

裁判系统用户手册

发布者：RoboMaster 组委会

发布版本：**错误!未知的文档属性名称**

发布日期：2019/1/1

目录

[1. 硬件部分 3](#_Toc11469)

[1.1 机器人端机械改动说明 3](#_Toc19601)

[1.2 哨岗端机械结构接口设计 3](#_Toc8890)

[1.3 传感器 3](#_Toc8404)

[1.3.1传感器选型 3](#_Toc501)

[1.3.2传感器选型理由 4](#_Toc29875)

[1.4 计算设备选型及理由 4](#_Toc8388)

[2. 软件部分 5](#_Toc31650)

[2.1自动打击算法 5](#_Toc18948)

[2.1.1 装甲板识别算法 5](#_Toc3417)

[2.1.2 自动射击时的云台控制算法 6](#_Toc5527)

[2.2 决策算法设计 6](#_Toc11543)

[2.2.1 行为树模型 6](#_Toc8477)

[2.2.2 进攻决策制定算法 7](#_Toc29677)

[2.2.3 机器人配合调度 8](#_Toc4999)

[2.3定位 8](#_Toc16007)

[2.4运动规划 9](#_Toc27216)

[2.5哨岗全局感知 11](#_Toc16903)

[2.5.1 目标检测算法 11](#_Toc17753)

[2.5.2哨岗通讯方式 12](#_Toc2288)

[2.6潜伏阶段的敌我识别 12](#_Toc20150)

[2.7调试工具设计、界面设计 13](#_Toc8243)

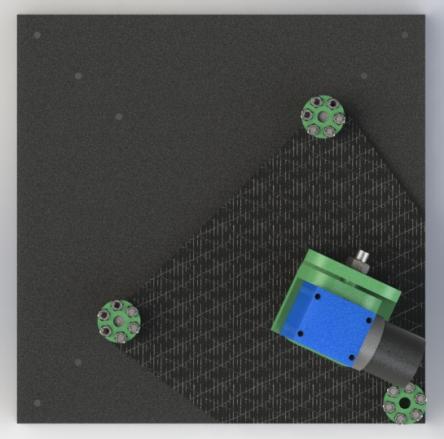
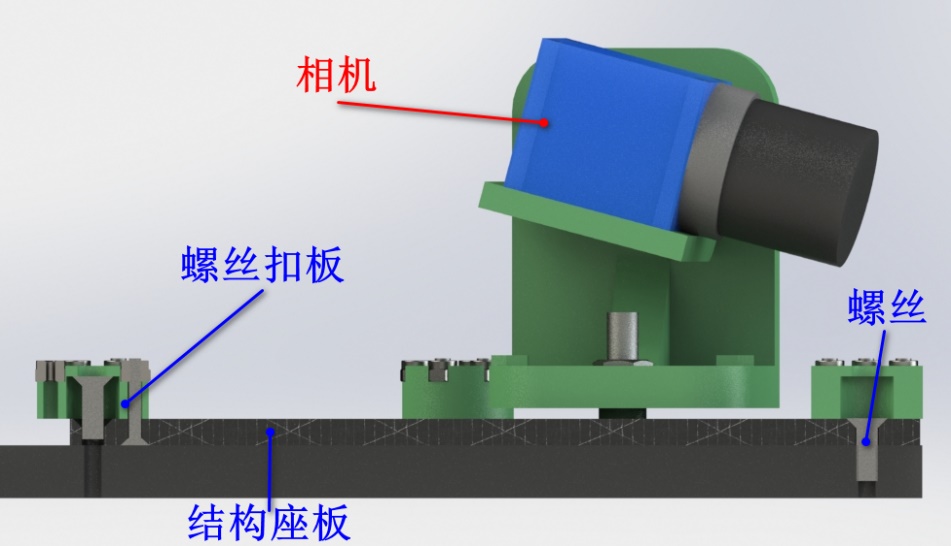
[2.8 未来展望 13](#_Toc1531)

