工程实践与科技创新 3F - 05 小组周报

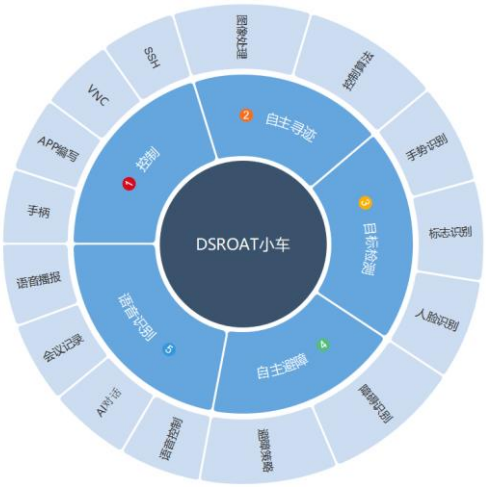
小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 5 月 21 日　　第十三周

这应该是最后一次进行周报的更新了。

本周我们基于助教老师在群里回复具体细节，对代码任务进行微调，主要是将双环巡线变为两圈，阳光下赛道障碍识别后从停车变为换道，并在实验场地进行测试与视频拍摄，两个赛道基本都可以在 30s 左右跑完全部任务。至此，基础任务已收工，代码封存。

拓展任务，上周语音测试发声的 BUG 基本判定为树莓派音频接口输出问题，我们判断此问题是难以解决的，因而采用迂回的策略，将语音发送给 Web 端进行发声。人脸识别 Demo 测试了两种方法，其一速度过慢，帧数过低，另一准确率一般，还在进行最后的测试与优化。



目前，我们计划由专业的同学进行报告撰写、视频剪辑与前端效果展示编写。截至目前，结题报告基本完成了一半内容，其余两项进度需要取决于某两名同学的拖延症能力。

另外，由于某同学计划 16 周周三回家，因而可能需要在 14 周周末或 15 周初进行答辩，我们计划在下周四计控考完后，在 14 周周末，完成 3F 最后的收尾任务。

至此，C05 小组，工程实践与科技创新 3F 基本结题，感谢助教老师的悉心批阅与群里的热心帮助，谢谢！

DSROAT 小车项目，结题！

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 5 月 14 日　　第十二周

本周周四过控考试，处于摸鱼状态，周四考试之后才开始补充做了一些拓展任务。

语音识别的简单任务目前已经完成，完成了语音识别、语音合成、聊天交互等 demo 的代码与测试。目前以多进程的方式编写了语音命令识别，小车已经可以识别语音指令并进行相应的操作（见测试视频）。目前存在的问题，小车的麦克风与音频音箱接口同时使用会出很奇怪的 Bug，原因未知，怀疑是声卡问题，导致音频输出存在问题。

人脸识别 Demo 在自己电脑上测试完成，尚未进行部署。不过按照目前在电脑上的运行速度，预计在树莓派上不会有很好的帧数，但完成基本的识别拓展任务应该问题不大。

基础任务由于周四考完到周五时间仓促，未能测试，还停留在 4.29 版本（老拖延症了）。

趁下面两周无考试时间，计划由部分同学开始最终报告的撰写、视频的剪辑与 PPT 的制作，以减轻期末周的压力。

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 5 月 7 日 **第十一周**

本周五一假期，放假一周。

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 4 月 29 日　　第十周

首先，本周针对上周遗留的停车障碍检测全开造成的时间浪费问题，我们采用全图分类的方法，训练了一个全图分类器，效果出奇的好，基本达到了我们的应用要求。

同时，我们对对速度控制算法进行了微调，使速度连续，且启动较快的效果，，让小车的运行更加连贯。

我们考虑在训练模型时融合半监督的思想，让模型训练更加准确与有效。因而，本周测试时进行了较长的视频录制工作，以半监督的思想进行数据集尤其是负样本的扩充。

目前，巡线、转弯检测、停车检测三者全开的帧数为 25fps 左右，基本达到了应用要求。可以在 30s 左右完成双环巡线的完整流程，在 24s 左右完成走廊尽头赛道的巡线流程。

计划在下一周进行最后一次的测试，之后几周进行拓展模块的开发测试以及报告的撰写。

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 4 月 21 日　　第九周

本周主要进行了分类器数据增强、停车标志检测、代码框架重构等任务，并前往实验室进行调试。在场地测试的结果是，可以在 35s 左右完成双环场地的巡线、转向与停车的任务。

本周首先对分类器的训练补充进行了数据增强，考虑光线、倾角等的影响，对原有数据集做了对比度微调、亮度随机调整、倾斜一定角度、椒盐噪声等一系列的数据增强，使得原有的正负样本各从 1k5 张数据扩充为 1w 张，重新进行了模型训练。

本周也对停车的障碍进行了代码测试与检验。这一任务包括三个基本任务，即检测到标志、判断左拐右拐、开环转弯停车这三步。第一步可采用与原有的左右标志检测类似的方法，暂时的测试结果表示，跑满停车标志检测的代码对原有控制周期有一定的影响，但还在可接受范围内。我们也尝试使用简单的方法进行优化来减少运行次数与触发条件，但发现停车这一标志可使用快速的传统方法特征识别很少，我们尝试进行窗口阈值、断路阈值等多种方式进行特征优化识别，但发现很容易因反光、褶皱等造成阈值不合适，导致漏识别。而漏识别的代价是不可接受的，因而暂时仍采用跑满的模式。

