长空御风
雷达站
开源报告

王书钰 1134643765@qq.com
刘建航 1358446393@qq.com
南京航空航天大学 长空御风

目录

1. 机器人功能定义

图2-1 机器人功能定义

2. 机器人核心参数

表 1.1：雷达站主要技术参数

主要参数	数据
重量（含线缆）	5.7kg
总体尺寸（长*宽*高）	700mm*700mm*1450mm
激光雷达数量	1
相机数量	3

表 1.2：雷达站传感器类型

传感器	型号
左右相机	MV-SUA134GC-T 8mm 1:2.0 1/1.8”
上相机	Realsense D435i
激光雷达	Livox Mid70

3. 设计方案

3.1 机械结构设计

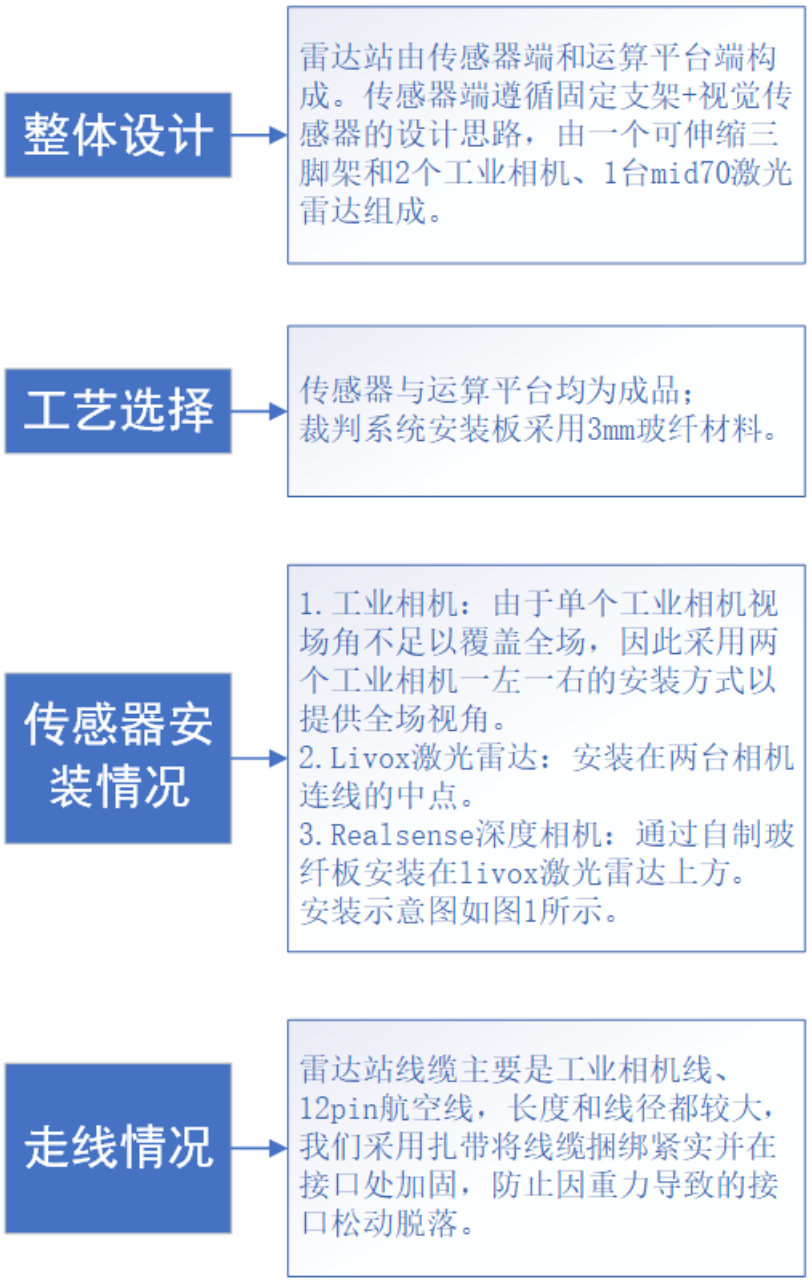

图2-2 机械设计方案

雷达站的硬件结构主要分为支架、传感器和配件三部分。

支架

我们使用了普通的三脚架作为雷达站支架，同时使用了配套的滑轨和球头云台，使得相机可以在一定范围内自由转动。

图2-3　雷达云台和滑轨

传感器

我们一共使用了两个工业相机和一个激光雷达作为主要的传感器，两个相机互相补充视野。同时为了使云台手可以看到完整的画面，我们额外增加了一个广角相机。

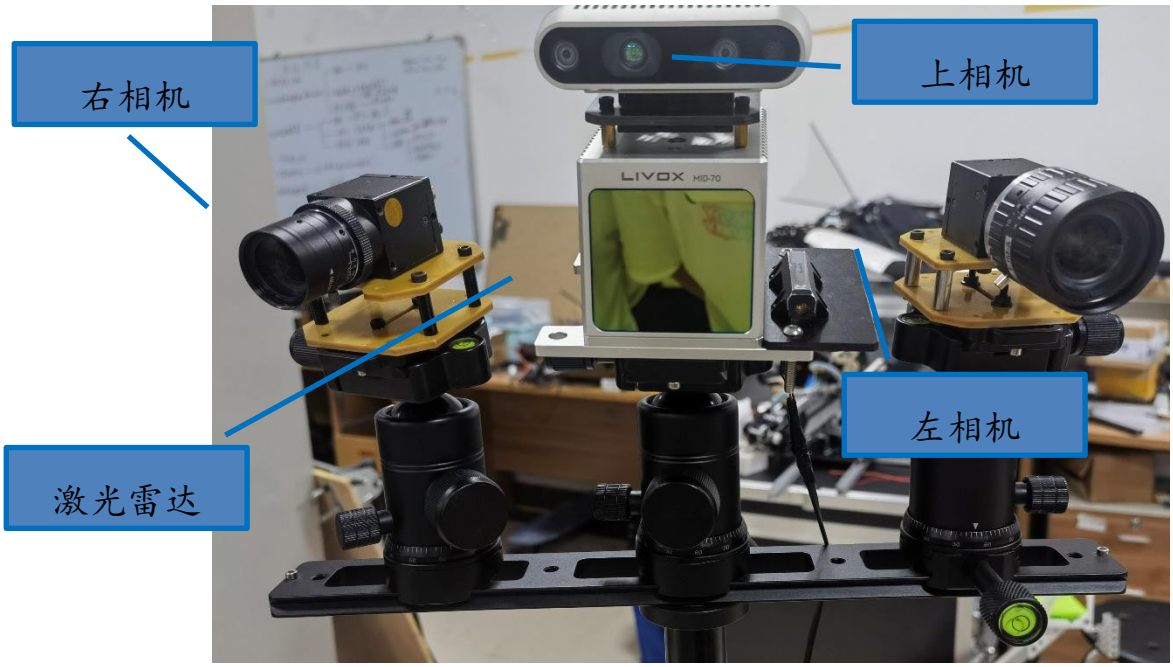

图2-4　雷达传感器

配件

诸多有趣、实用的小配件也是我们雷达站的一个小特色。

为了解决主控和电源管理模块和运算端的固定问题，我们设计了一个支座，可以将裁判系统和电池都固定上去，提高了检录和上场时的通行速度。

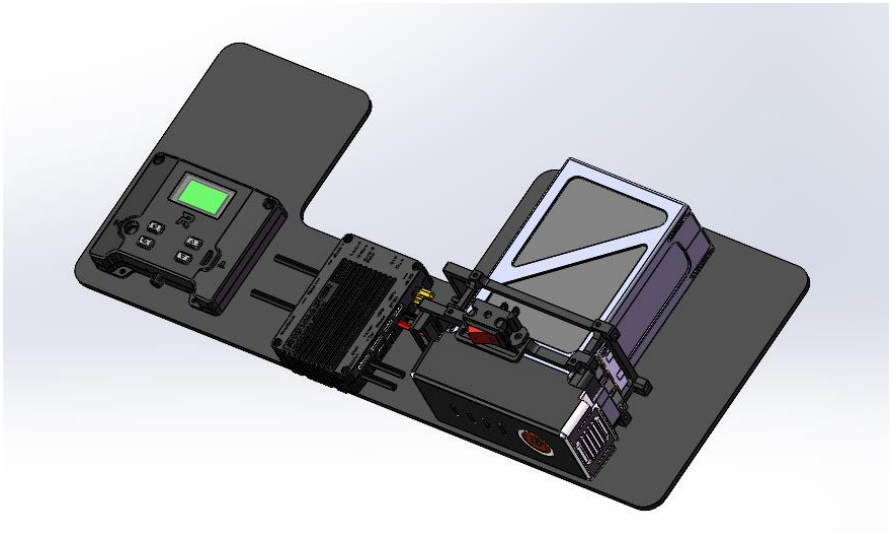

图2-5 裁判系统底座

工业相机和激光使用 M3 螺栓进行固定，而支架上是标准相机螺栓，同时广角相机需要固定到激光雷达上方。于是我们制作了转接板。既解决了固定的问题，又保留了调节角度的功能。

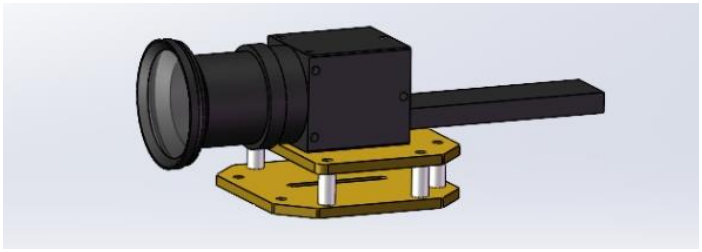

图2-6 工业相机转接板

