 Wingbot）通讯使用了 ROS Multimaster 下的 master_discovery 和 master_sync 节点。两机在本组搭建的私服上通信。Master_discovery 自动检索网络中一定频段的机器。Master_sync 同步特定的 ros topic 和 service。

使用到两机通讯的模块：弹药箱管理、障碍块管理、地图更新、敌人信息。

弹药箱管理：每当 Wingbot 或 Master 更新一个弹药箱点的状态（视觉未发现弹药箱、视觉识别到弹药箱但抓取失败、成功抓取），会通过 shared_frommaster/CamFeedBack 与 shared_fromwing/CamFeedBack 发布一个弹药箱信息。另外一台机器的 blackboard.h 文件中的 Subscriber 和回调函数会根据此信息更新弹药数据，以便后续补弹的决策。理论上，任何一个弹药箱位置都不会被搜索一遍以上，并且两机当前目标不会为同一个弹药箱。本组的战术是由 Wingbot 夺取 Buff，抢占场地中间的优势位置并补弹。由 Master 负责己方家里的弹药，成功取到两箱之后到场地中间伏击敌人。理论上，两机弹药箱同步管理可以极大的提高补弹的速度和成功率，使两机尽可能早的抵达埋伏位置，抢占先机。

障碍块管理：探测障碍块由 Wingbot 负责。在准备时间和启动区域，Wingbot 通过视觉识别检测 3、4、5、6 号位置是否有障碍箱，并将障碍块信息发送给 Master。如果 3 号位置不存在障碍块，在夺取完 Buff 之后（不论是否成功）Wingbot 会移动到指定的第二探测位置检测 1 号与 2 号位置，并将信息同步给 Master。获得障碍块位置之后两机自动遍历预先设置好的取弹位姿，并删除一部分危险（卡在障碍块中或者离障碍太近，例如七号弹药箱的上方和下方）的位姿。

地图更新：如上，Wingbot 在检测到障碍块之后，通过发布事先做好的带有障碍块的地图到/master/map 与/wing/map 上，更新当前地图。由于地图文件比较大，网络条件差时地图更新需要数秒时间。除此以外，地图更新避免了 Global Planner 与 Local Planner 矛盾导致机器人卡入死循环的 Bug。
敌人信息：使用的 ICRA 官方代码中的两机协作机制。

3.4.7 控制

不同模块的控制主要基于 ros 的 sub 与 pub 功能

Messages::
ClampState
ClampControl

```
enum class ClampMode{
        CLAMP_MODE_NONE = -1,
        CLAMP_MODE_INIT = 0,
        CLAMP_MODE_VISION = 1,
        CLAMP_MODE_WORK = 2,
        CLAMP_MODE_OFFSET = 3,
        CLAMP_MODE_CATCH = 4,
        CLAMP_MODE_DROP = 5,
        CLAMP_MODE_KEEP = 6,
        CLAMP_MODE_RELEASE = 7,
};

enum class ClampPlace{
        CLAMP_PLACE_NONE = -1,
        CLAMP_PLACE_CENTER = 0,
        CLAMP_PLACE_UP = 1,
        CLAMP_PLACE_DOWN = 2,
};

enum class ClampFeedback{
        CLAMP_FB_NONE =0,
        CLAMP_FB_GET =1,
        CLAMP_FB_SUPPLY =2,
        CLAMP_FB_KEEP =3,
        CLAMP_FB_END =4,
};
```

Messages::
BlockInfo
BlockControl

```
class BlockInfo{
public:
        int Permisiion;
};
class BlockFeedback{
public:
```

```
        int state1;
        int state2;
    };
    class CamStartSearching{
  public:
        int Permission;
    };
```


```
enum class CamFeedback{
    CATCHSTATE_BEGIN_SEARCH = -1,
    CATCHSTATE_NORMAL= 0,
    CATCHSTATE_NONETARGET =1,
    CATCHSTATE_LOSETARGET= 2,
    CATCHSTATE_ARRIVETARGET =3,
     };
```

3.5 测试结果

弹药箱识别 稳定
弹药箱补给控制 稳定
弹药箱管理系统 稳定
弹药箱共享系统 测试稳定，实战效果不明显
障碍块识别 稳定
障碍块在地图更新 稳定
地图共享 测试时稳定，实战较稳定（因网络问题会丢失数据）
“抢”buff 测试时有隐患（机器人 pose 因撞击而跑飞），比赛稳定
行为树 wing bot 稳定
master 不稳定
多兵作战 由于 master 机器人在比赛中不稳定，所有多兵作战只在比赛中出现过一次效果

3.6 可优化方案

（1） 弹药箱实时监测补给
我们采用的策略是到达补给点，再开启视觉检测，这样带来的问题是，去的补给点不一定有弹药箱，这样在比赛中是十分浪费时间的，观赏效果也

不好。

本算法中的弹药箱识别最远距离可达 300cm,稳定距离为 5cm~200cm。因此完全可以让机器人在运动中事实检测弹药箱的信息。

然而这需要机器人的行走姿态和路径的配合，为了稳定的完成比赛，我们放弃了该方案。

（2） 多兵作战

如何更好的实现多兵作战一直是这个比赛的难题。然而这需要两台机器人都十分稳定的大前提。我们组在本次夏令营中设置了简单的多兵作战方案，但由于有一台机器人并不十分稳定，该方案没有很好的得到执行，因此也没有继续往后研究。

4.夏令营感想、总结

RoboMaster 大赛组委会

邮箱：robomaster@dji.com

官方论坛：http://bbs.robomaster.com

官方网站：http://www.robomasters.com

电话：075536383255（周一至周五 10:00-19:00）

地址：广东省深圳市南山区西丽镇茶光路 1089 号集成电路设计应用产业园 2 楼 202

微信

微博