左拐右拐的判断我们对标志物的特征进行识别，从左右两边环境的不同进行判断，判断出有遮挡的一边。然后控制小车朝着无遮挡的一边转向、再前行一段距离即可。

同时，本周对代码框架进行完全地推倒重写，对诸多代码进行封装与参数规范化，使得代码框架更加合理易懂。也完成了 h5 模型与 pb 模型的转化，代码中也不用再调用初始化加载速度较慢的 tf 框架，使用 dnn 即可完成调用，识别准确率与之前无异。封装好的代码可以在调试时快速闭合需要测试的部分，更加方便了我们的调试。我们也在代码中加入了录制视频的功能，便于离开场地后的调试。

目前仍然存在的问题：

1 褶皱与反光有时会对巡线与标志识别造成影响，这一问题主要体现在放置在走廊尽头的场地，收到光线影响过大。

2.将停车标志检测部分跑满毕竟代价过大，间隔跑又担心在高速下反应不及时撞到检测牌，还是需要寻找停车的特征进行优化与改进。

3.识别检测的位置难以确定，存在或多或少的偏差，对于转弯标志可能出现拐弯过晚撞动检测牌的情形，对于停车标志则可能在高速时停车不及时撞上检测牌才转向。

4.pid 等具体参数仍然是随机给的，未调整，因而巡线中存在一些超调震荡，计划在完成所有代码框架后再进行修正。

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 4 月 14 日　　第八周

本周主要完成了，对多进程代码框架的重构、小车算法的优化与加速以及实地进行了策略的调整与加速，基本完成了标志识别的任务，并融合进巡线程序当中。

在代码架构上，我们基本修复了上一次遗留的多进程的框架问题，解决了多个进程之间通讯的 Bug。目前使用的多进程框架，在融合标志检测与基础巡线后的运行帧数基本稳定，不会出现某一帧的突然抖动对巡线造成的干扰与影响，。

本周，我们也尝试着对于摄像头的曝光时间进行简单的测试与调节。我们发现在小车高速运行时画面模糊严重，可能会错过标志位，于是我们尝试着能否减小高速运行下的晃动与模糊镜头问题。目前的摄像头采用的自动曝光方式，我们采用手动曝光的方式，通过手动调节增加曝光时间的方式来优化画面，但发现效果改进并不明显。

对于小车算法的优化方面，为了追求巡线的速度，我们对于代码进行了大量的简化与优化加速工作，采用了如单通道加速、判断条件加速等方法，并将标志检测的运行速度优化至 20ms-30ms 左右。同时，我们也对巡线代码进行优化加速，使得融合标志检测后的完整算法基本可以跑满摄像头的极限帧数，大致在 30FPS 左右，实际的运行速度也基本回复到早期测试巡线的速度，在未细节调参的情况下，基本可以实现在精准检测的情况下 50s 左右跑完两圈（双环有标志）。

由于电池问题，本周并未对小车进行细节上的调参与加速，下一周我们将对一些参数进行调节加快小车速度。同时，我们在下一周也计划使用图像增强的方式，对分类器的训练进行优化，使得分类器可以更好的适应不同环境的影响，更具有普适性。

批注 [HJ1]: 进度还是挺快的，完成基础任务后，可以考虑一些额外的创新实验。

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 4 月 9 日　　第七周

本周清明假期，暂停一周。

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 4 月 2 日　　第六周

首先，由于之前修车返回的小车原本固定好的摄像头固定角度出现问题，我们先在实验场地参考原本的方案重新进行了摄像头的固定，并测试了原本的巡线代码。

同时，本周对上一次新采集的数据的训练模型进行了测试，在实验场地测试时基本可以可靠识别“左”“右”两个标识，但由于霍夫圆检测的缘故，识别距离受到限制，只有在 25cm 左右方可检测到圆形标志，不过对于低速模式已经足够作出判断与完成转弯操作。

我们本计划以消减运行速度为目的，将 Tag 检测的代码设置为每 0.2s 左右执行一次，但发现效果反而形成了卡顿现象，或者说 Tag 检测的代码使采样周期变长造成系统稳定性变差，因而我们认为想要达到更好的效果代码的串行执行架构是不可取的。

所以，我们尝试着采用并行的结构，即多进程（multiprocessing）的方式对小车进行控制，使用 Pipe 完成常规巡线与标签检测的通讯，目前简要完成了基本架构，但尚且有一些小 Bug，计划下周前去场地对多进程的架构进行实地的测试。

批注 [HJ2]: 多线程的解决方案不错，多个线程分工协作。继续加油

同时，我们也在考虑能否用 C++完成代码的构架，以加速运行速度，且保持实验质量。目前代码迁移到 C 的主要问题在于标志检测部分原本训练的分类器基于 Tensorflow 与 Keras 进行，我们考虑将其使用 OpenCV 的 DNN 模块进行调用，即将 h5 文件转化为 Opencv 可调用的 pb 与 pbtxt 文件，但最终转化的效果并不理想。因而，我们考正在考虑是否重新利用 Pytorch 对分类器模型进行训练，以方便在 Opencv 的调用。