图2-7 Realsense 转接板

虽然我们使用了两个相机补充视野，但是视场角仍然没有很大的余度，使得我们对上场时雷达的摆放角度要求较高。为了方便在紧张情况下快速将雷达摆放稳妥，我们在雷达上增加了红点瞄准。如图所示：

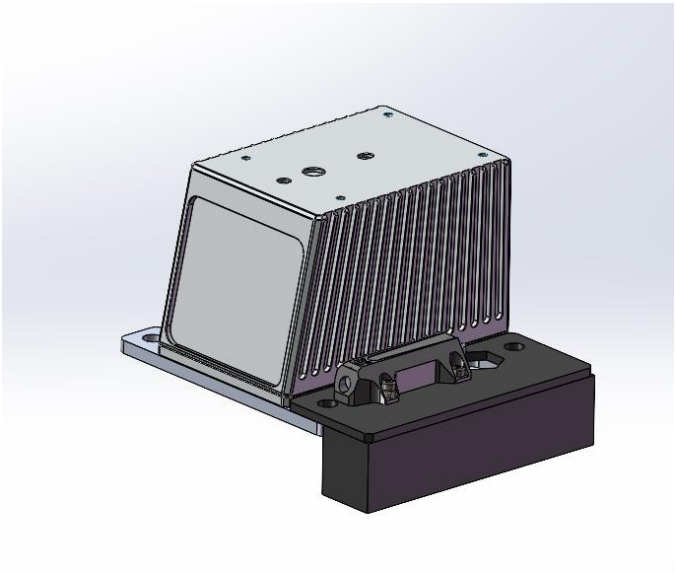

图2-8 红点瞄准器

3.2 软件设计

系统架构

系统层级:

整个雷达站采用 ROS（机器人操作系统）开发，代码遵循 ROS 架构。分为 4 层：硬件驱动层，数据处理层，功能逻辑层和前端显示层。

第三方中间件及模块:

CUDA_10.2, TensorRT, Tensorrtx, yolov5-6.0，OpenCV-4.5.4, Cmake

另一些驱动程序，以 ROS 功能包的形式运行：Livox_ROS_Driver, MindVision_ROS_Driver, RealSense_ROS_Driver,ROS_Serial

开发及运行环境:

Ubuntu20.04(操作系统), CLion(软件开发环境)

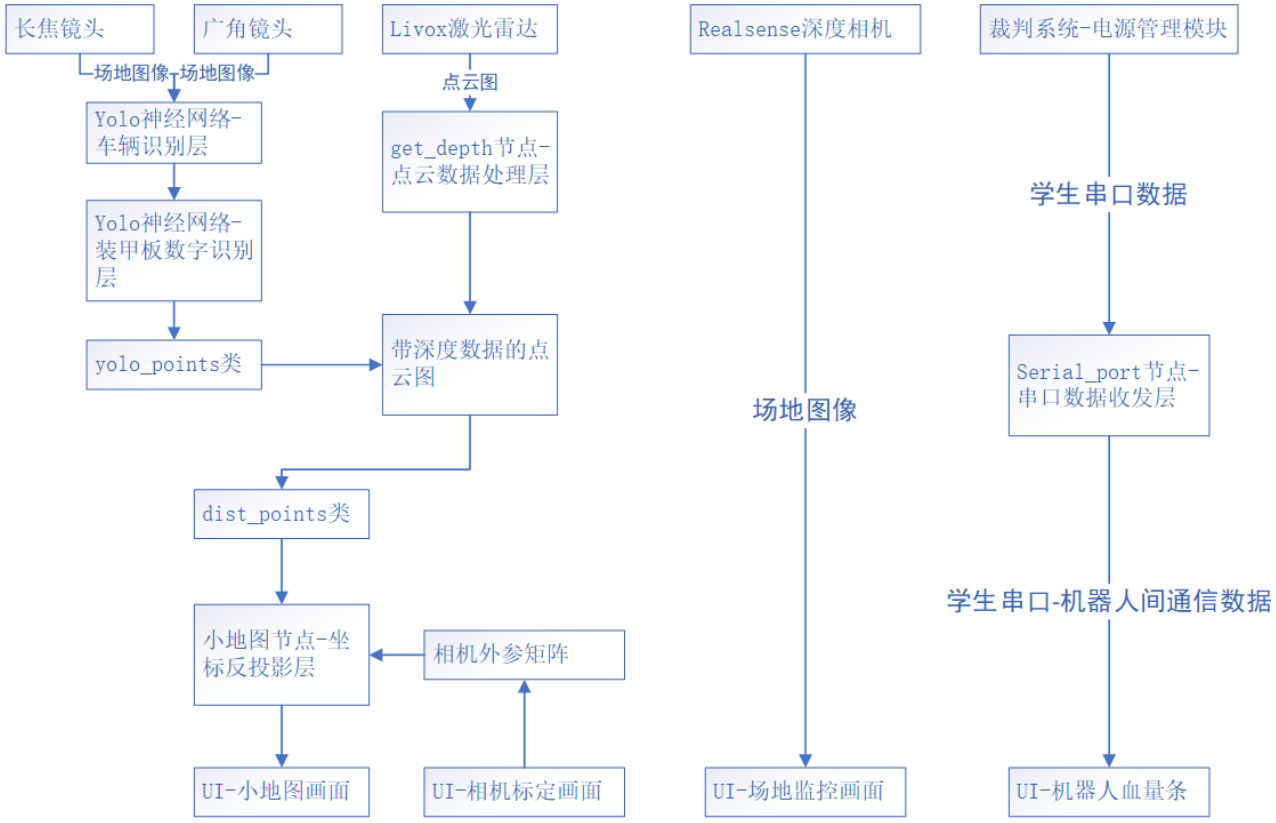

图2-9 运行流程图

重点功能

小地图功能

小地图功能测算出场地上车辆的世界坐标，并绘制在场地俯视平面图上。处理过程见运行流程图（图），在这个功能中，我们使用两层深度神经网络对车辆及其兵种类别进行了识别，神经网络使用 yolov5-6.0，在自制数据集上进行训练。使用 PCL 点云库对 Livox 激光雷达获取的场地点云图进行处理，得到每个点距离雷达的距离数据。使用事先标定好的雷达与相机之间的旋转平移矩阵将车辆识别框投影至点云图中，对每个投影框内部的点的深度求取距离统计值（依照点个数多少分别取不同的统计值），得到该车辆相对雷达站相机的距离，将距离和车辆在相机画面中的二维坐标打包发送至小地图节点。在小地图节点，利用赛前 3 分钟之内标定好的相机与场地外参矩阵及含深度的二维坐标，使用公式 1 求解出车辆的世界坐标，完成车辆世界坐标的解算，并通过 opencv 绘制在 ui 中。