# 硬件部分

## 机器人端机械改动说明

在YAW轴电机直连的水平板材上使用打印件固定相机；将机器人底盘前方顶部板材更换为自制固定板件，利用铜柱在其下方再布置两层板件，分别用于部署激光雷达和算力主机。

## 哨岗端机械结构接口设计



利用整张板件作为哨岗的基座；用于固定的M4螺丝被限制在扣板内，不易丢失、方便拆卸；相机架可以手动调整YAW和PITCH轴角度，两轴各由一套M6螺栓组合构成。

### 1.3 **传感器**

机器人利用传感器进行定位。在场地地图已知的情况下，可以实现路径规划、决策控制以及其他功能。

### 1.3.1**传感器选型**

|  |  |
| --- | --- |
| 传感器 | 型号 |
| 激光雷达 | RPLIDAR S1 |
| 超声波测距模块 | 电应普超声波模组一代 |
| 相机 | 迈德威视SUA133GC |

### 1.3.2**传感器选型理由**

机器人主要依赖激光雷达进行场地实时定位。RPLIDAR S1 激光雷达可以进行360°扫描测距，测量距离为12 米超过场地长度，足够机器人在场上的定位所需。由于安装处有部分遮挡，实际上测量的角度为10°~350°。

因其使用光磁融合测距，而比赛场上存在透明障碍物，对光的反射性降低，使得激光雷达产生测距误差。为了避免测距失灵，拟选用电应普超声波传感器进行补偿。该超声波模组能够检测透明障碍物，最大测量距离为450cm，盲区为3cm，测量角度为60°。该传感器一方面能够弥补激光雷达的死区，另一方面能够与激光雷达进行数据融合。为了提高联合定位精度，将超声波模组安装在车体的八个方位角。

## 1.4 计算设备选型及理由

机器人搭载的处理器为 [Intel 寒霜峡谷 NUC10 i5 FNH](http://www.baidu.com/link?url=XqtXpNZg-pt-BMxlxWt4GPI5WjdKH02egIK3f8XdUeTsnpm81gP83F12xekk357rbSVbDqsUAFkkv011ZAQbs_" \t "https://www.baidu.com/_blank)，配备8G运行内存。

哨岗电脑为自行组装，CPU为Intel i7 11700K，配备32G运行内存，高速USB3.0接口，显卡选择了NVIDIA GeForce RTX 3070 Ti，8G显存。

机器人计算端需要运行多个节点，如自瞄节点、运动规划节点（负责路径规划与导航）、通讯节点、决策节点，其中自瞄节点还要求高帧率。若自瞄系统以640\*480的分辨率读取uint\_8类型图像，保持200FPS运行，则每秒仅原图像的读入量就达到了

运动规划节点经过测试，所需运行内存平均在80Mb上下浮动。虽然以上两个数字看起来占用内存并不大，但是由于各个节点均需保证高频率运行，如导航节点的高频率（100Hz）向下位机发送底盘运动控制信息、自瞄节点的高帧率（大于200FPS）识别，此型电脑虽然运行内存上能够胜任计算任务，不过客观来说，1.6GHz主频还是显得不够高，今后可能考虑更换NUC8i7BEH，主频高达2.7GHz，能带来更优秀的算力。

哨岗需要运行两个使用TensorRT量化的，GPU加速的高帧率，高分辨率的目标检测网络。目前虽然没有测出确切的占用内存的数值，但是已知在当前配置下未经TensorRT加速的模型推理时CPU使用率可达60%以上，足以证明此配置能够支持所需算力。故所选用的配置显存、运存都较大，CPU、GPU较好。

# 软件部分

### 2.1自动打击算法

### 2.1.1 装甲板识别算法

稳定、高速的装甲板识别算法是比赛的基本要求之一。虽然深度学习的方法相比传统视觉具有很多优势，比如受场地光环境干扰小，但是其精度的提高比较依赖于相机和计算平台，故我队目前采用的仍然是传统视觉方法，在Intel NUC10代i5上有200帧/s的平均帧率，无论敌方是否处于高速运动状态，均可做出有效的识别、打击。