此外，针对霍夫圆检测有时的不稳定，我们考虑采用角点与轮廓检测的方法对正方形进行检测，直接对 Canny 图或二值化图处理的效果极差，又考虑使用颜色阈值确定可能位置与大致范围，再进行检测，但实验结果发现很容易由于各类问题使得正方形被切割，难以检测或者说可靠性极差，暂时没有想到其它的方法解决这一问题。

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 3 月 26 日　　第五周

本周由于小车故障，回实验室返修，无法进行相关测试，只能对上周拍摄的照片进行数据标注，重新训练了模型，并尝试简化模型提高效率，实际准确率有待测试。

这里总结一下仿真部分到现在为止的进展，并介绍已完成部分的实现原理。

我们完成仿真停车任务的基本思路是：

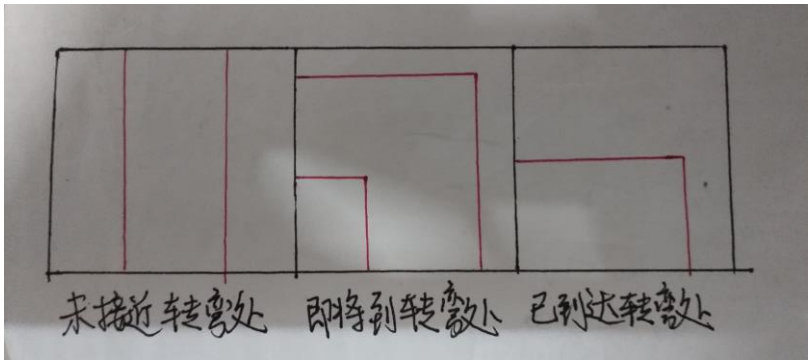
1. 根据俯视图，判定转弯时机
2. 通过调节参数，确定转弯的主要动作幅度
3. 根据俯视图，进行双线巡线
4. 根据后视图，判定根据标志牌倒车入库和侧方停车的时机
5. 根据右视图，判定带纹理的停车时机
6. 完成停车动作

批注 [HJ3]: 解决思路还是挺不错的，小车方面如果有问题可以随时联系助教帮助解决。仿真平台的相关实验要抓紧了。

1. 判定转弯时机

合适的转弯时机是到了转角的时候。因此，判定转弯时机即判定转角的出现。

在俯视图中，可以清楚地看到道路两边的黄色边线。边线的方向和数量可以作为车辆行驶状态的特征。



因此，判定转角的出现即检测横向和纵向边线的数量。

检测边线的步骤是：

（1） 提取场景中黄色的部分视为边线。

（2） 通过二值化和形态学腐蚀操作，把有一定宽度的边线变为细线。

（3） 用 Hough 变换提取直线，直线的表达式为 $\rho = x\cos\theta + y\sin\theta$ 。

（4） 根据直线的 θ，把所有直线分为横线和竖线两类。如果不能分为两类，说明只检测到了横线或者竖线。

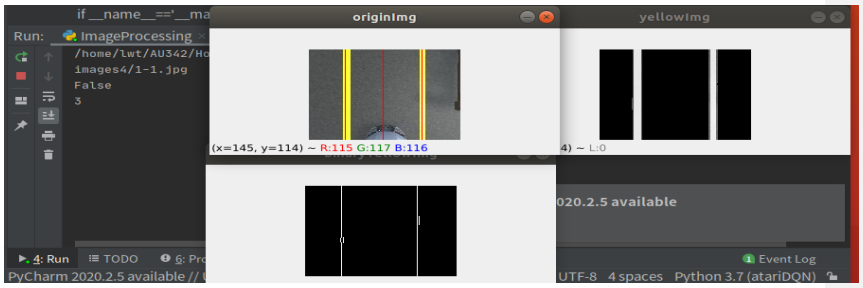
（5） 求两类直线的 ρ 的均值，把类内直线分为显著小于均值和显著大于均值的两类。如果不能分为两类，说明只检测到了一条线，否则说明检