哨兵辅助自瞄

在小地图功能的基础上，选取每次小地图解算结果中的己方哨兵世界坐标 Og 和所有敌方车辆坐标 E1,E2,⋯ ,En,计算出以己方哨兵为原点的相对坐标 OE1,OE2,⋯,OEn。将相对坐标发送给哨兵，哨兵通过姿态解算得到应朝目标转

动的云台 pitch,roll,yaw 角度，完成辅助自瞄。

英雄前哨站测距

在赛前 3 分钟内，圈出前哨站位置并获取其二维坐标。使用与小地图相同的功能测算出前哨站的世界坐标。在小地图解算结果中选取己方英雄的世界坐标，计算前哨站与英雄的欧氏距离。

敌方车辆实时预警

将每次小地图解算结果中的敌方车辆坐标加以判别，若其出现在己方半场的危险区域（程序中定义），则在小地图中将对应区域置为红色高亮显示，并在 ui 消息框中打印预警消息。

标定

精准的标定是雷达站能够准确定位的基础，为了使标定更加准确，我们在赛季初进行了很多的尝试。最终使用了如下的方法。

相机内参标定

我们使用了 ROS 下的 camera_calibration 包标定相机内参。相比于 OpenCV 或者 Matlab 中的标注方式，camera_calibration 的显著优点是自动化程度极高。通过进度条指引标图者调整标定板的角度，并且无需手动拍照，短时间内便可以自动拍摄和解算大量的图片。操作的方便间接的提高了标定的精度。通过这种方式便可以获得相机较为准确的内参。

$$K = \begin{matrix} f_x & o & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{matrix}$$

联合标定

首先，我们使用了 livox 官方提供的激光雷达标定方式（https://github.com/Livox-SDK/livox_camera_lidar_calibration）进行手动标定。在拍摄了多角度的图片并录制点云 rosbag 后，分别使用鼠标在点云图和彩色图中按照相同的顺序点出四个角点，通过 PNP 解算即可获得旋转和平移矩阵。

由于为手动标注，工作的繁琐注定了图片和点云图的数量不会太多，同时由于手动选点的误差，使得得到的参数注定不会十分精确。所以还需要想办法进一步提高联合标定的精度。

我们使用了香港大学开源的无目标场景下激光雷达和相机自动标定（https://github.com/hku-mars/livox_camera_calib）的方法。该方法可以自动提取深度图和彩色图中共同特征，并通过迭代的方式，使得误差达到最小。通过这

种方式，我们进一步提高了联合标定的精度。

串口通信

串口通讯是雷达站发挥作用的重要窗口，通过官方的小地图通信和车间通信，以及裁判系统下发的其他消息，可以大大提高雷达站这个兵种的上限，实现出奇制胜。

我们使用了 USB 转 TTL 模块实现雷达站运算端和裁判系统的通信。使用了 ROS 的 serial 包实现。

串口接收

我们从裁判系统中读取了机器人血量、比赛状态、比赛结果、补给站状态、场地机关状态、裁判判罚、飞镖闸门关闭倒计时等数据。程序主循环不断试图从串口中读取数据，当存在可用数据时，便将其读入数组中。从串口中读入可用数据后，依次使用所有消息类型进行检验，通过 CRC 校验判断是否匹配。如果匹配，则为该类型消息；否则，则使用其他类型进行检验。

小地图消息

小地图基于官方的通信协议实现。消息以 10Hz 定频发送。来自小地图节点的坐标信息被存放在数组中，依次向裁判系统发送。当数组被清空后，重新从小地图节点获取新的坐标信息。以此来实现小地图通信频率和解算频率的匹配。

其他消息发送

我们使用了车间交互消息，来实现雷达赋能其他兵种。消息的发送策略与小地图类似。消息定频发送，当没有交互信息需要发送时，则发送约定好的空消息。

3.3 算法设计

重要算法原理阐述、公式推导

小地图反投影算法原理

小地图算法的流程在 1.4.3 中已提到，其可以更抽象地概括为：世界坐标系中一点的坐标向相机坐标系中投影（正投影过程）+投影点使用深度 d 和内外参矩阵的逆 inv[R,T], invM 向世界坐标系反投影的过程（反投影过程）。其中正投影过程已经在相机的成像过程中完成，而反投影过程则需我们编写算法，使其在运算平台端的计算中完成。这其中亦包括对深度 d、inv[R,T]、invM 的获取和计算，我们将这三个参数称为投影参数。因此，小地图算法的核心步骤即为两个：1.投影过程。2.投影参数的获取及计算。

下面，我分别介绍这两个核心步骤的原理及实现。

1.正投影过程和反投影过程有着数学上互逆的关系，只要理解了正投影过程的公式表示，就可以推出反投影过程。这里以正投影过程为例来讲解。正投影的过程可概括为：世界坐标系->相机坐标系->成像平面坐标系->像素坐标系。

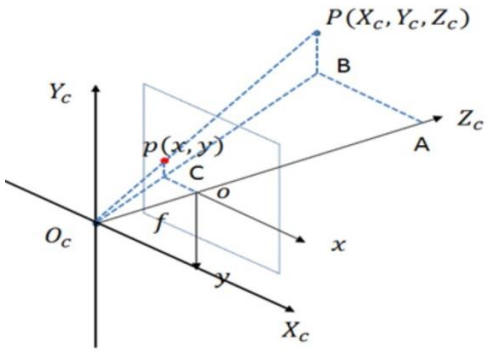

图2-10 小孔成像相机投影示意图

如图 2-1 所示为一个小孔成像相机模型。相机坐标系 $X_cY_cZ_c$，成像平面 uv，相机坐标系的 Zc 轴与相机光轴重合。有真实世界中一点 P，其在相机坐标系下的坐标为(Xc,Yc,Zc)，则根据小孔成像原理，P 点与相机坐标系原点 Fc 的连线与成像平面坐标系 xy 交于点 P'（坐标为(x,y)），此即为真实世界中的点 P 在相机成像平面上的二维投影点 P'。由几何学关系可得：

$$\frac{O_cO}{Z_c} = \frac{OC}{BA} => \frac{f}{Z_c} = \frac{x}{X_c} => xZ_c = fX_c \tag{1.1}$$

同理可得

$$yZ_c = fY_c \tag{1.2}$$

将其转化为矩阵关系式，即为

$$Z_c \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1.3}$$

之后，我们将 P'从成像平面坐标系 xy 转化至像素坐标系 uv，需对 x 轴和 y 轴分别乘以每毫米像素点数(dpm)，并加上成像平面中心点相对 u 轴和 v 轴的偏移量 u_0 和 v_0.即

$$p(u, v) = p\left(\frac{x}{dx} + u_0, \frac{y}{dy} + v_0\right) \tag{1.4}$$

其中 1/dx 和 1/dy 分别为 x 轴每毫米像素点数、y 轴每毫米像素点数。

将公式 1.4 转化为矩阵乘法形式并与 1.3 合并可得

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1.5}$$

其中 fx=f/dx, fy=f/dy. 至此我们完成了相机坐标系->成像平面坐标系->像素坐标系之间的转换。为方便将公式 1.5 中的系数矩阵记为相机内参矩阵 M。

现在，我们假设世界坐标系为原点在图 2-2 的左下角，向右为 x 轴，向上为 y 轴，向屏幕外为 z 轴。此世界坐标系记作 $X_wY_wZ_w$.

