第十五届全国大学生智能汽车竞赛

人工智能创意赛（预赛）

技 术 报 告

学　　校：　　长安大学

队伍名称：第一次 train 模型，显卡不炸就算成功

参赛队员：毕晨晓，肖凌宇，刘腾龙，毛帅，王隽雨

带队教师：王畅，孙秦豫

关于技术报告和研究论文使用授权的说明

本人完全了解第十五届全国大学生智能汽车竞赛关保留、使用技术报告和研究论文的规定，即：参赛作品著作权归参赛者本人，比赛组委会和赞助公司可以在相关主页上收录并公开参赛作品的设计方案、技术报告以及参赛模型车的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名：

王儒雨

毕晨晓

毛帅

刘腾龙

贾法宇

带队教师签名：

日　　期：　2021.8.24

第一章 方案概述

1.1 比赛简介

第十六届全国大学生智能汽车竞赛创意组——百度智慧交通的比赛以“封狼居胥”为主题，基于车道定位及识别、标志与指示牌的检测和识别、目标物检测与识别等多种人工智能技术，打造超越以往以遥控、传感器和机械结构为主的竞赛形式，契合人工智能时代发展特点，充分调动学生的创新、创造活力。比赛开始时，智能车从起点线出发，沿着车道线行驶、行进途中需要在特定地点识别标志物，并完成指定任务，当越过了终点线后，视为完成了行驶任务。最终根据每个参赛队的总得分和时间进行排名。

1.2 方案概述

比赛需要智能车在百度飞桨框架上完成任务，比赛任务大致可分为：车道线识别、标志物检测和指定动作执行三项。为完成上述三个任务，我们在严格遵守使用大赛组委会规定的百度 EdegBoard 算力板及 25GA 闭环电机的基础上，通过购买套件和自主设计等方式完成了车模搭建工作。如图 所示，车模使用 usb 摄像头采集正、侧面信息，进行车道线和正侧面标志物的识别；使用百度 EdegBoard 算力板作为主控，运行 Linux 系统负责进行摄像头采集数据的预处理、与 FPGA 运算单元的数据交换以及向 Wobot 控制器下达控制指令等任务；使用 25GA 闭环电机驱动车模行驶，使用 25GA 闭环电机和数字舵机控制机械结构，根据指令完成规定任务。

在完成车道线识别任务时，我们采用了传统视觉中提取中线的算法和深度学习算法相结合的方案。中线提取算法是以 opencv 库函数为基础，使用 RGB 转 HSV 的算法对图像处理后，进行图像二值化得到黄色赛道边线，提取出赛道中线。运用方向 PID 算法，控制智能车寻赛道中线行驶。由于在赛道中颜色并非完全二值化，且赛道元素复杂，完全使用传统提取中线算法部分赛道元素无法通过，对此，我们对官方提供的车道线预测模型进行了一些优化，并用手柄采集了大量车道线数据，并进行了人工修正，在 Ai Studio 平台的算力支持及 paddle 框架的帮助下构建并训练得到卷积神经网络模型，作为传统中线算法的参考依据，如果传统中线算法与神经

网络模型得到的预测值偏差过大，则认为传统中线算法在当前赛道上的识别出现问题，采用神经网络得到的预测值。

标志物识别任务包含地面标志物识别和侧面标志物识别两部分，是完成“出征” “激战” “宿营” 和 “补充粮草” 任务的基础。在考虑硬件条件后，我们制定了以地面标志物识别为主、侧面标志物识别为辅的策略：我们首先采集了大量的地面标志及侧面建筑的图片，人工使用”Labe Me”标记后，使用官方提供的 SSD 神经网络模型，在 Ai Studio 平台上训练，得到训练模型。之后，智能车在行驶的过程中，不断检测地面标识，如果发现较靠中心的地面标识，则首先依据地标通过深度学习结合边缘检测的办法调整车速姿态，再进行侧面建筑识别并执行对应标志任务的算法。

在硬件层面，除了鲸鱼机器人提供的基础版智能车，我们还为智能车安装了 “麦伦” 以完成车身姿态矫正及 “宿营” 任务中智能车的横向平移， 三自由度机械臂完成 “封狼居胥” 任务中方块的抓取，PPC 管以完成 “补充粮草” 任务中小球的投放。

在上述方案下，我们对智能车进行了硬件安装及软件开发，并对官方给出的代码进行了重构。该方案下，智能车在巡航时有较好的表现：电机速度从 20 ~ 80 都能在赛道上稳定巡线，并且智能车在直道行驶时基本保持车身居中且不抖动，弯道行驶时能根据不同的弯道情况平稳驶过弯道。智能车在执行各项任务时，侧面建筑类型识别精确，位置识别稳定，在 “举旗” 任务时有较好表现；由于对麦伦进行了良好封装， “宿营” 任务表现完美； “激战” 任务中靶心命中率达到 85%以上；安装的机械臂在 “封狼居胥” 中表现良好，粮草在不同的位置时能够稳定抓取，测试中，约 80%抓取两个，10%抓取一个； “粮草” 任务中，由于我们设计并安装了 PPC 管，小球能够沿管道平稳滑落进粮草框，90%以上倒入 4 个球。

第二章 问题描述

2.1 智能车行驶

百度赛道的车道地图较传统智能车比赛赛道更为复杂，在实际处理时主要存在如下的问题：

1. 车道线易丢失（车道宽、摄像头范围窄）、赛道颜色多样、赛道亮度变化大，导致传统方法提取车道线存在很大限制的问题。
2. 智能车由于还需要进行地面标志物识别、侧面标志物识别等任务，且巡航时预测转角值需要在一定频率以上，否则难以维持智能车转向的灵活性。无法采用运算量大、处理时间长的大型网路的问题。
3. 在采集数据时，由于手柄精度有限及遥控人的智能车遥控精确度限制，往往难以获得平滑、精确的转角值的问题。

2.2 标志识别

由于智能车与目标物的相对位置不断变化，摄像头拍摄角度抖动等，导致标志物在摄像头内的大小、形状特征会随着智能车的移动而不断发生改变，且标志物外侧的背景不断改变。在传统方案下，对于地面标识，想要对一系列相似的地面标志进行精确区分并准确判定其位置是难以实现的。而对于侧面标志，除了靶标在特定的背景条件下能够通过识别圆形区域获取，其他的侧面标志由于从三维空间映射到二维照片上时有极大的可变化性，如果使用传统算法实现，不仅保证不了分类的准确性，也无法保证位置识别的精确度，更难以保证在不同背景条件下算法的稳定性。