测到了两条线。

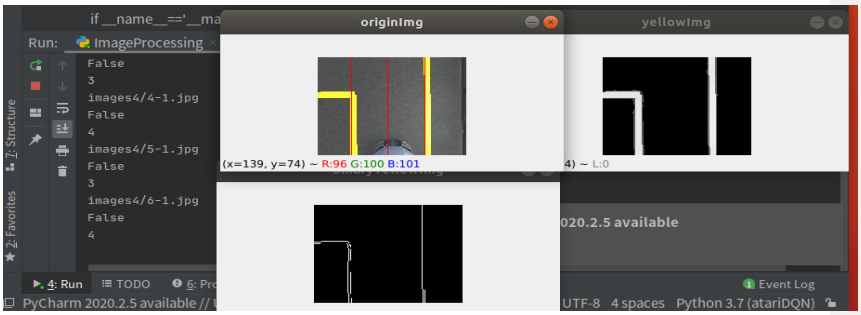
（6） 综合横线和竖线的有无和数量，给出检测结论。

横线数量 / 竖线数量	0	1	2
0	脱离车道	脱离车道	脱离车道
1	未接近转弯处	已经到转弯处	即将到转弯处
2	未接近转弯处	即将到转弯处	即将到转弯处

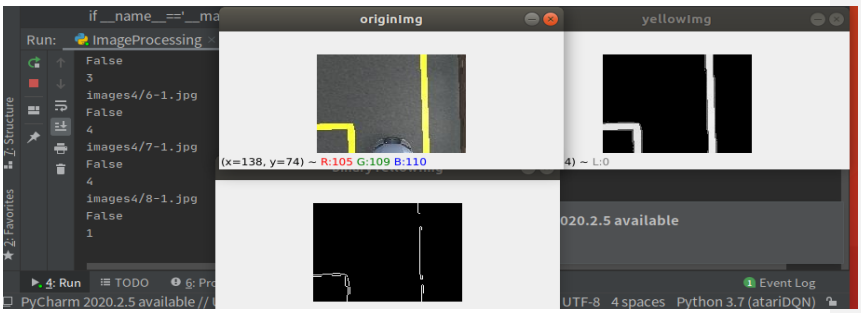
转弯时机判定效果如下图所示。



标志 3 代表未接近转弯处



标志 4 代表即将到转弯处



标志 1 代表已经到转弯处（为了避免误检，设定不检测过短的边线。考虑到从发出控制命令到车辆响应存在延迟，车辆的实际转弯时间不会过早。）

2. 确定转弯的主要动作幅度

通过调节参数，我们让车辆在一帧之内大致完成了转弯，以保证鲁棒性。这种做法的依据是，在现实中，在有标线的停车场中转弯的中途，我们往往也是不依赖眼前景象，到了转角就根据经验大致转弯，接近完成转弯时才看车道回正。然后，我们通过巡线算法让车辆回正。

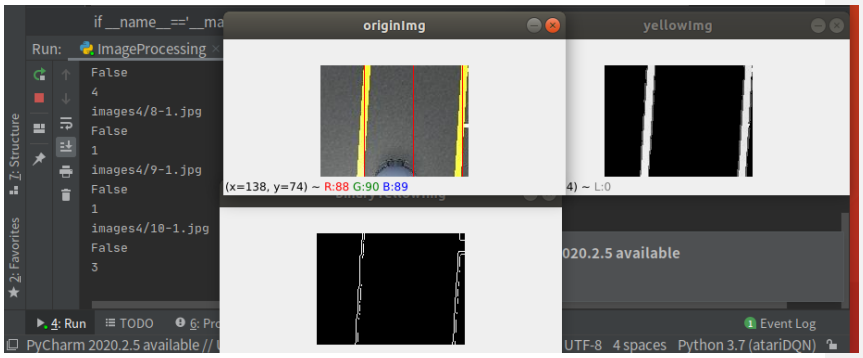
3．双线巡线

考虑到应用场景，我们目前只需要在检测到两条竖线边线的时候进行双线巡线，不需要应对其他的巡线场景。

巡线的算法包括定位理想路线和实施巡线控制。

我们通过计算两条边线的中线来求出理想路线。前面已经把检测到的竖线 $\rho=x\cos\theta+y\sin\theta$ 分为位置不同的两类。分别计算两类竖线 ρ 和 θ 的均值，确定两条边线的表达式。再计算这两条边线 ρ 和 θ 的均值，就得到了作为理想路线的中线。斜率绝对值过大的中线会被修正为垂线。

定位理想路线效果如下图。



我们采用预瞄法实现巡线控制。把与车头有一定距离的理想路线上的一个点设定为预瞄点，把车辆和预瞄点的连线与理想路线的夹角为 0 作为控制目标。由于任务目标是停车而不是巡线，巡线不需要太精确，我们仅采用了 P 校正就达到了响应较快，稳态误差较小的控制效果。