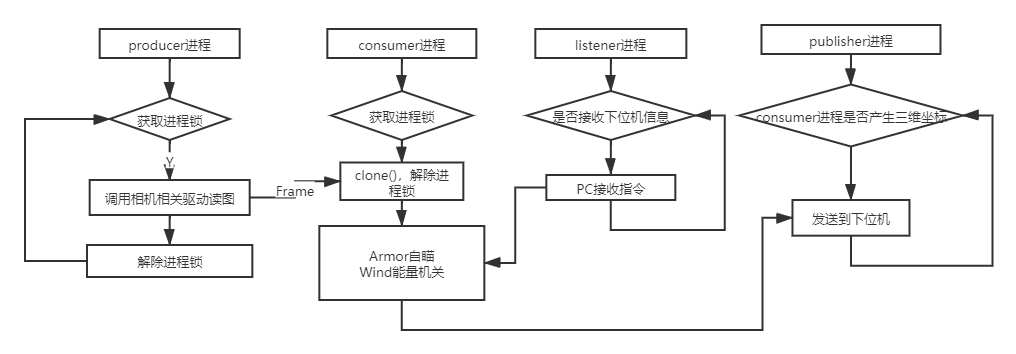
装甲板识别的主要流程是：灯条筛选、灯条匹配、位置解算。

程序从相机获取一帧图像后，先进行图像预处理：对采集的图像进行色彩通道分离、二值化、膨胀腐蚀之后，执行findContours()。遍历每一个轮廓并使用旋转矩形进行拟合，即可进行几何特征的筛选。灯条匹配时，每次取两个灯条，同样利用装甲板的几何特征，计算灯条间距与灯条长度比值、两灯条倾斜角度差等特征来进行筛选。最后一步我们只需取出装甲板的四个角点，按照对应装甲板的真实大小解算三维坐标，发送给下位机即可。

这也是常规的装甲板识别方法。不过我队的代码能够有这样的平均帧率，主要得益于这样两个设计：

1. 多线程架构

自瞄节点内我们设置了四个线程，功能分别是：读图、处理图像、接收下位机反馈、向下位机发送解算结果。下图是多线程的各线程流程图，理论依据是生产者-消费者模型(*producer-consumer-model*)。



2.装甲板跟踪算法

然而，仅仅有多线程的优化是不够的。不难发现，如果每次都把整张图像进行findContours()并筛选所有轮廓、识别装甲板，速度会很慢。为了优化检测时间，我们设置了装甲板跟踪算法。

由于我们的算法有极高的平均帧率，不难想到，在这样的图像采样频率下，任何比赛机器人所能达到的速度在图像中所造成的帧差位移都是很小的。那么，由前后两帧的时间差以及像素位置差可以得到装甲板在图像坐标系下的速度、加速度，再由位移公式即可预测出装甲板在下一帧图像中的大概位置，我们将跟踪ROI框设置为以该点为中心，边长为当前ROI边长+帧差位移\*2的矩形。由此可以加速装甲板的检测。 但是我们也会注意到,无论是装甲板在视野中消失，还是装甲板被击中时灯条闪烁，都会造成该帧图像里识别不到装甲板。所以我们应该设置一个丢失目标时间的阈值。每丢失一帧目标计数器就加1，当超过某个阈值时认为该目标消失。这样就可以解决装甲板闪烁导致的丢帧问题。

由于使用的是传统视觉方法，以及使用了上述两个优化方法，目前的程序无论对于高速运动的机器人还是距离不同的机器人，4m 有效距离内对匀速目标命中率为 96.49%，综合机动目标命中率为 69.7%，具有良好的作战能力。

### 2.1.2 自动射击时的云台控制算法

解算出目标的相机系坐标。为满足命中率要求，云台在跟踪过程中需对目标进行运动预测，而运动预测需要估计目标在惯性系 (*n* 系) 的运动状态。根据相机安装位置与姿态可得到目标的云台系坐标 (机体坐标系)。惯导姿态解算利用状态约束卡尔曼滤波器得到从云台系 (*b* 系) 到惯性系 (*n* 系) 的坐标变换矩阵 ，通过 将目标坐标由*b*系变换到*n*系以得到目标的惯性系坐标 。利用匀速模型设计卡尔曼滤波器估计目标在惯性系的位置与速度。通过目标运动状态与弹道模型实现运动预测与弹道补偿，从而得到云台的Yaw轴、Pitch轴期望角度，由云台控制系统最终实现对目标的跟踪与打击。详细数学公式推导在此不再赘述，相关论文已经上传[researchgate](https://www.researchgate.net/publication/356209829_jiyukaermanlubodemubiaoshibiegenzongyushejixitongsheji)，点击该链接前往阅读。