2.3 任务执行

我们购买了鲸鱼机器人提供的基础版智能车，由于缺乏一些配件及实际执行时，我们存在以下问题：

1. 智能车在执行任务前由于巡航不准确或刚刚过弯导致车身姿态不确定、与靶标位置不确定，导致后续执行任务时精度不足、容易出车道线的问题。
2. 如何将小车准确的移动到任务指定位置的问题。
3. 基础版车身高度不足，侧摄像头安装后拍摄高度不足导致侧面建筑有些(靶标)过高，有些(粮草篮)过低的问题。
4. 基础版缺乏抓取方块的机械件、倒小球的机械件等问题。

第三章 技术方案

3.1 智能车行驶方案

3.1.1 行驶方案概述

针对智能车行驶问题，传统的中线提取方法在本比赛中并不能较好的完成智能车巡线任务，对此我们参考官方的方案，经过数据采集和训练，将得到的巡线模型投入使用，我们发现神经网络方案虽然能跑完整条赛道，但是我们对其在智能车行驶时，智能车位置的居中和车身姿态的保持并不满意。

由于传统中线方案在直道表现良好、神经网络能处理复杂弯道，对此我们提出了传统中线提取结合神经网络的巡线方案。该方案设计如下：

我们在每一次进行巡线预测时，将 OPENCV 的巡线预测值与神经网络的巡线预测值进行对比，如果差距较小，则说明该帧 OPENCV 并没有出现丢线等情况，此时我们采用 OPENCV 得到的预测值；如果差距较大(绝对值大于阀值)，说明该帧 OPENCV 存在丢线或是光线、噪点等其他因素的干扰，此时我们采用神经网络的巡线值。

在上述方案下，智能车在直道时能保证智能车抖动小、姿态稳定、车身居中，且在车道线丢失、光线变化、图像噪点等情况下有良好的适应性。

另外，神经网络在数据采集时，存在手柄精度和人工误差的问题，对此，我们重新设计了数据采集程序，并且把原始赛道分成许多小段，分段进行数据采集。由于每一段的赛道长度很小，在数据采集开始前，操作者可以提前预测手柄大致方向，减少了人工误差。数据采集结束后，我们人工对异常数据进行了清理。

3.1.2 中线提取算法

首先在 HSV 空间中提取图像的黄色部分，再通过从参考行中心往右和往左分别得到两个像素点，再将像素点的横坐标相加除以二，与图像横轴正中心坐标(320)做差，将差值与差值的平方通过 pid 算法叠加算出巡航拐角值。详细代码见附录。

3.1.3 神经网络模型优化*

在 Ai Studio 上，官方给出的巡线模型参数大小大致为 7M 左右，该模型卷积层数少、缺乏池化层，我们在官网模型的基础上，加入了一些卷积层和池化层，并对不同大小的模型进行测试，参数 20M 左右的模型大致能达到 15 帧，该帧数下能保证智能车转角时的灵活性。

最终结合中线提取算法，构建巡线模型。代码见附录。

3.2 标志识别方案

3.2.1 标志识别方案概述

针对标志识别问题，可以发现传统的图像做法实现目标检测算法是十分困难的，对此，我们决定采用深度学习的方案。借用 Ai Studio 智能小车检测上所提供的 Mobile Net 网络模型，我们大量采集地面标识及侧面建筑图像数据，并使用“Label Me”软件对数据进行标注。标注后，使用官方提供的数据转化程序将标注的数据转化为指定格式，在 Ai Studio 平台上进行训练得到模型。

3.3 任务执行方案

3.3.1 任务执行方案概述

针对任务执行问题，对于智能车在行驶时由于各种因素导致在行驶到任务点前时车身位置不居中，姿态不平直的问题，我们提出了一种基于深度学习和边缘检测算法的车身姿态调整方案。该方案能精确地控制车辆与地标的相对位置。通过神经网络讲识别出的地标从图像中提取出来，再对提取的小图像使用边缘检测算法，之后，对于每一个边界，我们用 n 变形最适拟合并寻找一个四边形且面积大于阀值（500）的边界，再将该边界的最下方两个端点提取出来，根据最下方两个端点在图像中的位置来调整车身姿态。

车身姿态调整完毕后，再根据识别的地标执行对应的程序。

在进行侧面识别时，由于机械臂、靶笔等结构与侧面摄像头的相对位置固定，我们首先确定智能车刚好完成任务时，目标检测后侧面建筑物在侧摄像头中的位置，并将该位置记录

为标志位置，智能车在执行任务时，会缓慢前进并比对目标检测后物品当前位置与标志位置之间的差，当接近标志位置后，智能车立刻停止，并进行微调。

3.3.2 车身姿态调整算法*

该算法的流程如下:

姿态调整算法

1 $k \leftarrow 1, k_{lim} \leftarrow 0.01, x \leftarrow 100, x_{lim} \leftarrow 3, y \leftarrow 100, y_{lim} \leftarrow 5$

2 **while(** $abs(k) > k_{lim}$ **or** $abs(x) > x_{lim}$ **or** $abs(y) > y_{lim}$ **) do**