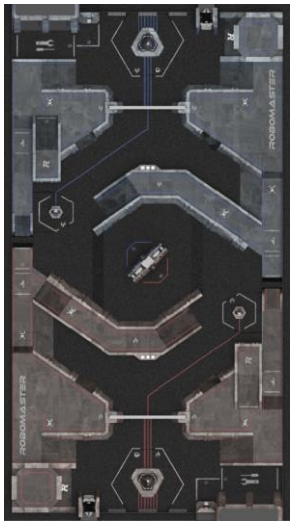

图2-11 赛场俯视图

根据空间直角坐标系之间的线性变换关系，$X_wY_wZ_w$ 到 $X_cY_cZ_c$ 的变换关系为

$$\begin{aligned} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} &= R \begin{pmatrix} x_{\text{world}} \\ y_{\text{world}} \\ z_{\text{world}} \end{pmatrix} + T = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} x_{\text{world}} \\ y_{\text{world}} \\ z_{\text{world}} \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix} \\ &= \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \end{aligned} \tag{1.6}$$

其中，R 记作两个坐标系之间的旋转矩阵，t 记作平移矩阵。

综合公式 1.5 和 1.6，我们可以写出世界坐标系到像素坐标系之间的变换关系

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M \cdot \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \cdot \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{1.7}$$

公式 1.7 即为世界坐标系到像素坐标系之间的转换关系，亦即正投影过程。

由线性代数理论推导，可得由世界坐标到像素坐标之间的转换关系（即反投

影过程）为

$$\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = invR \cdot \left(invM \cdot Z_c \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - t \right) \tag{1.8}$$

其中 invR 是 R 的逆矩阵，invM 是 M 的逆矩阵，Zc 是点 P 在相机坐标系 Zc 轴的截距，其近似等于通过激光雷达所求得的点 P 距离雷达光源点的深度 d。在实际的程序代码中，我们使用深度 d 代替 Zc，解算精度达到要求.

$$\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = invR \cdot \left(invM \cdot d \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - t \right) \tag{1.9}$$

2.投影参数的获取与计算

从公式 1.9 中可见须获得的参数是 invR、t、invM、d。下面分别介绍他们的获取方法。

(1)相机的内参矩阵 M 我们由棋盘格标定法获得。求解其逆矩阵 invM。

(2)旋转和平移矩阵 R、t 由我们在赛前 3 分钟内使用 solvePnP 方法临场标定获得。我们使用经典的四点法，事先选取好赛场上的 4 个标定点，根据规则手册计算出以式 1.9 中 $X_wY_wZ_w$ 为基准的世界坐标，在赛前 3 分钟内通过 ui 选点操作获取其像素坐标，输入 OpenCV 库的 cv::solvePnPRansac 函数并计算出旋转和平移矩阵 R、t。通过 cv::Rodrigues 将 R、t 转化为罗德里格斯形式，并求解其逆矩阵 invR。

(3) 使用 PCL 点云库对 Livox 激光雷达获取的场地点云图进行处理，得到每个点距离雷达的距离数据。使用事先标定好的雷达与相机之间的旋转平移矩阵将车辆识别框投影至点云图中，对每个投影框内部的点的深度求取距离统计值（依照点个数多少分别取不同的统计值），得到该车辆相对雷达站相机的距离 d，将距离 d 和车辆在相机画面中的二维坐标打包发送至小地图节点。

车辆目标检测及兵种识别算法原理

在相机节点获取到图像之后，图像被送入神经网络节点进行推理。

由于官方的小地图通信频率只有 10Hz，对于系统的帧率要求并不高；所以我们采用了双神经网络的设计。通过这种方式，解决了在雷达站全画幅画面中装甲板尺度过小，场地中干扰过多的问题，同时也使得数据集标注更加方便。

我们使用的数据集为 DJI ROCO 数据集、RMCV 视觉开源数据站（https://rmcv.52pika.cn）中的数据集以及自行制作的数据集。数据集规模为

15000 张左右（识别车辆）和 3000 张左右（识别装甲板）。

图2-12 装甲板数据集示意

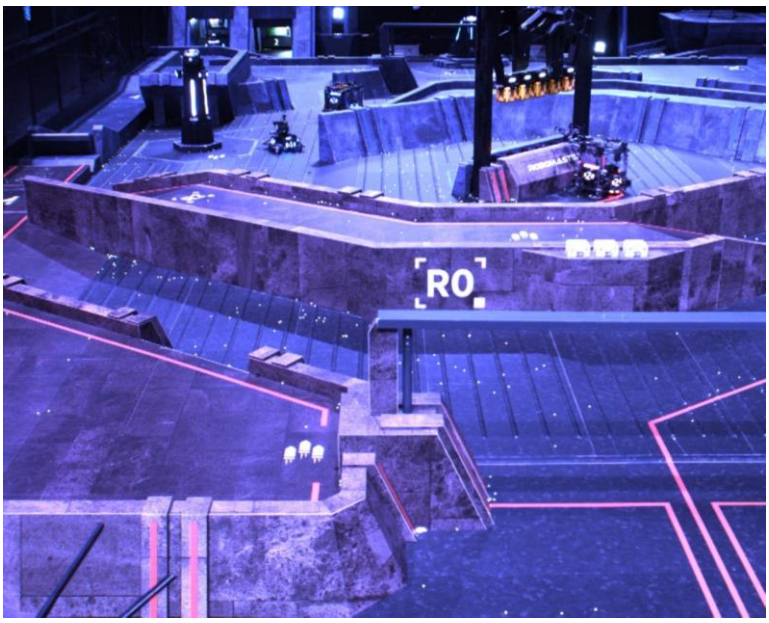

图2-13 车辆数据集示意

并非所有的数据集均为手动标注。在标注时，我们仅先手动标注一小部分，进行训练；然后用所得模型推理新的数据集，然后对数据集进行校对；重新训练，获得新的模型；增加新的数据集……如此，仅需手动标注少量图片（其余需要校对），便可以短时间内标注大量图片。通过这样的方式，标图的效率大大提高。

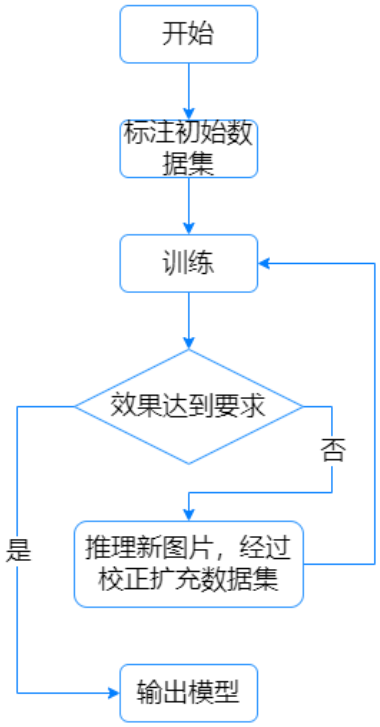

图2-14 数据集标注流程图

我们的内外两层神经网络均使用的是 Yolov5s 神经网络，并使用了 TensorRT 加速，其中第一层的输入尺寸为 640*640，第二层为 128*128。第一层的主要任务是从场地中识别到蓝方车辆和红方车辆，第二层的主要任务是通过识别装甲板数字判断兵种。

具体流程如下。从相机节点获取到图片之后，对于每一张图片进行推理，获得车辆的 bbox，并获取与之对应的图片 ROI，送入第二层神经网络，以获得每个车辆的 ID。考虑到第一层神经网络获得的 ROI 在大部分情况下并不唯一，因此我们将多个 ROI 合为一个 batch 送入第二层。经过多次尝试，我们最终选择以 8 个 ROI 为一个 batch，使得在大多数场景下，一个 batch 可以包含所有的 ROI，同时在 ROI 较少的情况下也能保持较高的效率。通过这种操作，第二层神经网络对整个系统的影响大大减小。经测试，batch size 为 8 时第二层的运行速度，比 batch size 为 1 时的速度，在大多数场景下快一倍以上（推理相同数量的 ROI）。第二层神经网络的存在，仅仅使得推理时间增加了 15%左右。