4．判定根据标志牌倒车入库和侧方停车的时机

我们根据后视图，调用示例代码中的目标检测库函数进行判定。为了避免误判带来的影响，在连续多次检测到标志牌之后，我们才开始停车。

5．判定带纹理的停车时机

我们通过识别纹理的特殊颜色判定带纹理的停车时机。由于右视图中车位距离镜头相当近，纹理的色彩和面积也非常明显，一旦发现纹理就要开始停车。

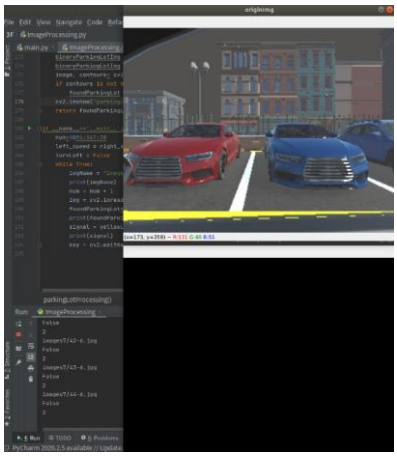
检测纹理的步骤是：

（1） 提取场景中青绿色的部分视为纹理。

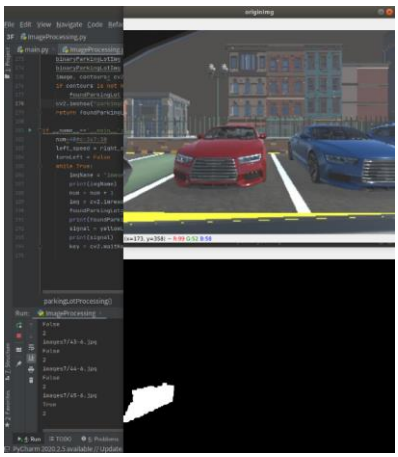
（2） 通过二值化和形态学腐蚀和膨胀操作，滤除噪点，保留大块的纹理。

（3） 圈出纹理的轮廓，如果找到了轮廓则检测到了纹理。

纹理检测效果如下图。



无纹理时



有纹理时

6. 完成停车动作

这部分尚未完成。

综上所述，我们已经完成了绝大部分的仿真工作，这部分工作具有良好的实现效果。遗憾的是，受仿真平台的性能和稳定性所限，我们无法测试一些更科学的控制方案，不得不有意规避平台的 bug，优先保证程序能够在平台上正常运行。

小组成员：柴子豪 唐欣阳 陈登舰 赖文涛 许言轩

更新日期：2021 年 3 月 18 日　　第二周

本周我们小组将任务重心放置在标志位的识别与判断上，我们使用上周一离开前时拍摄的照片（150 张左右）做为数据进行算法的编写与模型的训练，并在本周三下午前往实验场地进行模型的初步测试与检验，并额外拍摄了 2000 张照片作为数据集的补充。

关于数据集的识别我们起初主要考虑了以下的方法：

1. 基于 opencv 的模版匹配、颜色判断等等
2. 滑动窗口与分类器结合
3. 基于深度学习的目标检测

批注 [HJ4]: 这一周基本上就完成了标志牌识别相关的任务了，节奏很好。在设计相应算法的时候，可以调研一下树莓派小车的性能，以免和预期效果有较大差异

1. 模版匹配

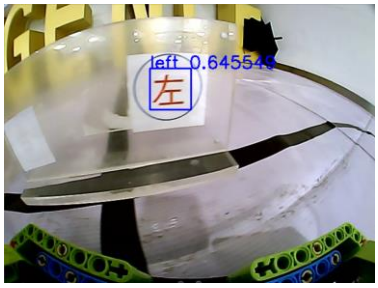
模板匹配是一种在一幅图像中寻找另一幅模板图像最匹配部分的技术。通常来说，模板为已知的小图像，使用模板在一副大图像中搜寻目标。模板匹配的原理如下：在待检测图像 I 上，按照从左到右、从上向下的顺序滑动模板，计算模板图像与重叠字图像的匹配度。匹配度越大，两者相同的可能性越大。

我们在实验中采用 opencv 的模板匹配函数，即使用模板图像，在输入的大图像上滑动以寻找相似的最佳位置。函数返回的结果是一个灰度图像，每一个像素值表示此区域与模板的匹配程度。

Opencv 提供了六种计算匹配程度的方法：方差匹配法、归一化方差匹配法、相关性匹配法、归一化的相关性匹配法、相关系数匹配法、归一化的相关系数匹配法。我们采用归一化相关系数匹配法（TM_CCOEFF_NORMED）进行实验。

模版制作。


匹配效果图：



实验过程中，对于模版匹配的基本认识如下：

1. 优点在于代码简单，难度不大，运行速度勉强可用
2. 缺点，过度依赖模版，即模版需与被识别标志十分相似才能识别到，识别效果一般，精度不高，鲁棒性极差
3. 不优化的条件下，完整图像的运行速度为 1.8s（2*0.9s），考虑缩小 ROI 后运行时间应该可以进一步调到 0.5s 以内，且有进一步的优化空间。但考虑其鲁棒性与实际运行效果，我们选择放弃此方法。

2. 基于颜色判断的优化

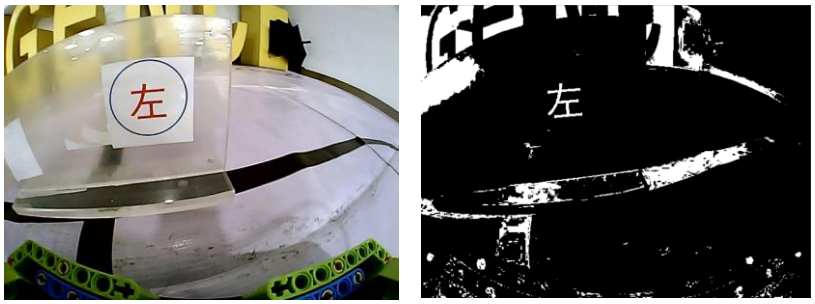
考虑到左右字体的颜色与背景颜色差异明显，我们也尝试了基于颜色判断优化或判断，本意是想借用此方法缩小 ROI 锁定区间减小计算量，但实验证明，此方法可靠性较差，很难使用。