3 img $\leftarrow$ **read( )** 前摄像头图片

4 $border$ $\leftarrow detect(img)$ 神经网络预测标识的边界

5 img $\leftarrow$ **img(border)** 提取神经网络标记的图像块

6 $rectangle$ $\leftarrow extraction(border, img)$ 提取图像块中的四边形

7 $bottom\ point$ $\leftarrow bottom(rectangle)$ 提取四边形最下方两个顶点

8 k, x, y $\leftarrow adjust(bottom\ point)$ 根据最下方两个顶点调整车身并更新

9 **end while**

详细代码见附录。

3.3.3 目标移动算法

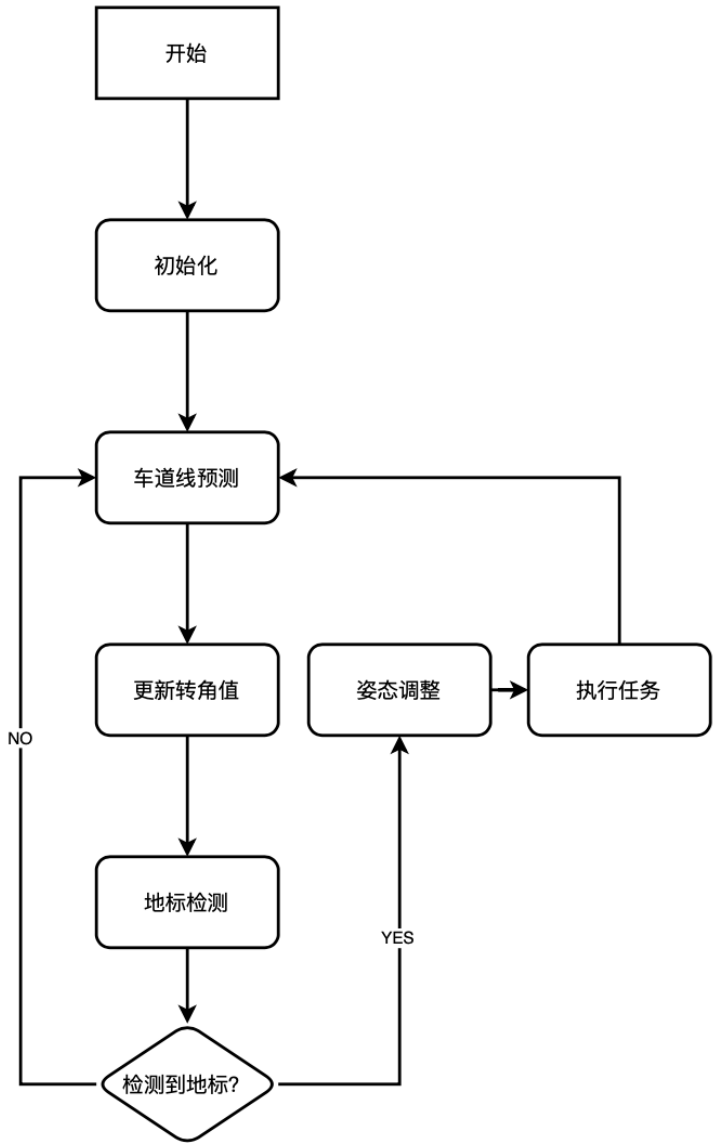
该算法流程如下：

目标移动算法

```
1  stander ← 0.5, border ← 0
2  car.speed ← 12                          缓慢前进
3  while(abs(stander − border) > 0.01) do
4    img          ← read( )                侧摄像头图片
5    border       ← detect(img)            神经网络预测标识的边界
6  end while
7  car.speed ← 0                           停止
8  border       ← detect(img)              神经网络预测标识的边界
9  if (abs(stander − border) > 0.03) then    挺稳后略有偏差
10   car.speed ← 12 if stander − border > 0 else −12
11   time.sleep(0.05)
12 end if
13 car.speed ← 0                           停止
```

第四章 方案实现*

4.1 方案软件运行流程



4.2 车道线数据采集优化

神经网络在数据采集时，存在手柄精度和人工误差的问题，对此，我们重新设计了数据采集程序，并且把原始赛道分成许多小段，分段进行数据采集。由于每一段的赛道长度很小，在数据采集开始前，操作者可以提前预测手柄大致方向，减少了人工误差。数据采集结束后，我们人工对异常数据进行了清理清洗了一些在程序刚开始和刚启动时产生的预测值和图片不对应的图片及数据。

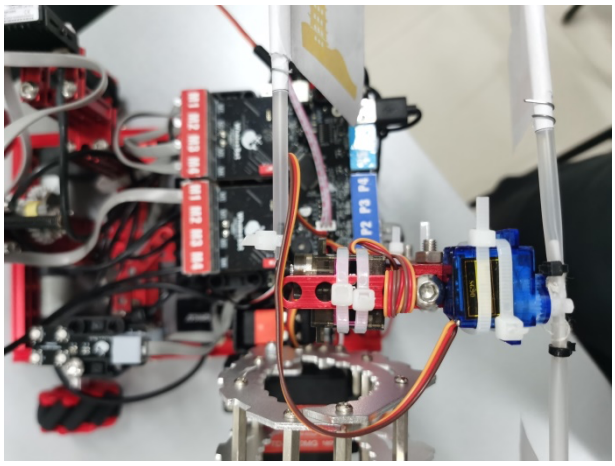
另外为了能够方便进行分段数据采集，我们在官方提供的代码基础上进行了一些改动，代码能实现自动创建、更新文件夹，以及摁下”R”键暂停采集，”L”键重新开始采集的功能。详细代码见附录。

4.3 标志、建筑数据采集优化

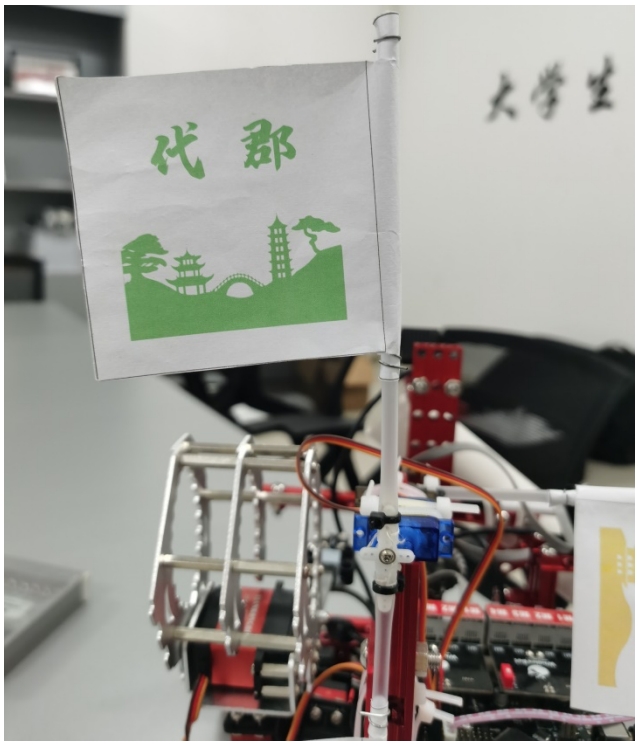
为了模拟真实的行车环境，我们对官方提供的数据采集程序进行了一些优化，能够实现智能车在巡线时自动采集侧面数据，这样的数据更加模拟智能车行驶时真实出现的数据，以及摁下”R”键暂停采集，”L”键重新开始采集的功能。详细代码见附录。

