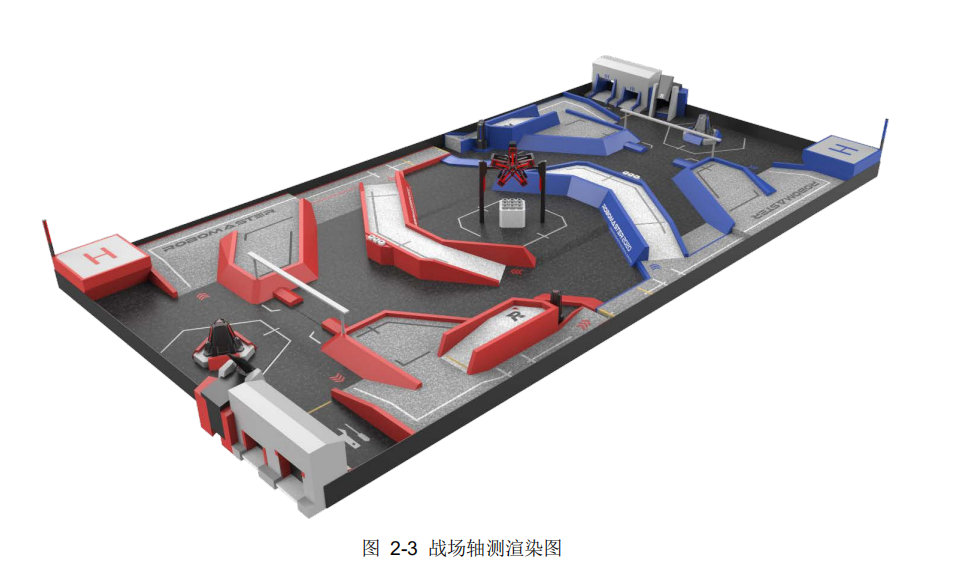
# 能量机关模块



规则1.0

红蓝双方能量机关共同旋转，即红方能量机关顺时针旋转时，蓝方能量机关同时逆时针旋转（旋转方向 以面朝该方能量机关时的旋转方向进行确定）。比赛过程中，小能量机关的转速为 10RPM，大能量机关以特定策略运动，具体方案后续将修改更新。

2019.12.31规则更新2.0

红蓝双方能量机关共轴旋转，即红方能量机关顺时针旋转时，蓝方能量机关相应地逆时针旋转（旋转方向 以面朝该方能量机关时的旋转方向进行确定）。每局比赛开始前，能量机关旋转方向随机。比赛中，能量机关旋转方向保持一致。

小能量机关的转速固定为 10RPM。大能量机关转速按照三角函数呈周期变化。速度目标函数为：spd = 0.785 ∗ sin (1.884 ∗ t) + 1.305，其中 spd 的单位为 rad/s，t 的单位为 s。能量机关转速的跟踪精度后续更新。

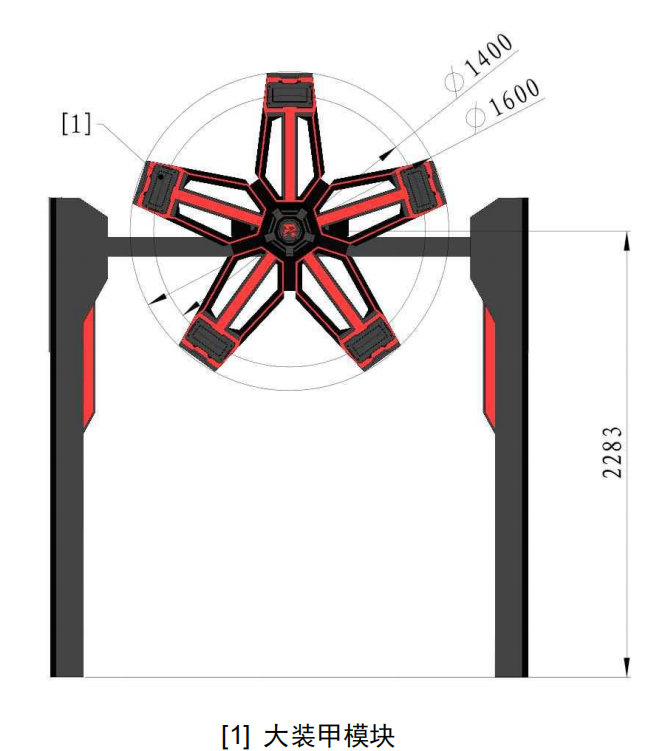
射击过程中，若出现以下任意一种情况，则此次激活失败，能量机关将恢复至可激活状态。激活失败的情况：