### 2.2 决策算法设计

### 2.2.1 行为树模型

决策系统可以左右比赛的走向，是最重要的系统。我们采用行为树作为方案选择器，同时自行设计了进攻规划算法，可以以大于100Hz的速度运行，完全满足实时性，并且将监视场上情况、打击位置寻找的过程通过数学条件量化，具有良好的可控性。

行为树是控制机器人决策流程的分层节点树，在树的范围内，叶子是控制机器人人的实际命令，而形 成分支的则是各种类型的效用节点，它们控制机器人沿着树走，以达到最适合这种情况的命令序列。机器人的决策主要分攻击、靠近敌方、躲避敌方、巡逻/等待 BUFF、迂回攻击。决策的条件主要为己方血量、己方/敌方子弹量、敌方是否在射程外。拟采取决策树结构如图。



### 2.2.2 进攻决策制定算法

对于一个进攻性决策，需要三个步骤：1.确定攻击目标；2.寻找合适打击位置3.调度我方人员机器人

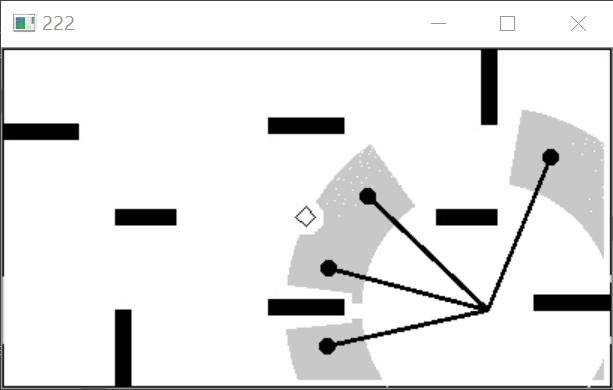
1.确定攻击目标

识别出敌方位置后，需要确定打击目标，我们设计了评估函数来评估一个目标的作战能力:

使用此公式找到敌方目前相对较弱的机器人，确定为攻击目标。

2.寻找合适的打击位置

我们认为，一个良好的打击位置应该符合如下要求：1.保持合适、稳定的距离 2.前后地势开阔，易于撤退 3.在其邻域内都可以打击到敌方，不会因为敌方的走位导致自己反复调整位置。因此，我们设计了如下算法：

设全集为地图栅格化后的点集，对于∃Enemy，取环状区域，使得，满足 。这样的环状区域（右图的灰色区域）无疑满足了条件1。

欲满足条件2、3，需在该环状区域内寻找连续的带状区域。对于

∃Enemy，以其为原点建立极坐标系，如右图所示。该带状区域需要满足两个条件：*α*域上的连续性、*α*确定时*r*域上的连续性。即连续角度α满足

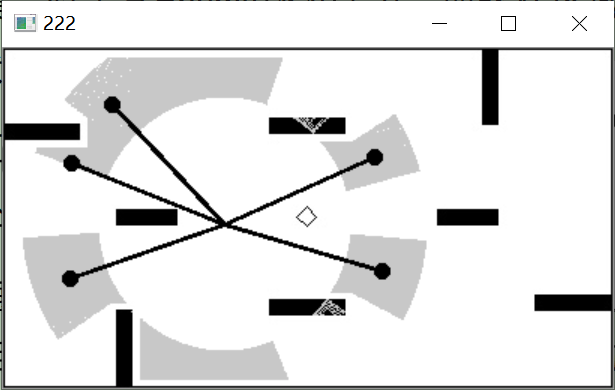
决策树代码运行截图1

，并且当α确定时，在内没有障碍物。

例如右图中，红线所夹区域不满足α确定时域上的连续性，因此不作为备选区域。

选出所有备选区域的几何中心，认定为合适的打击点。如上图所示，合适的打击点已经使用实心黑点标出。

1

3.调度我方人员进行打击

1

目前我队并未实现“暴击”机制相关的调度算法，但是当前的决策算法也已经做好了铺垫。如右图所示，对于一个敌人，已经选好了1~5五个备选打击点。由于攻击背面装甲板伤害更高，我们显然希望两个选点夹角最大。