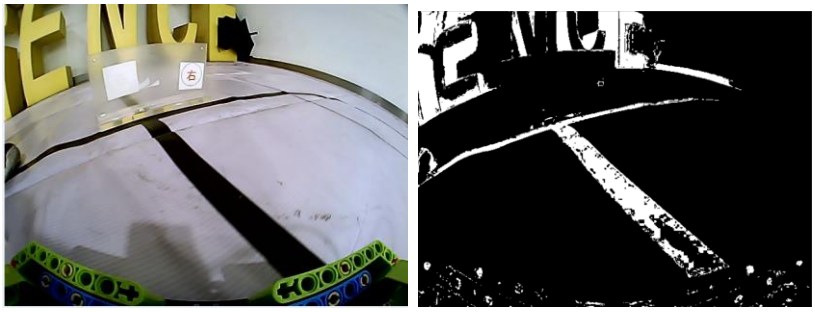
我们首先尝试在 RGB 空间内直接对颜色进行限制与筛选，颜色区间极大，得到效果较差，且背景与前景难以有明显区分。

之后我们将颜色空间转移到 HSV 空间，颜色区间减小，效果如下图所示。


但在全局照片进行测试后，我们发现效果并不像想象中那样好，可以看到前

两幅图片效果尚可接受，但 34 图已经难以使用，受光线、视角影响过大，且距离较远时字体不清，识别效果极差，因此舍弃。





3. 滑动窗口与 keras 模型分类器结合

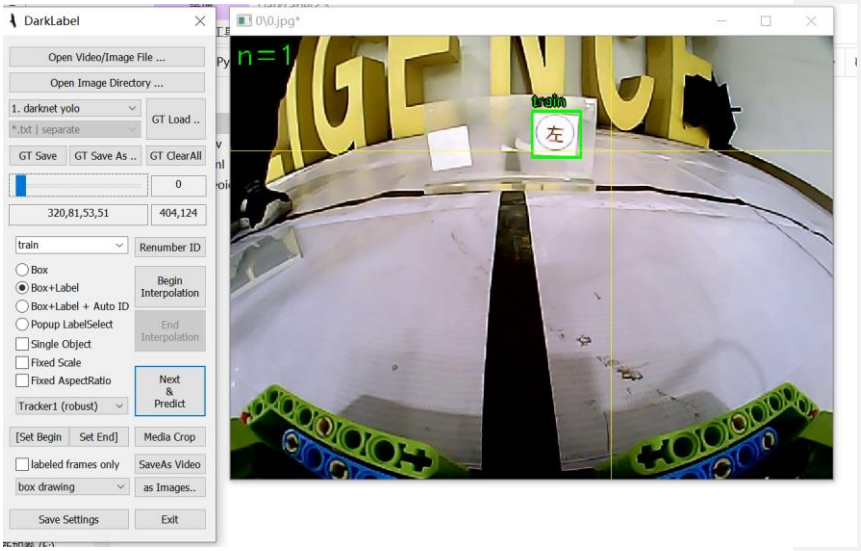
考虑到实际的应用效果，我们尝试设计一款分类器，然后使用滑动窗口进行滑窗的方法。

在结合滑动窗口的条件下设计分类器，我们可以假定待检测图片内部只包含其中一种物体，也就是说不会涉及到多物体检测，不必采用 OD 模型，而且待检测图片背景不复杂，可以大大降低背景噪声的引入对分类模型的影响。

因此，我们基于 Tensorflow 与 Keras 进行分类器的设计。

（1）数据标定

我们使用了 DarkLabel 对图像类别（左右）进行标定，并以 yolo 格式（主要是为了方便深度模型的训练）存储到 txt 之中，标定过程如下所示。目前完成了初次采集的 150 帧左右的图像的标定。



（2）训练

模型的训练我们基于 Keras（2.2.4）与 Tensorflow（1.14.0）完成，

Keras 是一个用 python 编写的深度学习 API，在机器学习平台 Tensorflow 上运行，它以 Tensorflow 为后端，作为神经网络的推理引擎，使用简单，功能灵活强大。

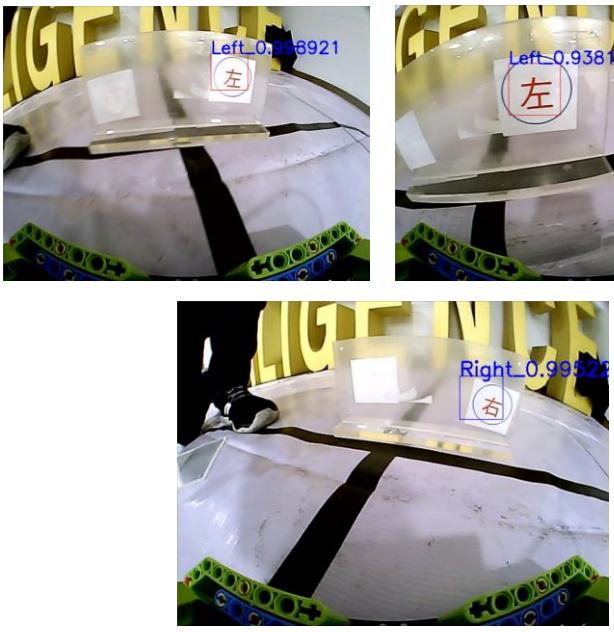
我们经过训练集测试集二八分、归一化等处理后生成了训练数据，并通过 Sequential 搭建简单的串行网络，网络包含基本的卷积、池化、激活等等，最后通过全连接与 softmax 完成分类输出。

在最早，我们只将左、右两个类别作为输入进行训练，后来发现实际效果较差，误识别现象极度严重，于是，我们采用随机采样的方法，随机选取一些非标志图片作为负样本，训练三分类模型，效果得到了极大的改善。