 未能在 2.5 秒内击中随机点亮的装甲模块

 击中非随机点亮的装甲模块

能量机关在每次开始旋转时，其旋转方向随机确定。





# 1.图像处理

## 1.1.思路

1. 图像处理

用到的主要方法：腐蚀膨胀、漫水填充

1. 轮廓查找

在二值图中找到矩形

1. 找到打击目标

根据矩形的长宽比与面积来寻找已经点亮的扇叶

1. 绘制打击中心

绘制出当前需要击打的扇叶部位

## 1.2参数说明

1. area = contourArea(contours[i])

(area < 50 || 1e4 < area)

1. aim = rrect.size.height / rrect.size.width

(aim > 1.7 && aim < 2.6)

# 2.预测算法

## 2.1卡尔曼滤波器√

KalmanFilter(int dynamParams, int measureParams, int controlParams=0, int type=CV\_32F); 测量值measureParams需自己定义

算法的核心思想是，根据当前的"测量值" 与上一刻的 "预测量" 和 "误差"，计算得到当前的最优量，再预测下一时刻的量。把误差纳入计算，分为预测误差和测量误差，且误差独立存在， 始终不受测量数据的影响。

优势：适合运用在连续变化的实时系统中，整个算法占用的系统内存比较小，计算速度较快。

KalmanFilter KF(stateNum, measureNum, 0);

Mat processNoise(stateNum, 1, CV\_32F);

Mat measurement = Mat::zeros(measureNum, 1, CV\_32F);

KF.transitionMatrix = (Mat\_<float>(stateNum, stateNum) << 1, 0, 1, 0,

0, 1, 0, 1,

0, 0, 1, 0,

0, 0, 0, 1);

setIdentity(KF.measurementMatrix);

setIdentity(KF.processNoiseCov, Scalar::all(1e-5));

setIdentity(KF.measurementNoiseCov, Scalar::all(1e-1));

setIdentity(KF.errorCovPost, Scalar::all(1));

randn(KF.statePost, Scalar::all(0), Scalar::all(0.1));

## 2.2最小二乘法拟合圆×

通过最小化误差的平方和找到一组数据的最佳函数匹配，在计算角度变化时会受到旋转周期和初始速度的影响，暂不清楚运行效果，小能量机关可以考虑，大能量机关还有待研究。

Point cc;

float R;

circle(binary, Point(rrect.center.x, rrect.center.y), 1, Scalar(0, 0, 255), 1);

CircleInfo2(cirV, cc, R);

circle(image, cc , 1, Scalar(255, 0, 0), 2);

cirV.erase(cirV.begin());

if (cc.x != 0 && cc.y != 0) {

Mat rot\_mat = getRotationMatrix2D(cc, 30, 1);

float sinA = rot\_mat.at<double>(0, 1);

float cosA = rot\_mat.at<double>(0, 0);

float xx = -(cc.x - rrect.center.x);

float yy = -(cc.y - rrect.center.y);

Point2f resPoint = Point2f(cc.x + cosA \* xx - sinA \* yy, cc.y + sinA \* xx + cosA \* yy);

circle(image, resPoint, 1, Scalar(0, 255, 0), 10);