决策树代码运行截图2

2

例如右图中1、2号点符合我们的夹角最大的条件。但是绕后的我方机器人显然更易受到围攻，所以应该调度当前作战能力更强的机器人前往绕后位。例如右图情况下，如果我方起始点在左边，那么健康状况更好的机器人应当被派往2号位进行打击，作战能力略差的则留守1号位。评估我方机器人作战能力使用的同样是这一函数。

### 2.2.3 机器人配合调度

良好的机器人配合调度能够快速建立优势、发动有效打击，是比赛中最为复杂、核心的部分。目前决策部分使用行为树算法，在设计时考虑了己方和敌方的血量、弹量、buff 激活状态、是否在火力覆盖范围内等信息作为决策系统的输入，根据状态信息变化实时输出机器人行为。机器人由自身状态可完成自主决策，可独立完成攻击、躲避、巡逻和抢占 buff 点动作。哨岗可根据判断敌我状态进行全局决策，使我方机器人协同作战进行夹击支援动作。后续可以加入强化学习来更好的调整阈值，使得决策更合理。

### 2.3定位

传感器模块获取环境信息后，定位系统读取数据后利用融合、滤波算法解算出机器人的姿态。拟选用

的定位方法如下：

1. 障碍物定位：

启动区围挡近距离围绕距离机器人，并且位于拟选用的超声波测距模块的精准测距范围内，故拟基于

八个环绕车身的测距模块数据解算出机器人的初始位姿。这种方法的初始化精度能够达到厘米级精度。

1. 雷达定位：

激光雷达的信息是机器人位置信息的主要来源。我们采用自适应蒙特卡洛定位（AMCL），该方法

利用粒子滤波在已建好的二维地图上计算求出机器人的位置，同时能够解决机器人绑架的问题，计算出里

程计的累积漂移（/map\_frame -> /odom\_frame）。在里程计漂移过大、两方处于对峙状态时，机器人将重返启动区进行里程计重置。

1. 误差累计消除：

里程计的误差积累是一个缓慢的过程。在其正常运转情况下，里程计的精度远高于其它方法。我们读 取四个轮子的编码器数据，然后结合陀螺仪解算机器人的运动。我队已在自制的车上测试，每百厘米里程 计误差稳定在 3%，并且不随时间变化。通过引入一个比例系数，我们成功弥补了这一误差。并且经过长程测试，发现没有大于 1%的累积误差。为了进一步避免比赛时可能出现的刮蹭、轻微碰撞以及打滑的情况，在已有避障措施的基础上，我们结合 AMCL 定位返回的累积漂移。即当累计漂移大于一个阈值时，返回启动区通过初始化重置误差。

1. 哨岗辅助定位：

哨岗上运行的目标检测网络给出检测框之后，认为检测框中心点向下 1/4 边长为机器人的位置中心，由手动选取的场地四个角点进行透视变换，得到机器人的位置中心在场地中的实际位置，与机器人颜色、编号绑定后实时发送给机器人。能够辅助AMCL方法进行定位。

1. 位姿指纹定位： （此方法暂未实装上车）

我们还采用结合神经网络的位姿指纹定位算法。将八个超声波测距模块得到的测距信息组成八维向量， 称为位姿指纹。将机器人在场地上的每一个位置，每一个姿态反复采集位姿指纹信息，组成详尽、 精确的位姿指纹数据库，传入全连接神经网络中进行学习、拟合。将学习得到的模型部署之后，传入实时采集的位姿指纹信息，即可推断出当前的精确位姿，此方法的精确性由海量的预采集位姿指纹保证。

目前的定位方法能够保证定位误差在6cm以内，并且能够以平均每秒2~3次的频率更新位置，能够满足当前需要。

### 2.4运动规划

为了让机器人在复杂场地中与敌方进行对抗并获胜，需要进行路径规划和避障。

我队的路径规划综合考虑了敌方位置、我方位置、最短路等因素，并且最坏情况下能够在 0.1 秒内规

划出一条带有速度信息的运动路径，平均时间在 20ms 左右，比赛时以最大频率运行，以保证规划出的路

径具有时效性。我们采用的算法是 A\*算法，该算法作为启发式搜索的经典算法之一，能够通过精心设计的

启发函数来将各种因素考虑在内，规划出最优路径。缺点是作为 BFS(宽度优先搜索)的变种，虽然有了启发函数的指导，但仍然具有盲目的尝试性，所以我们针对性设计了优化方法：