(3)应用

在完成训练后，我们将输出的模型转移到树莓派上，并采用滑动窗口算法在指定 ROI 内以指定的重叠度进行滑窗。

实际应用效果如下所示：



这一方案的基本状况如下：

1. 优点：训练简单速度快（1min 以内即可），模型较轻（1M 以内）
2. 缺点：模型依赖数据，目前数据集较少且重复度较高，在新的数据集上表现不佳
3. 运行速度取决于滑动的窗口个数，每一窗口判断用时 0.02s，整体运行时间为窗口数*0.02s 左右，高重叠度完整滑 64*64 的窗口大致需要 4s 左右。
4. 模型有进一步简化的空间，我们也尝试使用将 h5 模型简化为 tflite 模型，模型大小可从 1M 缩小至 440KB，运行速度小幅提高。也可以采用量化的精简方法，将 float32 存储为 uint，目前量化简化后模型大小为 107KB，但部署有一些 Bug，尚未解决。

4. 霍夫 Hough 圆检测与 keras 模型分类器结合

这一方法其实是对上一方法在时间上的优化，考虑到滑动窗口需滑动的窗口较多，极大浪费到运行时间，我们便考虑能否利用其它的特征进行简化模型，使检测窗口数目减小，于是便有了这一方法。

霍夫变换的思想是将图像从原图像空间变换到参数空间。

霍夫变换的对应关系为 1 直线映射为点；2 点映射为正弦曲线；3 一条直线上的多个共线点映射为参数空间相交于一点的多条正弦曲线。

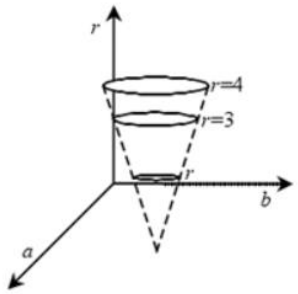
二维图像空间中一个圆转换为该圆半径、圆心横纵坐标所确定的三维参数空间中一个点的过程。因此，圆周上任意三点所确定的圆，经 Hough 变换后在三维参数空间应对应一点。该过程类似于选举投票过程，圆周上任意三个点为一选举人，而这三个点所确定的圆则为一候选圆。遍历圆周上所有点，任意三个点所确定的候选圆进行投票。遍历结束后，得票数最高点所确定的圆为该圆周上绝大多数点所确定的圆。

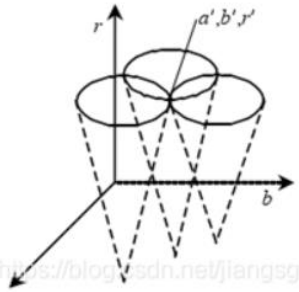
霍夫变换的具体步骤：适当的量化参数空间；将参数空间的每个单元看作一个累加器；初始化累加器为 0；对图像空间的每一个点，在其所满足参数方程对应的累加器上加 1；累加器储存的最大值即为对应的图形参数。

检测圆的原理：对一个半径为 r，圆心为(a, b)的圆，表示为：

$$(x-a)^2+(y-b)^2=r^2$$

坐标向量为 x=[x, y]，参数向量 a=[a, b, r]。则图像空间的一个点(x, y)对应参数空间的一个圆锥。而图像空间的一个圆就对应着这一簇圆锥相较的一个点。这个特定点在参数空间的三维坐标就代表着图像空间那个圆的参数坐标。

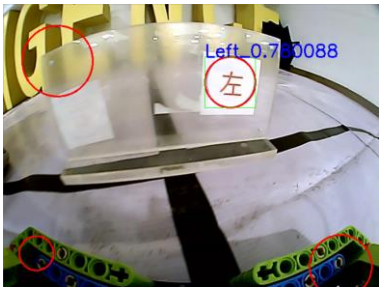


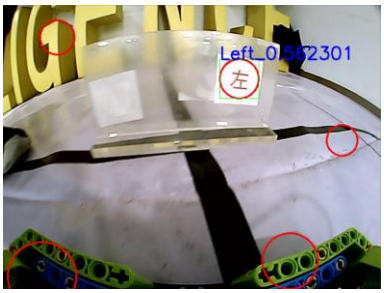


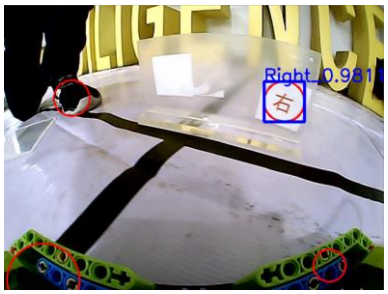
将图像上的所有点映射到三维参数空间上，，寻找参数空间的累加器最大的值，作为图形参数。

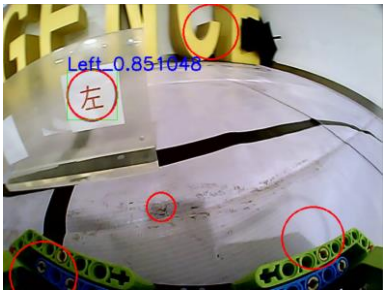
首先对全图进行霍夫变化与圆检测，存储圆心位置与圆半径，并以此作正方形作为窗口进行分类器检测。