}

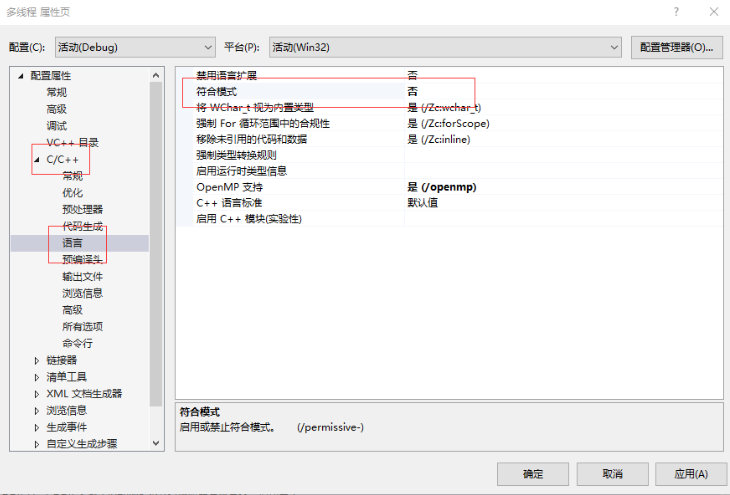
# Omp加速

## 3.1在VS中启用OpenMP

原则上，在项目上右键->属性->配置属性->C/C++->语言->OpenMP支持，选择“是”即可。操作过程中出现以下错误：

错误 C2338 C++/CLI、C++/CX 或 OpenMP 不支持两阶段名称查找；请使用 /Zc:twoPhase- 多线程 C:\Users\tonyson\_in\_the\_rain\source\repos\多线程\多线程\c1xx 1

解决方法：在属性菜单中找到C/C++ --> 语言 -->符合模式下拉菜单中选择"否"。



## 3.2添加编译器指令

（1）产生一个并行区域

（2）划分线程中的代码块

（3）在线程之间分配循环迭代

（4）序列化代码段

（5）同步线程间的工作

## 3.3优化结果

| **成对样本统计量** | | | | | |
| --- | --- | --- | --- | --- | --- |
|  |  | 均值 | N | 标准差 | 均值的标准误 |
| 对 1 | btime | 48.4919 | 20 | 3.63637 | 0.81312 |
| omp | 47.6977 | 20 | 2.94898 | 0.65941 |

| **成对样本相关系数** | | | | |
| --- | --- | --- | --- | --- |
|  |  | N | 相关系数 | Sig. |
| 对 1 | btime & omp | 20 | 0.306 | 0.190 |

**成对样本检验**

|  | 成对差分 | | | | | t | df | Sig.(双侧) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 均值 | 标准差 | 均值的标准误 | 差分的 95% 置信区间 | |
|  | 下限 | 上限 |
| btime - omp | 0.79420 | 3.91995 | 0.87653 | -1.04039 | 2.62879 | 0.906 | 19 | 0.376 |

P>0.05，两组数据无显著性差异，说明使用omp加速无明显作用。

# 误差分析（来源于网络）

## 4.1相机标定误差

由于涉及视觉测量，相机参数在进行视觉测量中，具有重要的作用，其误差会直接导致视觉测量误差。所以，需要减小其相机标定误差。多次标定，去掉偏差较大的标定结果，取中位标定结果来减小误差。

## 4.2相机安装位置误差

由于相机安装在底盘上，相机与发射云台有个坐标变换，包含一个平移向量和一组旋转参数，理想安装位置，相机与云台安装没有旋转关系，只有平移关系。

平移向量直接采用直尺测量，精度在mm级别，不会引入太大误差。旋转参数选择参数采用欧拉角方式，理想状态均为0，由于误差，其真实值并不为0。可通过经验补偿方式减小误差

## 4.3算法误差（图像识别与单目定位）

采用PnP定位算法，来对目标进行定位。

定位深度距离误差预估为：8m\*(1/50)\*100% = 16cm

对于单目测距来说，远距离识别误差大，很大原因是被测量物体分辨率不够，像素波动对于定位误差影响很大。可以考虑其他的一些高精度定位方案，如双目定位等，来减小算法误差。

## 4.4弹道误差

机械因素：包含枪管等子弹发射经过的机构设计，导致子弹发射并不是平行于枪管中轴线，而形成散射的效果。

射速因素：由于远距离设计，子弹并不是直线发射，所以，还受发射速度的影响，不稳定的发射速度也会影响弹道的上下偏移。

**射速影响理论计算：**

抛物线模型：

则

假设速度存在微小波动，变成了.

则

在距离8m，速度为23m/s的情况下，计算速度波动1m/s情况下：

* 在0度仰角下，偏移误差为5cm左右。
* 在25度仰角下，为6cm。

<https://blog.csdn.net/whl970831>