当第二层神经网络只识别到一个装甲板时，则直接输出；当识别到不止一个装甲板时，则选取置信度最高的装甲板输出；当没有识别到装甲板时，则将 ROI 缩小一定尺寸（减少车辆本身的镂空造成 Z 值偏大和车辆被场地遮挡造成的 Z 值偏小），作为结果输出，此时仅能分辨敌方和我方车辆。

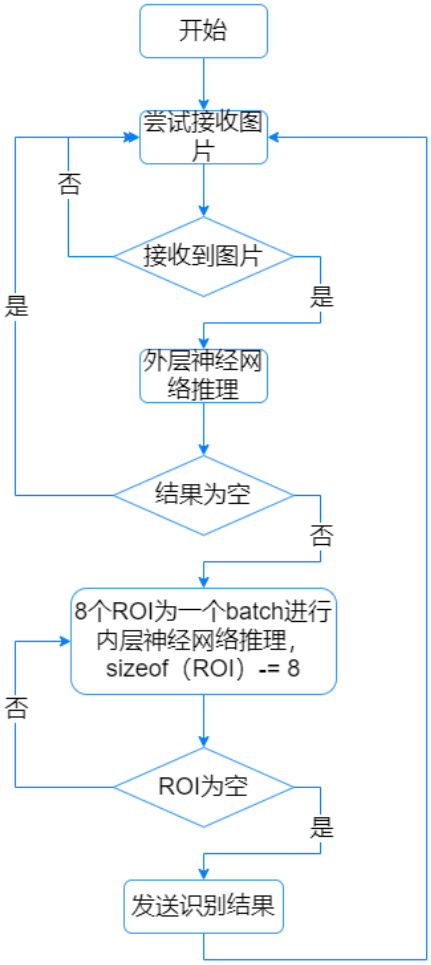

图2-15 神经网络节点运行流程图

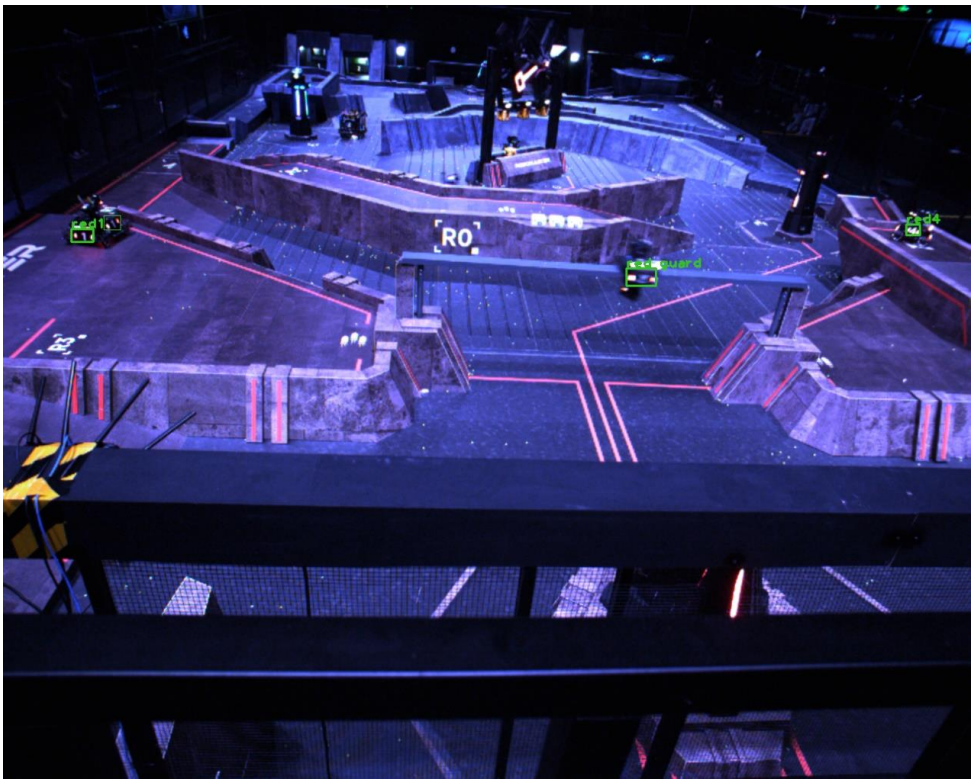

图2-16 双层神经网络识别效果

神经网络仅仅对所有地面单位和哨兵进行了识别，而且由于数据集有限，对于装甲板的识别准确率有限。这是目前神经网络节点存在的主要问题。

性能分析及优化方案

与完整形态相比，主要改进有：增加了云台手操作端图形化界面（UI），完成了小地图功能及其派生功能：哨兵辅助自瞄、英雄与前哨站测距、敌方车辆预警。

UI 性能分析及优化方案：我们的 UI 起初在 Ubuntu18.04 系统上进行开发，试运行过程中，发现 UI 启动 30 秒左右之后就会逐渐卡顿直至卡死。经过 Perf 工具的分析我们发现，雷达站 ui 显示、神经网络推理和激光雷达点云数据处理这三个节点的 CPU 占用率最高，其中 ui 显示在三者中最高。我们尝试优化 Qt 程序中的图像显示函数，但效果不明显。最终，我们将雷达站程序迁移至 Ubuntu20.04 系统中重新运行，在一行代码未改的情况下，消除了卡死问题。对于这个问题的成因以及迁移系统即解决问题的内在原理，我们还在探索。

小地图性能分析及优化方案：在精度方面，解算的车辆世界坐标精度在 2m 以内。受制于神经网络识别准确度的限制，当出现误识别情况时，会导致小地图画面中车辆点的颜色和数字绘制错误；同时，由于投影参数 d 的误差甚至错误也是影响精度的重要原因。针对此问题，我们尝试记录本次识别结果与上一帧识别结果，将二者进行综合比对，消除识别框中的干扰点，以获得正确的深度 d。

由于时间有限，该优化方案尚处在开发测试过程中。在运行速度方面，小地图在每秒内可更新 10 次以上，这已经超过了车间通信的带宽(10Hz)，因此小地图可以最高的刷新率在机器人操作手的小地图画面上显示。

哨兵辅助自瞄性能分析及优化方案：在禁用哨兵自身的视觉自瞄条件下，完全使用雷达站的视觉识别方案可将哨兵的自瞄精度做到 1m 以内（哨兵红点距车辆的距离误差）。这个精度已经可以令哨兵根据雷达站提供的方位将云台快速瞄向目标的大致位置，再使用自身的自瞄系统对目标进行精确的打击。由于时间限制，并未测试辅助自瞄与自身自瞄共同工作时的工况，可以预计到会出现哨兵只使用辅助自瞄而弃用自身自瞄的误决策问题，这类问题有待进一步的测试与优化。

英雄测距性能分析及优化方案：英雄与雷达站距离测量误差在 1.5 米以内。对于英雄的吊射重力补偿程序来讲，这个误差还不够小。我们尝试在距离解算结果上加上一个固定补偿量来修正，结果是对于同一场比赛，修正后的测算距离基本能够保持在 1m 以内的误差，这个结果已被控制在重力补偿程序允许的输入误差范围内，因此我们的优化方案就是在每场比赛开始前，快速地测出固定补偿量的值并传给英雄控制程序，使其在本场比赛中可以使用精度较高的距离值来吊射。