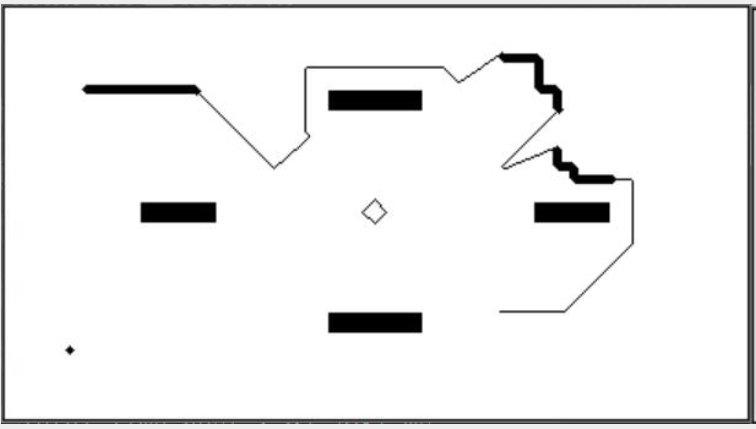
A\*所使用的启发函数 为

其中，为包括了敌我机器人位置、敌方机器人载弹量与血量的全局状态高维描述向量。函数评价了在当前状态下，我方机器人在点承受的被打击代价。 为在该点的协作利益估值函数，目前虽然已经有设计，但暂未融入程序编写，因为其功能与决策树部分的评价函数重合。,为可调的比例系数。

我们认为每一点受击概率符合以敌方机器人为中心的二维高斯分布，即

则被打击代价为：

其中，两个函数为对剩余血量、子弹的归一化函数。我们认为子弹低于阈值的敌方机器人不具有威胁能力。为可调的比例系数。这样就量化了一个敌方机器人对点的威胁程度。

将被打击代价纳入启发函数内，有利于寻找不被打击的路径进行运动。若仅考虑最短路代价，则可能盲目选择通过被敌方火力覆盖的区域，造成不必要的血量损失。

但是不难发现，这条路径由于盲目的尝试、取最优解而走了扭曲的路线，不利于机器人按照这一路线高速运动。所以必须进行二次优化。

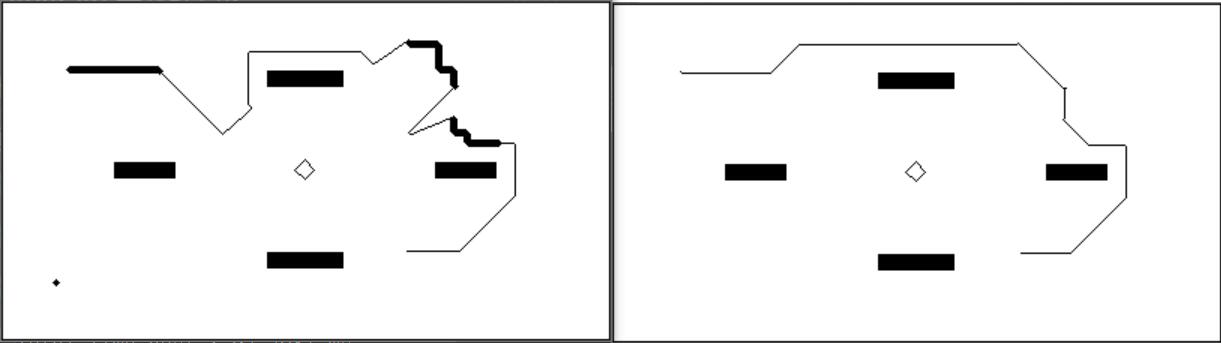
优化前路径(左下角黑点为敌人)

2)局部路径优化

我们不难发现一个事实：在敌方打不到我的时候，我可以以最短路算法快速通过当前路径；在敌方能打到我的时候，我更应以最快速度通过当前路径。所以，相邻接敌状态改变的点之间的路径是可以使用最短路优化的。

为了提高程序的模块复用性，使用的算法仍然是A\*算法，不过启发函数是：

其中，为：

对这些小段路径进行最短路优化之后得到的路径如右图所示。优化后的路径基本都是直线、易于转弯的折线，非常适合机器人高速通过。如此就实现了对固定障碍块的避障。当然，在编写过程中也使用了内存池等编程技巧优化代码，努力将路径规划这一过程速度提到最快。