4.4 举旗

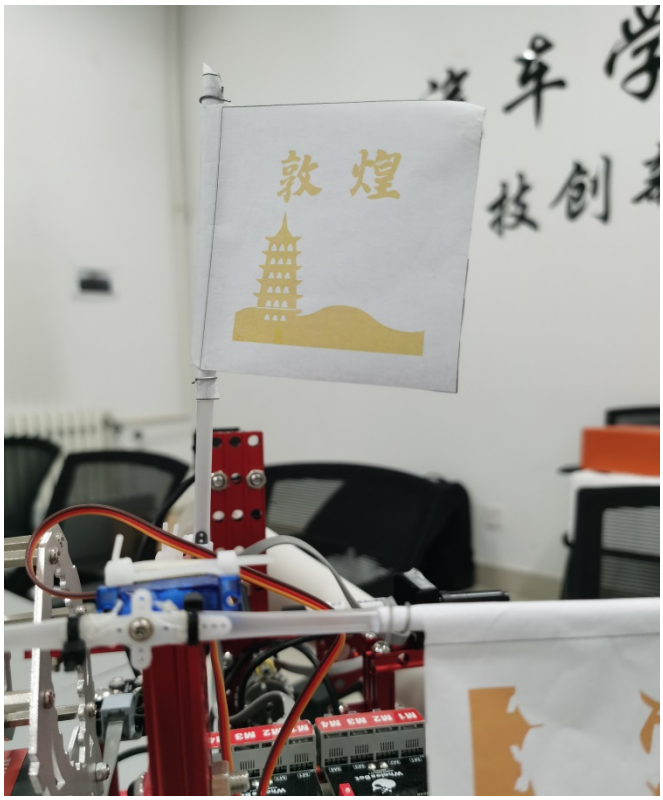
任务“出征”要求智能车在每座城池前举起带有相应标志物的旗帜，并亮绿灯三次。实现举旗任务的机械结构由自主搭建而成，用到了可转动 180°的 9g 数字舵机两个及舵盘、智能车结构件、扎带和扎丝，使用结构件拼成舵机支架，用螺丝固定在 Wobot 控制器支架上，两个舵机使用扎带一前一后绑到舵机支架上，使用扎丝、扎带分别把旗子、舵盘与旗杆连在一起，舵盘固定在舵机上，如图①所示。当识别到城堡时，舵机接收到来自主控和 Wobot 控制器的信号后，转动固定角度，完成举旗任务，调整舵机支架高度以保证执行举旗任务时，旗子的最低点高过车身最高点，如图②、③是“代郡” “敦煌”旗子举起时的状态。



①



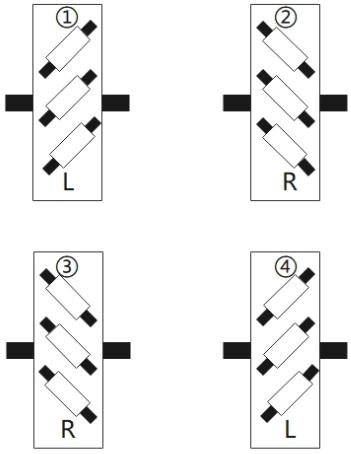
②



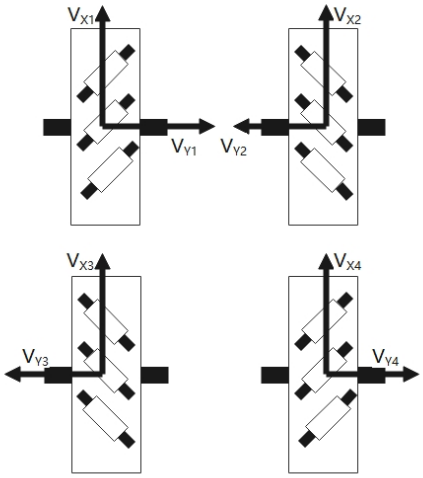
③

4.5 麦克纳姆轮移动

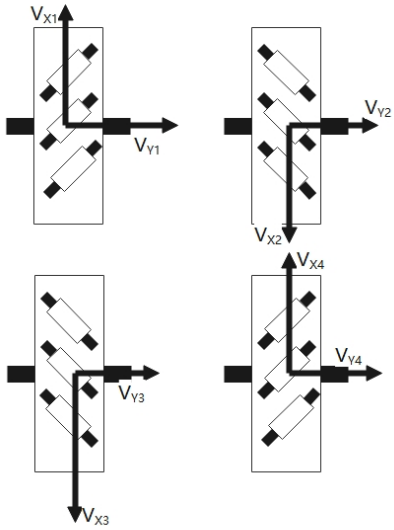
我们选择了一套可以全向行驶的麦克纳姆轮（下简称麦轮）作为智能车车轮，相比于原始的车轮套件，麦轮可以进行智能车车身姿态微调，并且可使任务“宿营”的执行更加简单。四个车轮分别记为 1、2、3、4 号，以左旋、右旋、右旋、左旋的顺序安装，如图④所示；根据麦轮的运动解算，当四个车轮转速大小相等、方向相同，如图⑤所示时，1、2 车轮侧向车速相互抵消，3、4 车轮侧向车速相互抵消，而且它们纵向车速相同，于是智能车可以以一定速度（各轮纵向车速）直线行驶；当改变各轮转速大小、方向时，可以实现弯道行驶；当 1、3 轮转速大小相等、方向相反时，纵向车速相互抵消，侧向车速相等，同时 2 轮与 3 轮转速相等、4 轮与 1 轮转速相等，2、4 轮也会抵消纵向车速，提供与左侧车轮相等的侧向车速，如图⑥所示，驱动智能车横向行驶，这在执行“宿营”任务时，可以实现平移进入营地，更加简洁高效。



④



⑤

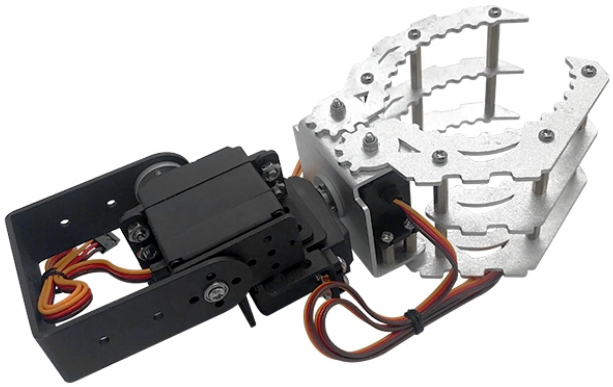


⑥

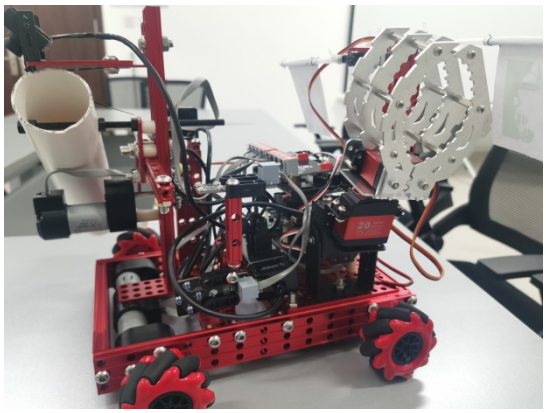
为了将麦轮安装到 25GA 闭环电机上，我们对电机进行了改造：用角磨机将电机轴切短 1cm；使用铝皮缠绕加粗电机轴，以保证电机轴与麦轮联轴器在运动时同轴转动。