自定义 UI

如图所示，展示雷达站的 PNP 选点，场地敏感地区预警、显示血量、飞镖打击回放、比赛状态显示、战场信息显示的功能。

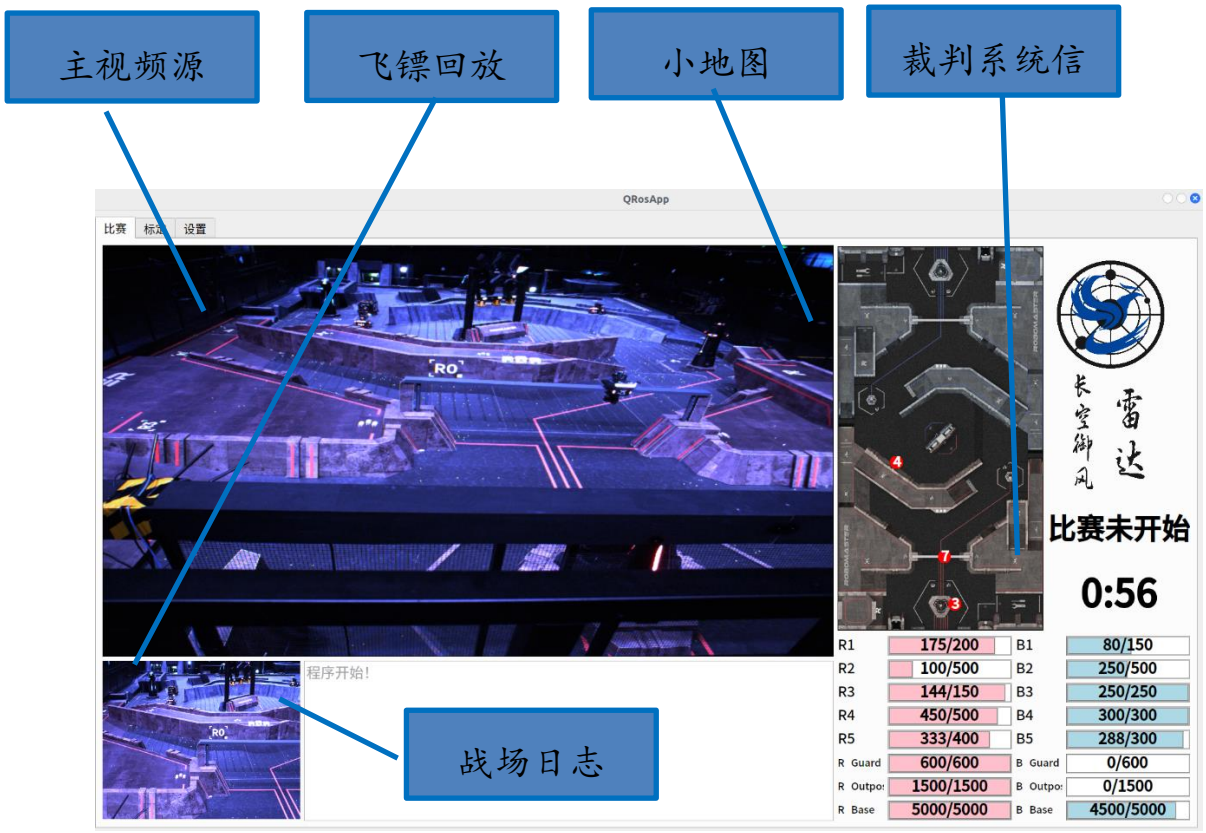

图2-17 自定义 UI 主页面

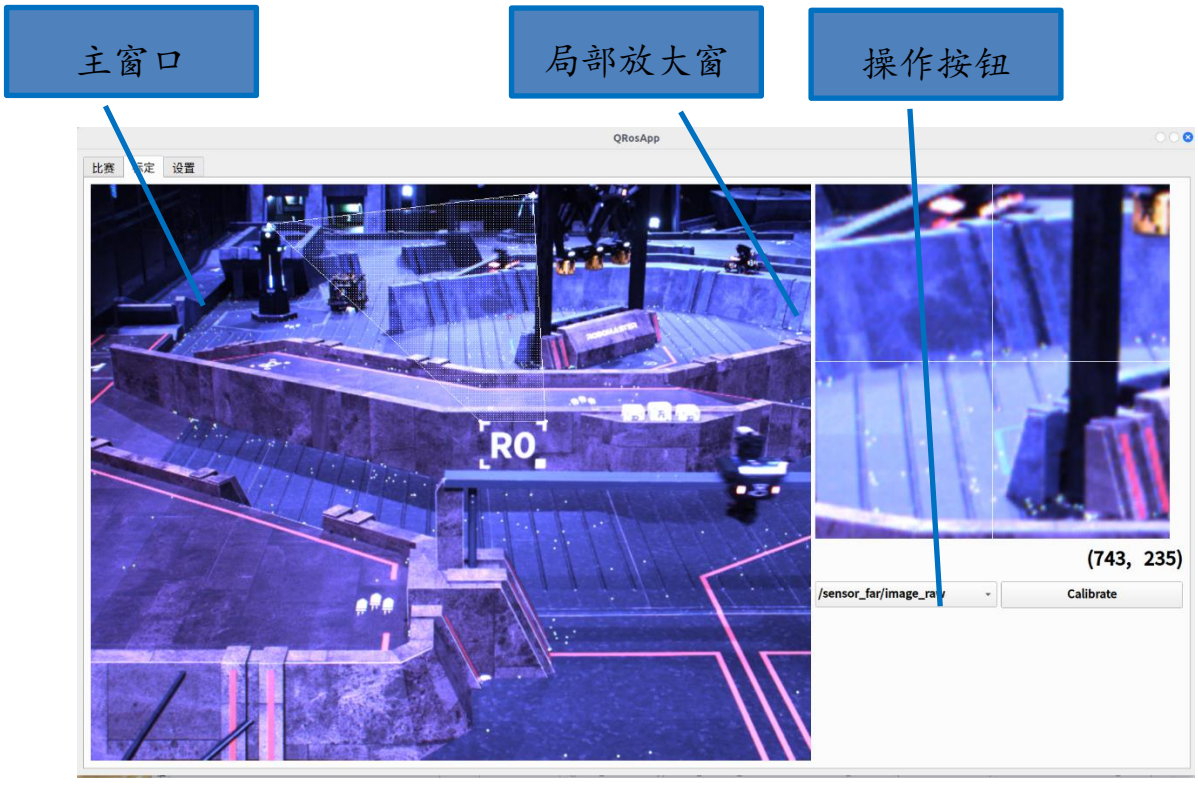

图2-18 自定义 UI 位姿估计页面

主视频源

主视频来自雷达站的上相机，为 UI 主页面的主要部分，方便云台手最直观的获取战场信息。基于 Qt5 的 QLabel 类实现，通过不断更换 QLabel 的贴图实现视频的实时显示效果。

战场信息显示

场地日志框是我们雷达站的一个特色功能。雷达站通过运算得到的信息（如敌方步兵飞坡、英雄击打前哨战等）和通过裁判系统直接获取到的信息（如判罚信息、能量机关激活情况）可以通过场地日志直接打印出来。

我们根据事件的紧急程度，将其分为五个等级，通过不同的字体、颜色显示。当提示框占满后，后续消息会自动覆盖最久的消息。

小地图

小地图是雷达站最基本的功能。雷达站自定义 UI 上的小地图主要承担着两个重要功能，一个是显示双方车辆位置，一个是区域预警消息的直观显示。

在比赛开始后，UI 会根据我方颜色加载正确的小地图背景，以使云台手获得最直观的观感。UI 收到来自小地图节点的解算消息后，根据兵种和阵营，将其绘制在小地图上。当收到来自小地图节点的预警信息后，UI 会在小地图对应区域以 3hz 的频率闪红，同时会在战场信息中进行打印。