优化后路径(同上图敌人在左下角)

但是，仅仅靠上述算法避障是无法满足需求的。

场地的障碍主要分为两种，一是场地上固定的障碍块，二是移动的机器人。下面分别针对两者简述避障策略。

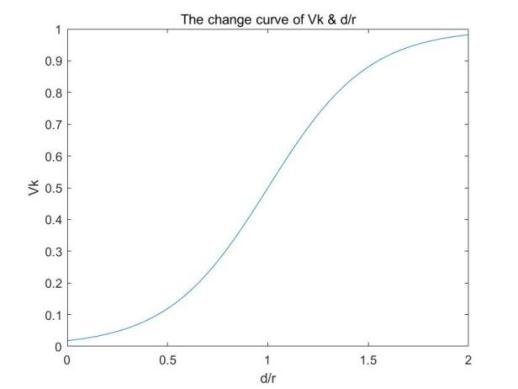
1)对固定障碍块的避障

对场地障碍块的避障是通过对预置地图的膨胀腐蚀完成的。其中膨胀腐蚀操作的结构元素要大于等于根号二倍的底盘半径，这样可以避免机器人在转弯的时候刮蹭到障碍块。算法就是上述的A\*改进算法。

2)移动物体的避障

对于移动机器人的避障，仅仅将其当前所在位置设为不可通过区域是不合理的，因为其处于移动状态。我们结合比赛地图的特质，不难发现有很多通道只能容纳一台机器人通过。故我们将比赛地图划分为若干个通道区块，哨岗通过全局的目标检测网络检测到敌我机器人之后，能够解算出其在场地中的实际位置，进而得到他目前是在哪一个通道区块当中，将这些通道称作不可通过通道，并广播给我方机器人，禁止我方机器人通过同一区块。由此即可防止机器人之间的碰撞。

不过想要实现实时避障，最根本的方法还是提高路径规划的频率。考虑到机器人实际移动的速度，如果规划算法能够达到10赫兹及以上就基本可以认为是实时。我队并没有采用ROS自带的导航包，而是自行开发了基于A\*算法的运动规划，在路径最长的情况下规划时间也仅需0.1秒，显然能够达到实时规划的要求。只要敌方的位置由哨岗广播实时更新，即可实现实时避障。

当然，哨岗也偶尔会有漏检的可能，甚至可能因为漏检而撞上敌方机器人，所以我们在速度规划内设计了根据测距模块返回的障碍距离计算的速度惩罚机制（速度势的另一种体现）。灵感起源于深度学习中的激活函数。将各个由测距模块得到的速度惩罚系数乘以该方向上的原计划速度，最后再进行速度合成，就完成了速度规划层面的避障。右图即为速度惩罚因子随d/r的变化曲线。不难发现，当d>1.5r的时候，障碍基本对速度没有什么影响了。

不妨设底盘半径为r，相距前方障碍块距离为d，则前进速度

其中速度惩罚因子

如此则实现的机器人的路径规划、运动避障。

### 2.5哨岗全局感知

### 2.5.1 目标检测算法

哨岗主要承担两个任务：全局目标检测与定位、向我方机器人发送位置坐标相关信息。具体流程是：

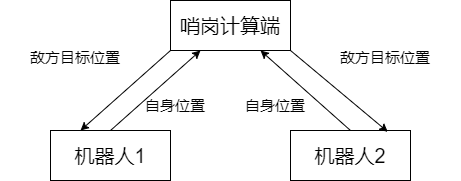
接受两车上传的位置信息，与本身目标检测网络得到的位置信息融合之后广播给机器人。

哨岗需要对连接到主机上的两个相机同时进行高速实时、高 IOU 的目标检测任务。为了提高响应速

度，所部署的目标检测网络必须经过加速，选择的加速工具是 TensorRT，单网络平均帧率 30FPS。

目标检测网络给出检测框之后，认为检测框中心点向下 1/4 边长为机器人的位置中心，由手动选取的场地四个角点进行透视变换，得到机器人的位置中心在场地中的实际位置，与机器人颜色、编号绑定后实时发送给机器人。不过，由于本赛季潜伏机制的存在，在 135 秒之后，哨岗所发送的机器人位置信息不再与颜色信息绑定，而是将车辆上传的位置与识别到的四个位置进行比对，取最近的两个位置作为我方位置，剩余两个位置即为敌方位置，通过TCP传输节点传输给我方机器人。