4.6 方块抓取

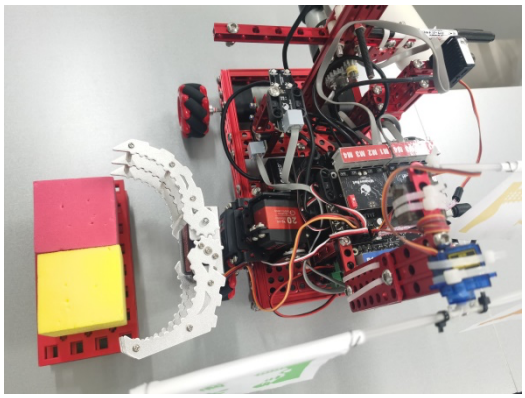
任务“封狼居胥”要求智能车抓取指定地点的海绵方块战利品，并携带回基地。完成这一任务的机械装置使用了一个三自由度的机械爪，如图⑦所示。机械爪使用螺丝固定在智能车左侧，从基地出发时保持直立状态，如图⑧所示；当到达任务区域、接收到来自主控和 Wobot 控制器的信号后舵机带动机械爪先后执行伸出、张开和抓取动作，获得“战利品”，如图⑨所示；随后机械爪收起，恢复直立状态，携带战利品回到基地，如图⑩所示。



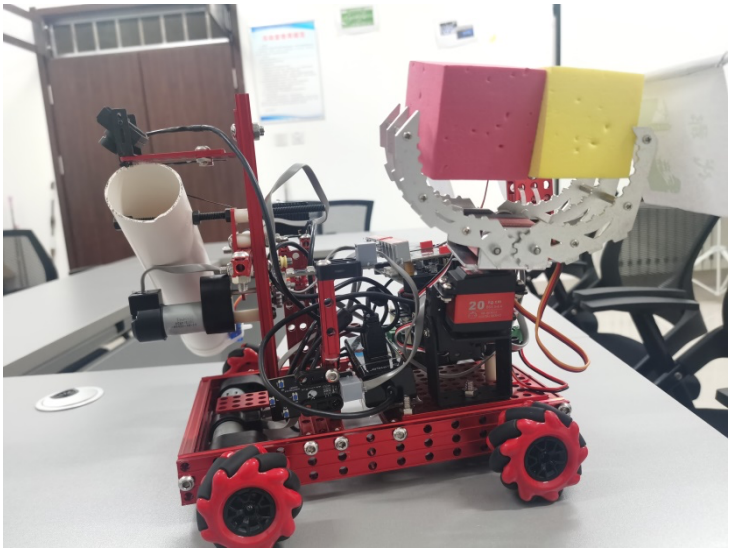
⑦



⑧



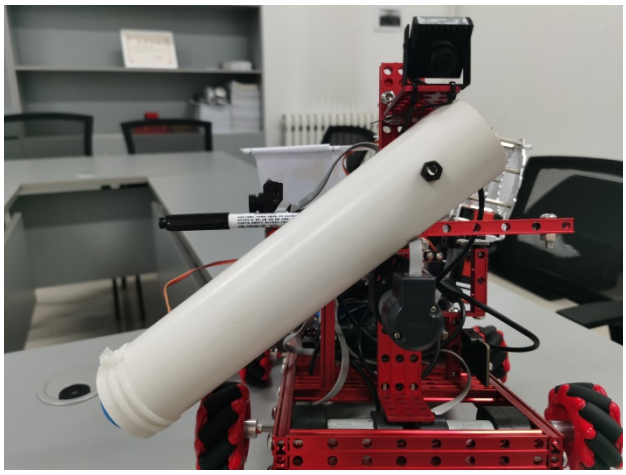
⑨



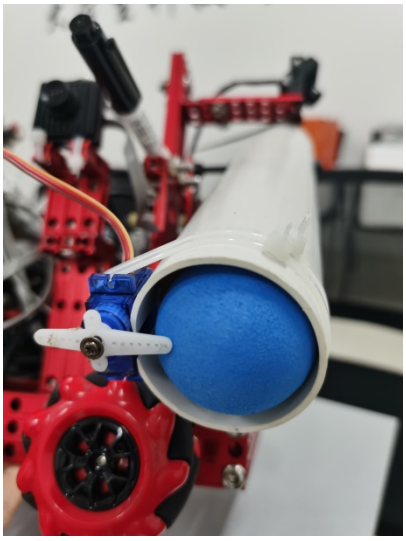
⑩

4.7 粮草投放

任务“补充粮草”要求智能车将出发时携带的 EVA 小球运送到指定位置的“兵营”中。受限于车模狭小的空间，我们在设计机械机构时采用了一节 PVC 塑料管，将它打孔后以一定角度倾斜安装在摄像头支架梁上，使用螺杆螺母进行固定，如图⑪所示。当小车从基地出发时，将 4 个小球依次放置在管中；管口处使用扎带固定一个 9g 数字舵机，在小车正常行驶时依靠舵盘阻拦管中小球自运动，如图⑫所示。待到任务执行处，舵盘旋开，小球沿倾斜的塑料管滑落，同时小车前后移动，实现小球的精准落位。



⑪



⑫

第五章 测试分析

5.1 巡线网络性能分析

在巡线时，既要兼顾准确性与性能，网络越复杂，可能能够得到更好的训练模型，但是智能车算力有限，不可能在智能车上运行过大的网络，否则会导致智能车更新转角值帧数过低，无法灵活转弯等情况。对此，我们设计了一系列的巡线网络，并进行测试分析。