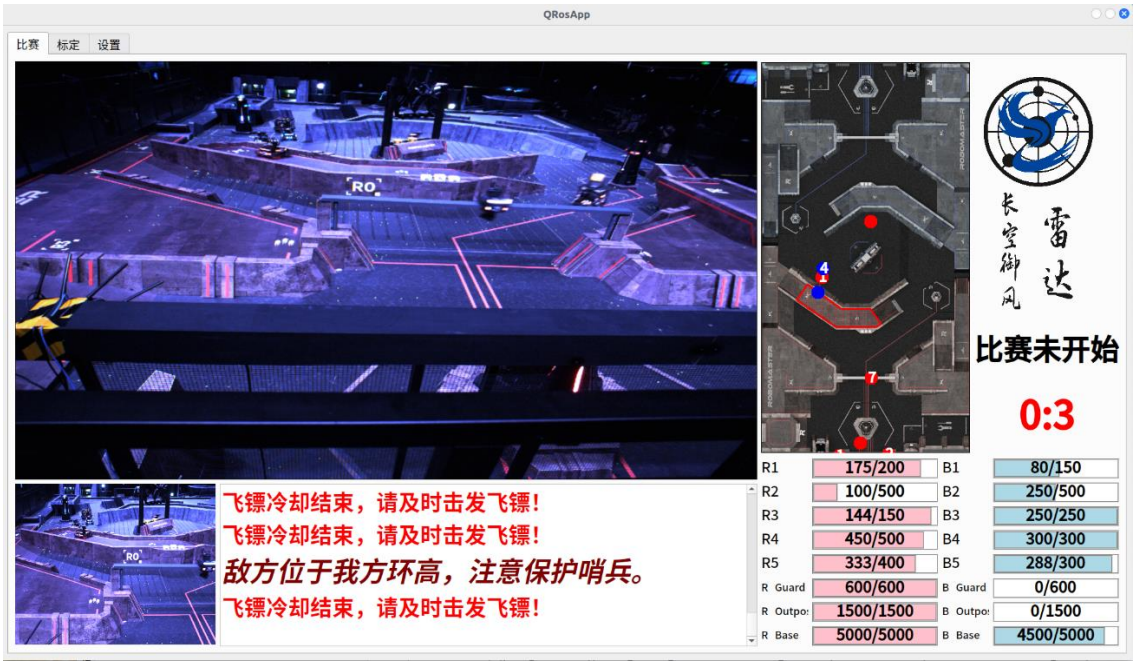

图2-19 小地图预警示意

飞镖打击回放

飞镖打击回放是为云台手击发飞镖提供便利的一个辅助功能，也是我们探索雷达站赋能全兵种的一次尝试。

在四川大学纯机械飞镖出奇制胜后，无控飞镖成为了主流。在这种背景下，云台手手动的瞄准和微调无疑是至关重要的。出于此目的，我们设计了飞镖打击回放功能。

当从裁判系统读取到飞镖闸门开启消息后，系统会自动开始记录左相机中飞镖行进轨迹部分的画面，将其保存在硬盘中。在飞镖闸门关闭自动从硬盘中加载，并开始连续三次回放，使得云台手有足够的时间评估本轮飞镖的打击效果，并进行微调，使得下轮飞镖打击取得更好的效果。

但是，雷达站作为和飞镖有着千丝万缕联系（雷达站位于飞镖正上方，且在飞镖打靶单项赛可以上场）的兵种，可以为飞镖做的远远不应该是一个简单的回放功能。这个功能，归根结底，是雷达站识别与定位、飞镖控制都不成熟的背景下，一个妥协的产物。

裁判系统信息显示

在雷达站的自定义 UI 上，通过读取裁判系统的数据，我们展示了机器人、前哨战和基地的血量，以及比赛阶段和倒计时等信息，方便云台手快速反应。

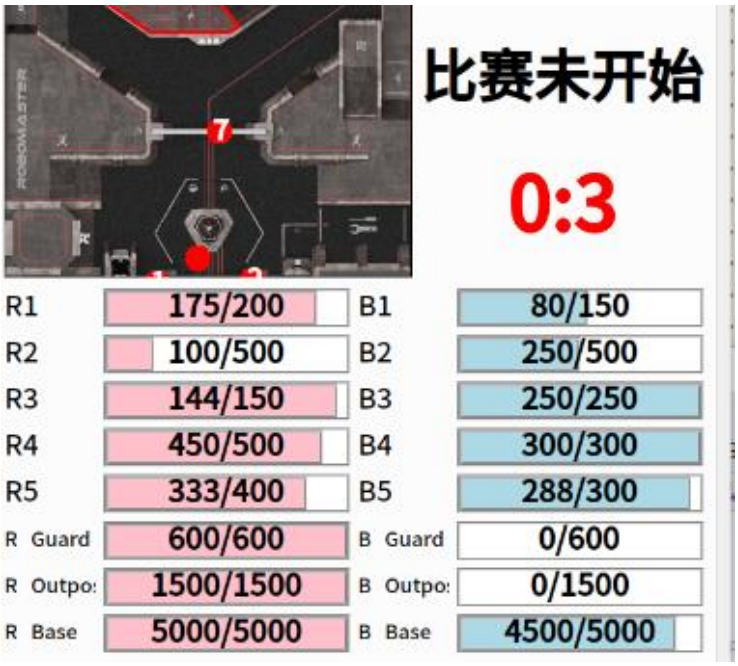

图2-20 裁判系统信息显示

机器人血量可以根据我方阵营，自动调整颜色，以始终保持我方机器人血量在前，敌方机器人血量在后。血条基于 Qt5 的 progressBar 实现。同时，通过获

取到的血量信息，通过遍历该兵种所有可能的血量值，便可以粗略估计出该车辆的最大血量，实现血条的效果。当未能成功获取机器人血量信息时，机器人血条显示为繁忙状态，如图所示。

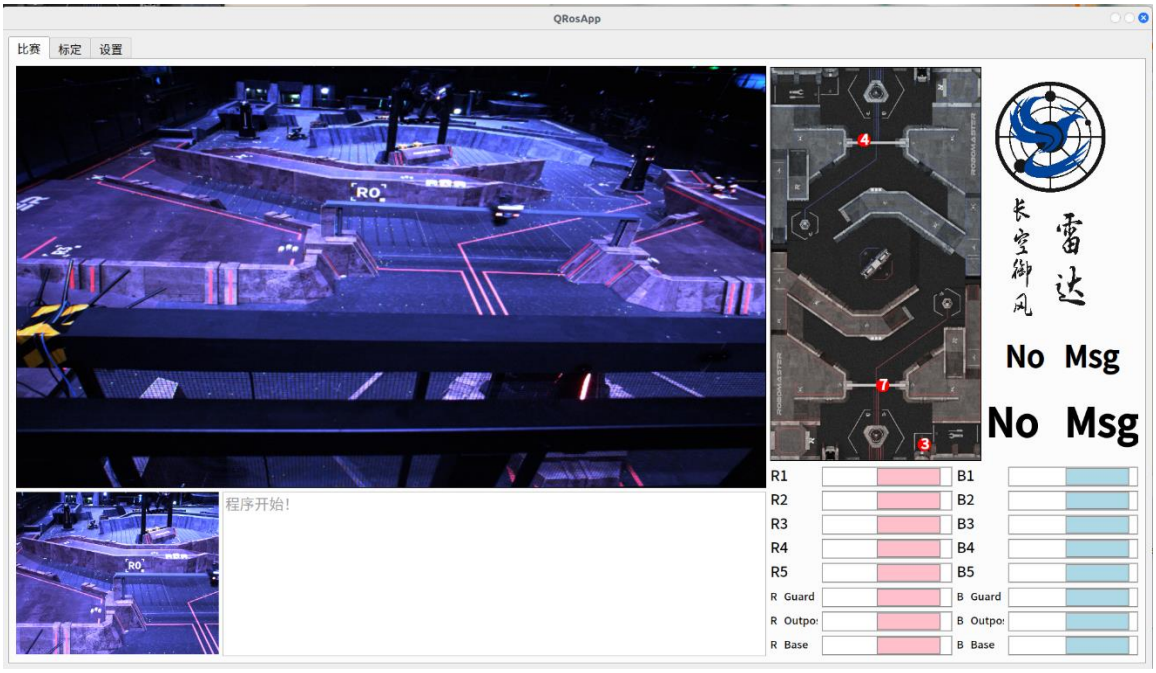

图2-21 血条的繁忙状态

相机位姿估计

相机位姿估计是雷达站三分钟准备阶段需要做的最为重要的工作，直接关乎整场雷达站的表现。所以方便雷达上场人员方便、快捷、准确的做好相机的位姿估计工作至关重要。

考虑到如果只对一个相机进行位姿估计工作，需要经过层层坐标变换才可以获得另一个相机的位姿信息，容易造成误差的放大。所以我们选择对两个相机分别进行位姿估计。赛前工作量增多，便对 UI 操作的流畅性提出了更高的要求。