### 2.5.2哨岗通讯方式

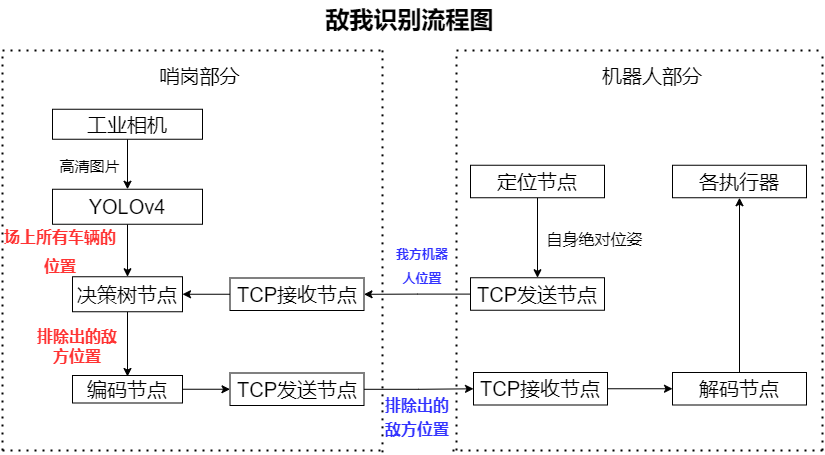
决策树部署的位置有两种：机器人、哨岗。这两种方案分别对应着集群控制算法中的分布式控制、集中式控制。二者各有优劣：分布式控制方案使得每一个机器人都具有独立决策能力，但是增加了通讯网络的复杂度，并且二者之间的协同调度不易解决；集中式控制决策效率高，易于协同调度，但是一但遭受网络波动，与被控机器人断连，则系统就会陷入崩溃。

我队综合分析后，认为集中式控制更适合当前已有系统，因此对网络通信的鲁棒性提出了较高要求。通信框图如右图所示。

### 2.6潜伏阶段的敌我识别

虽然进入潜伏阶段后不能仅仅通过装甲板灯条颜色来识别敌我

我队的核心思路是：我方机器人始终能够定位出*自己的位置*，再结合哨岗已经识别出场上*所有机器人的位置*，使用排除法即可轻易识别出敌我。算法流程图如下：



哨岗通过运行的目标识别程序可以检测到场上所有车辆的位置；我方各车辆上运行的定位节点可以得到自己的位置。各车辆将自身定位传输给哨岗，将车辆上传的位置与识别到的四个位置进行比对，取最近的两个位置作为我方位置，剩余两个位置即为敌方位置，通过TCP传输节点传输给我方机器人。

进入潜伏时间后，自瞄识别的灯条颜色由仅敌方颜色改为双方颜色。不过由于我方机器人已经获得了敌方位置，自瞄又可以解算出装甲板与自己的相对位置，那么就可以执行如下坐标变换：

得到敌方的绝对位置。打击前查看从哨岗处接收的敌方位置列表，如果不在列表里，说明识别到了己方潜伏的机器人，如果在列表里，就立即发起进攻。

### 2.7调试工具设计、界面设计

所有有需要的节点均使用Opencv编写了可视化界面，结合其内置的鼠标事件回调函数能够轻松实现鼠标控制。不过目前仍有部分节点的调试UI界面设计尚未完善，今后必然会完成编写。

调试工具选择了ROS自带的简单、好用并且强大的rqt工具。该工具可以监视话题上的数据包、节点间的通讯，也能够手动控制数据包的发送。

节点代码的测试也使用了Visual Studio自带的内存检测工具，用以评价节点运行时所需运行内存的大小、有无内存泄露等漏洞，以便提出针对性优化。

### 2.8 未来展望

我队目前在挑战赛方面总体上处于技术积累阶段，尚未形成足够成熟、高效的系统。不过大部分功能都已经实现并且可以实战，但是有很大的优化空间。未来的时间内将会以优化现有代码，尤其是优化定位算法方面为工作重点，同时增强各机器人之间的通讯、协同围捕作战。