Table 1 网络性能对比

编号	构建方式	参数大小	最终 Loss	运行帧数	运行情况
1	卷积	7.8M	0.08	31	直道不稳定，弯道出线
2	卷积(大)	24.6M	0.07	15	直行稳定，但是直道也出线，弯道稳定
3	卷积 + 池化	18.6M	0.04	18	不出线，直行存在抖动
4	卷积 + 池化 (大)	45.3M	0.03	10	不出线，直行抖动
5	残差块	100M	0.006	3	帧数过低，出线

训练时 Loss 曲线 (池化 + 卷积)：

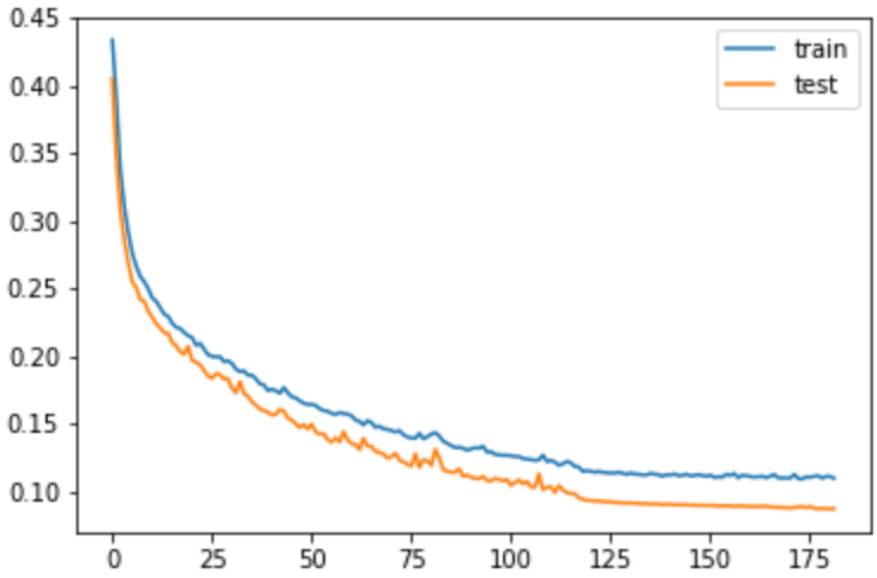


Figure 1 训练过程 loss

最终选用了 18M 的池化+卷积搭建的网络模型，该模型在帧数和稳定性上都有较好表现。

5.2 麦轮稳定性分析

由于安装的麦轮并非官方原配轮，我们需要验证麦轮在不同环境下前行和平移的稳定性是否达标，防止比赛时出现意外情况。我们进行了多组测试，图表中内容为平均值。

地图下铺设类型	直行距离(m)	水平偏差(m)	偏差率
瓷地板	2.55	0.06	2.3%
软地毯	1.22	0.02	1.6%
床垫	1.81	0.07	3.8%

基本可以认为麦轮的平移比较稳定。

第六章 作品总结

本次智能车大赛我们在第一轮中拿到 0 分，第二轮中拿到 500 分。第一轮比赛中，在试车时，发现车子跑不稳，尤其是在姿态调整的过程中，车辆不能进行正确的调整，最初我们以为是麦轮在赛场场地上表现不佳，自己平时用的赛道比较脏，灰尘很多，导致摩擦力比赛场赛道大一些，于是我们对智能车的各项参数进行调整，调了很久还没有调好，导致队员信心丧失，而我们封车前五分钟举旗的舵机与旗杆突然分离，没有调整好安装位置，上场后旗杆位置与原始位置偏差太大，举旗任务失败。调整过的参数让智能车姿态调整的时候与预期位置偏差过大，每项任务都没完成，还出线了。

我们没有准备一套纯巡线的方案，失策。回来以后拿赛场上拍下来的图片做了一些测试，发现是因为场上亮度比平时亮度高太多了，在颜色空间转化的时候要稍微改把鲜艳度给高一点，把参数改回了备份的参数，然后第二轮直接上去跑就满分了，很可惜。

本次参赛还是第一轮试车时车子跑的效果很差，队员心态都很差，钻牛角尖调整车辆移动相关的参数去了，实际上冷静下来想一想就能想到该把前摄像头拍下来的图片仔细研究研究，另外我们队没有准备第二套保底方案，属实是参赛经验不足。希望今后参赛能继续改进。

对于智能车本身，经过姿态调整后，各项任务都能十分精确的完成，主要问题在于姿态调整所用的时间过长，在微调时，电机的启动速度过大，达不到微调的精度要求，导致微调时车身一直在反复抖动，进一步改进的话可以把麦轮的摩擦与地面的摩擦系数减少，来降低智能车在微调时的速度，以满足微小调整时对智能车低速的要求。

另外，智能车的速度还可以加快，因为在平时测试时，智能车能做到 80 速度不出线，但是比赛时由于第一轮的后怕，第二轮以一个很低的速度(35)行驶。

对于前摄像头丢线的问题，可以通过 USB 外接一个广角的前摄像头，这样在预测中线时会有更好的表现。

附录

附录A 中线提取

```python
def get_angle(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    low_hsv = np.array([17, 41, 40])
    high_hsv = np.array([25, 255, 255])
    mask = cv2.inRange(hsv, lowerb=low_hsv, upperb=high_hsv)
    cv2.imwrite("../image/line.png", mask)
    left = right = None

    for i in range(320, 0, -1):
        if mask[450][i] == 255:
            left = i
            break

    for i in range(320, 640):
        if mask[450][i] == 255:
            right = i
            break

    if right is None and left is None:
        right = 640
        left = 0

    if left is None:
        left = -50

    if right is None:
        right = 710

    mid = (left + right) / 2
    p_one = 2.1
    p_two = 2.8
    one = (mid - 320) / 320 * p_one
    two = ((mid - 320) ** 2) / (320 ** 2) if one > 0 else -((mid - 320) ** 2) / (320 ** 2)
    two = two * p_two
    return one + two

```

附录 B 中线算法+神经网络

```
img = self.front_cam.read()
angle = self.cruise(img)
cnn_angle = self.cnn_cruise(img)
if abs(cnn_angle - angle) > 0.5:
    angle = cnn_angle
print("angle:", angle)
self.cart.set_steer(angle)
```