我们选择了使用 R0 上方两个角点、前哨站最高点（左右相机分别为地方和我方前哨站）和敌方基地最高点进行 PNP 解算，使各个点坐标跨度尽可能大，以提高 PNP 解算的精度。

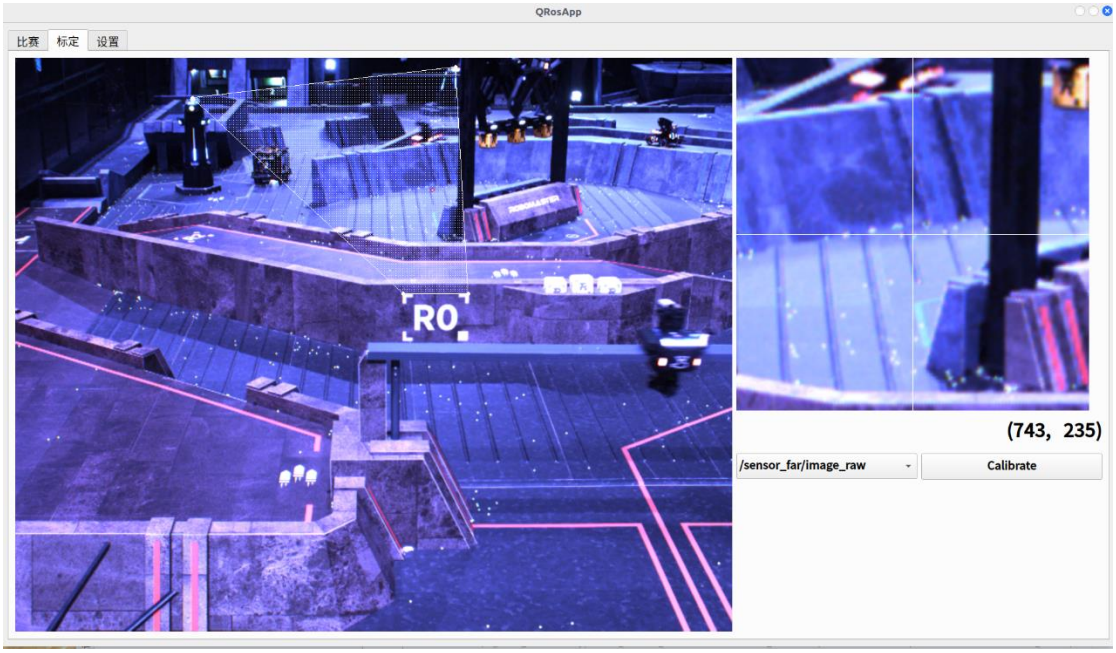

图2-22 左相机 PNP 选点

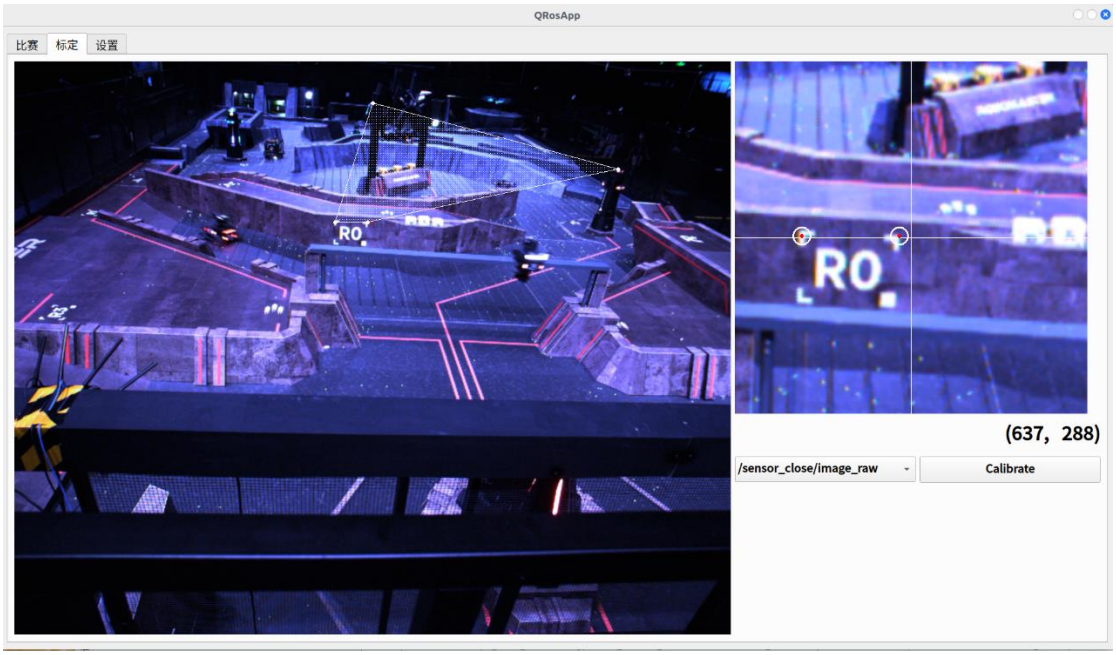

图2-23 右相机 PNP 选点

雷达站 UI 启动后，会首先加载默认的位姿估计结果，并使用该数据进行正常的解算。通过 Qt 的选项卡功能，可以从主页面切换到位姿估计页面。主窗口上绘制了默认的参数。通过鼠标拖动四个角点，即可调整该点的坐标。右侧小窗口会对鼠标的中心位置进行实时放大，并以十字叉丝标注鼠标中心所在位置，放大倍数可以在参数文件中轻松修改。同时，屏幕上实时显示鼠标所在坐标，当鼠标按下后，坐标变为红色，以提示雷达操作人员。

在所有点调整完成后，点击“calibrate”即可保存当前结果，或者通过下拉菜单切换相机自动保存。在两个相机全部标定完成后，便可通过选项卡切换回主

页面。

相机的位姿估计可以随时重新进行，只需重复上述操作，新的结果便可以覆盖旧的结果。同时，由于相机位姿估计是雷达站唯一需要手动进行的操作，我们基于 ROS 的参数服务系统，对所得结果采取了充分的防丢措施。当点的位置改变时，UI 会实时将点的位置保存到参数服务器中，通过这样的操作，当 UI 或者小地图节点意外重启时，原先的结果会自动加载，无需重新进行位姿估计，增强了系统的鲁棒性。

4. 代码仓库

https://github.com/nuaa-rm/radar_station2022

5. 参考文献

1.高翔、张涛：《视觉 SLAM 十四讲》

2. 坐标系变换 & 坐标变换 - 赖东风的文章 - 知乎 https://zhuanlan.zhihu.com/p/274976956

3.OpenCV-4.5.4 文档

4.Robomaster 2022 超级对抗赛比赛规则手册

5.裁判系统学生串口手册 V1.3

6.tensorrtx-GitHub 仓库及其使用方法：https://github.com/wang-xinyu/tensorrtx

7.上海交通大学 RM2021 雷达站开源报告

长空御风 雷达

王书钰 1134643765@qq.com
刘建航 1358446393@qq.com
南京航空航天大学 长空御风