附录C 姿态调整

```
# full_adjust.py
import time

import config
from units import *
from units.adjuster import get_points

cart = Cart()
front_cam = Camera(config.front_cam)
front_cam.start()
front_dec = Detector('front')


def adjust_k():
    img = front_cam.read()
    res = front_dec.infer(img)
    k = 1
    if len(res) == 0:
        cart.set_speed(-15)
        time.sleep(0.03)
        cart.stop()
    else:
        bottom_points = get_points(img, [res[0].left, res[0].top,
res[0].right, res[0].bottom])
        if bottom_points is None:
            cart.set_speed(-15)
            time.sleep(0.03)
            cart.stop()
        else:
```

```
            k = -(bottom_points[1][1] - bottom_points[0][1]) / (bottom_points[1][0] - bottom_points[0][0])
            if abs(k) < 0.001:
                return k
            if k > 0:
                speed = max(20 * k + 11, 11)
            else:
                speed = min(20 * k - 11, -11)
            print("k_speed", speed)
            cart.round(speed)
            # origin
            # time.sleep(abs(k * 0.75))
            time.sleep(abs(k * 0.82))
            cart.stop()

    return k


i = 0


def adjust_x(stander, lim=4):
    global i
    # adjust_k
    img = front_cam.read()
    res = front_dec.infer(img)
    x = 100

    if len(res) == 0:
        return x
    else:
        bottom_points = get_points(img, [res[0].left, res[0].top, res[0].right, res[0].bottom])
        if bottom_points is None:
            return x
        else:
            i = 0
            now_mid = (bottom_points[0][0] + bottom_points[1][0]) / 2
            print("x_mid", now_mid)
            delta = now_mid - stander
            x = delta
            if abs(delta) <= lim:
                return x
```

```
            else:
                delta_speed = 0.12 * delta
                speed = 10.5 + delta_speed if delta > 0 else -10.5 +
delta
                if speed > 15:
                    speed = 15
                elif speed < -15:
                    speed = -15
                print("x_speed:", speed)
                cart.right_move(speed)
                time.sleep(min((abs(delta / 2)) * 0.011, 0.15))
                cart.stop()
    return x


def adjust_y(stander, lim=3):
    img = front_cam.read()
    res = front_dec.infer(img)
    y = 100
    if len(res) == 0:
        cart.stop()
    else:
        bottom_points = get_points(img, [res[0].left, res[0].top,
res[0].right, res[0].bottom])
        if bottom_points is None:
            cart.set_speed(-15)
            time.sleep(0.03)
            cart.stop()
        # elif min(bottom_points[0][1], bottom_points[1][1]) <
        else:
            now_mid = (bottom_points[0][1] + bottom_points[1][1]) / 2
            print("y_mid:", now_mid)
            delta = stander - now_mid
            y = delta
            if abs(delta) <= lim:
                return y
            else:
                delta_speed = 0.115 * delta
                speed = 10.65 + delta_speed if delta > 0 else -10.65
+ delta
                if speed > 15:
                    speed = 15
                elif speed < -15:
```

```
                    speed = -15
                print("y_speed:", speed)
                cart.set_speed(speed)
                time.sleep(min((abs(delta / 5)) * 0.017, 0.15))
                cart.stop()
        cart.stop()
    return y


def adjust(x_stander, y_stander, k_lim=0.01, x_lim=8, y_lim=3):
    t = time.time()
    front_cam = Camera(config.front_cam)
    front_dec = Detector('front')

    while time.time() - t <= 10:
        adjust_k()
        time.sleep(0.03)
        adjust_x(x_stander, lim=x_lim)
        time.sleep(0.03)
        adjust_y(y_stander, lim=y_lim)
        time.sleep(0.03)
        adjust_k()

        img = front_cam.read()
        res = front_dec.infer(img)
        if len(res) == 0:
            cart.set_speed(-15)
            time.sleep(0.03)
            cart.stop()
            continue
        bottom_points = get_points(img, [res[0].left, res[0].top,
res[0].right, res[0].bottom])
        if bottom_points is None:
            cart.stop()
            print("not find")
            continue
        k = -(bottom_points[1][1] - bottom_points[0][1]) /
(bottom_points[1][0] - bottom_points[0][0])
        x = (bottom_points[0][0] + bottom_points[1][0]) / 2
        y = (bottom_points[0][1] + bottom_points[1][1]) / 2
        print("k, x, y", k, x, y)
        if abs(k) <= k_lim and abs(x - x_stander) <= x_lim and abs(y
- y_stander) <= y_lim:
```

```
            print("Adjust time:", time.time() - t)
            break

    return 0
```

```
# units.adjuster.py
import cv2
import time
import config
import numpy as np
from .cart import Cart
from .camera import Camera
from .detector import Detector


def bottom_two(points, borders):
    x_scale = 640
    y_scale = 480
    points = sorted(points, key=lambda x: x[1], reverse=True)
    points[0][0], points[0][1] = points[0][0] + int(x_scale * borders[0]), points[0][1] + int(y_scale * borders[1])
    points[1][0], points[1][1] = points[1][0] + int(x_scale * borders[0]), points[1][1] + int(y_scale * borders[1])
    return [points[0], points[1]]


def get_points(img, borders):
    x_scale, y_scale = 640, 480
    borders = [max(borders[0] - 0.05, 0.01),
               max(borders[1] - 0.05, 0.01),
               min(borders[2] + 0.05, 0.99),
               min(borders[3] + 0.05, 0.99)]

    img = img[int(y_scale * borders[1]): int(y_scale * borders[3]), int(x_scale * borders[0]): int(x_scale * borders[2])]

    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # low_hsv = np.array([0, 0, 190])
    # high_hsv = np.array([185, 221, 255])
    low_hsv = np.array([0, 0, 185])
    high_hsv = np.array([190, 221, 255])

```

```
    mask = cv2.inRange(hsv, lowerb=low_hsv, upperb=high_hsv)

    # 边界提取
    _, contours, hier = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for cnt in contours:
        cnt_len = cv2.arcLength(cnt, True)  # 计算轮廓周长
        cnt = cv2.approxPolyDP(cnt, 0.02 * cnt_len, True)  # 多边形逼近
        if len(cnt) == 4 and cv2.contourArea(cnt) > 600 and
cv2.isContourConvex(cnt):
            # M = cv2.moments(cnt)  # 计算轮廓的矩
            # cx = int(M['m10'] / M['m00'])
            # cy = int(M['m01'] / M['m00'])  # 轮廓重心
            cnt = cnt.reshape(-1, 2)
            # print(cnt)
            # max_cos = np.max([angle_cos(cnt[i], cnt[(i + 1) % 4],
cnt[(i + 2) % 4]) for i in range(4)])
            return bottom_two(cnt, borders)
    return None

```

附录D 数据采集

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
"""
**Collect the cruise data**

| Before you run this module, you need to change the ``result_dir``
and check it is exists.
| The collected data will saved into ``result_dir`` with format of
jpg.
| And the cruise data will be written into ``result.json``
"""


import os
import time

import cv2
import threading
```

```python
import json
import config
from constants import *
from units import JoyStick, Cart, Camera

# 18  130 139
result_dir = "/home/root/workspace/autocar/image/cruiser"


class Logger:
    def __init__(self):
        self.camera   = Camera(config.front_cam)
        self.stopped_ = False
        self.paused   = True
        self.epoch    = 141
        self.counter  = 0
        self.map = {}
        self.result_dir = result_dir
        # self.result_dir =
"/home/root/workspace/autocar/image/solider"
        # self.result_dir = "/mnt/data/train24"

    def start(self):
        if self.paused:
            self.epoch += 1
            self.map.clear()
            self.counter = 0
            self.camera.start()
            self.result_dir = os.path.join(result_dir,
"train{}".format(self.epoch))
            self.check_dir()
            self.paused = False
            cart.set_speed(25)

    def pause(self):
        if self.stopped_ or self.paused:
            return
        self.paused = True
        cart.stop()
        cart.stop()
        time.sleep(0.5)
        path = "{}/result.json".format(self.result_dir)
        with open(path, 'w+') as fp:
```

```
            json.dump(self.map.copy(), fp)
        print("PAUSED")

    def check_dir(self):
        if not os.path.exists(self.result_dir):
            os.mkdir(self.result_dir)

    def stop(self):
        self.stopped_ = True

    def log(self, axis):
        if not self.paused:
            cart.set_steer(axis)
            image = self.camera.read()
            path = "{}/{}.jpg".format(self.result_dir, self.counter)
            self.map[self.counter] = axis
            cv2.imwrite(path, image)
            self.counter += 1
            print("writing to {}".format(path))

    def stopped(self):
        return self.stopped_


def joystick_thread():
    """
    | This thread use to read the buttons from joystick,
    | after read an special button, use special function.
    """
    while not logger.stopped():
        button = js.read()
        if button.type == L1 and button.value == 1:
            logger.start()
        elif button.type == R1 and button.value == 1:
            logger.pause()
        elif button.type == A:
            logger.stop()
        elif button.type == RAXIS:
            if button.number == 2:
                js.x_axis = button.value / AXIS_MAX


def main():
```

```
    t = threading.Thread(target=joystick_thread, args=())
    t.start()
    cart.start()
    while not logger.stopped():
        logger.log(js.x_axis)

    t.join()
    cart.stop()


if __name__ == "__main__":
    js = JoyStick()
    cart = Cart()
    logger = Logger()
    js.open()
    cart.start()
    main()
```

附录 E 标志物采集

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
"""
| This model will collect the marks data.
| Before you run this model you need to change and check the ``basic_save_path``.
| while running, you can press ``L1`` to start, ``R1`` to pause and ``A`` to stop.
| All data will save into ``basic_save_path/mark(i)``
"""

import os
import time

import cv2
import threading
import config
from constants import *
from units import JoyStick, Cart, Camera, Cruiser, Steering


```

```
basic_save_path = "../image/side"


logger_lock = threading.Lock()
stopped = False


class Logger:
    def __init__(self):
        self._paused = True
        self.start_counter = 0
        self.counter = 0
        self.epoch = 0

    def construct_dir(self):
        return os.path.join(
            basic_save_path,
            "mark" + str(self.epoch)
        )
        # "image/marks/mark0"...

    def start(self):
        if self._paused:
            if not os.path.exists(self.construct_dir()):
                print(os.getcwd())
                os.mkdir(self.construct_dir())

            self._paused = False
            cart.start()
            cart.set_speed(20)

    def pause(self):
        if not self._paused:
            cart.set_speed(0)
            self._paused = True
            self.epoch += 1
            self.counter = 0
            cart.stop()
            print("STOPPED")

    def log(self):
        if not self._paused:
            # image = front_cam.read()
```

```
            image = side_cam.read()
            path = os.path.join(self.construct_dir(),
str(self.counter) + ".jpg")
            cv2.imwrite(path, image)
            self.counter += 1
            print("writing to {}".format(path))
        else:
            time.sleep(0.01)

    @property
    def paused(self):
        return self._paused


logger = Logger()


def joystick_thread():
    global stopped
    while not stopped:
        button = js.read()
        # print(button.type)
        if button.type == L1 and button.value == 1:
            logger.start()
        elif button.type == R1 and button.value == 1:
            logger.pause()
        elif button.type == RAXIS:
            if button.number == 2:
                steer = button.value / AXIS_MAX
                cart.set_steer(steer)
        elif button.type == A:
            global_release()
        elif button.type == R2:
            steer3.rotate(40, config.rights['3']['angle'])
            time.sleep(1)
        elif button.type == L2:
            steer3.rotate(40, config.lefts['3']['angle'])
            time.sleep(1)
        elif button.type == TB:
            if button.value == -AXIS_MAX:  # TOP
                steer2.rotate(40, 0)
                time.sleep(1)
            elif button.value == AXIS_MAX:  # BOTTOM
```

```
                steer2.rotate(40, 20)
                time.sleep(1)


# def global_init():
#     front_cam.start()
#     js.open()


def global_release():
    global stopped
    stopped = True
    logger.pause()
    front_cam.stop()
    side_cam.stop()
    cart.stop()


def take_picture():
    while not stopped:
        logger.log()


def cru_threading():
    while not stopped:
        if not logger.paused:
            angle = cruiser.infer(front_cam.read())
            cart.set_steer(angle)


def main():
    front_cam.start()
    side_cam.start()
    js.open()
    js_thread = threading.Thread(target=joystick_thread, args=())
    pic_thread = threading.Thread(target=take_picture, args=())
    cru_thread = threading.Thread(target=cru_threading)

    js_thread.start()
    pic_thread.start()
    cru_thread.start()
    js_thread.join()
    pic_thread.join()
```

```
    cru_thread.join()


if __name__ == "__main__":
    logger_lock = threading.Lock()

    cart = Cart()
    cruiser = Cruiser()
    front_cam = Camera(config.front_cam)
    side_cam = Camera(config.side_cam)
    js = JoyStick()

    steer2 = Steering(2)
    steer3 = Steering(3)

    main()
```