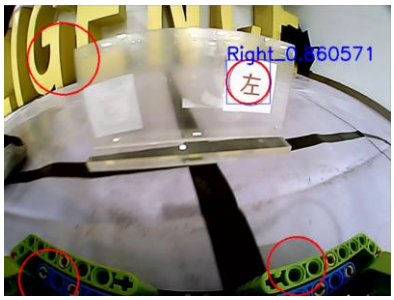
检测效果如下所示、

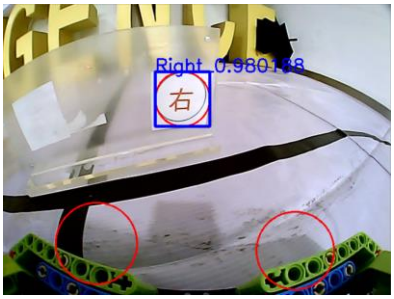












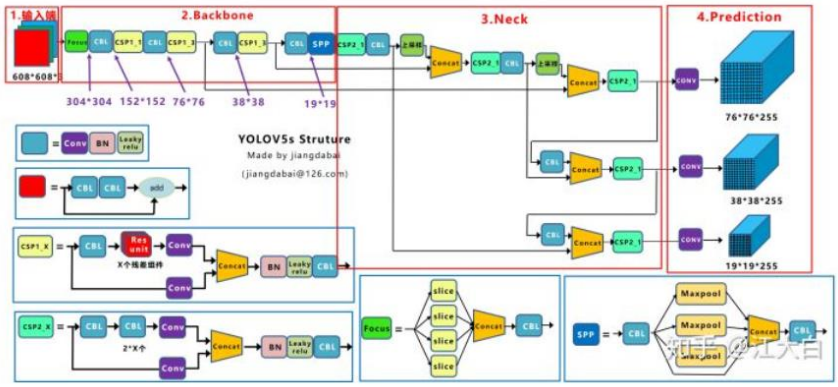
这一方案后的特点：

1. 优点：速度快，Hough 变换 0.1s（未优化），检测一次窗口大致 0.02s，因此算法可以控制 0.18s 左右，与其它方法相比已有较大的改善。降低 ROI 与优化我们认为可以控制在 0.08s 左右。
2. 缺点：分类器模型与上一方法一样，都有着依赖数据的问题。且距离较远

时 Hough 检测不到圆，自然也不会进行检测，因而识别距离不会很远，但应该足以达到巡线要求。

5. 基于深度学习的 Yolo 目标检测

YOLOv5 是在 PyTorch 中实现的，它受益于成熟的 PyTorch 生态系统，支持更简单，部署更容易，相对于 YOLOv4，YOLOv5 具有以下优点：速度快、精度高、体积小等优点。



YOLO 是一个轻量级单阶段目标检测网络，从 YOLOv1 到 YOLOv5 也经过了多次改良和升级。其主打的效果就是快速，同时检测精度也并不低。同时结构简单，没有特殊网络结构，使得它容易部署，从出现开始，便受到了工业界的广泛使用。因此网络上和 YOLO 网络相关的博客文章也十分丰富，极大的方便了人们的使用。因此我们选择了 YOLO 系列中的最新版本 YOLOv5 作为我们和目标检测算法。

YOLOv5 的特点主要包括：

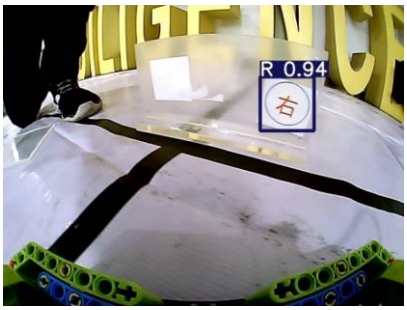
1. 特征金字塔的检测方式，覆盖各种尺度的目标。

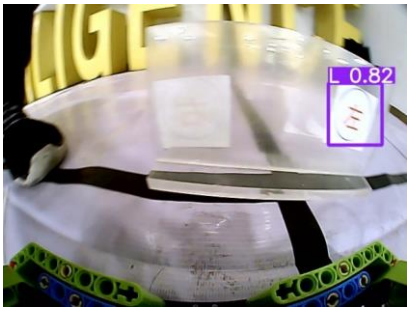
2. 全卷积网络，可以处理不同分辨率的输入图像。

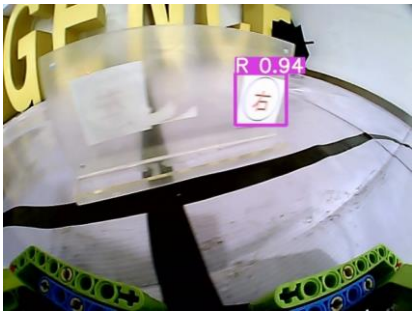
3. mosaic 数据增强，拼接图像，减少训练时的样本不平衡现象，提高小对象的检测精度与召回率。

4. Mish 激活函数、Dropblock、CSP 结构、Focus 结构。（融合一些较为有用的 tricks）

我们尝试使用 YOLOv5 进行标志物的识别，效果如下所示。







方案评价：

1. 优点：精度高，识别距离远，可以在较远的距离、较小的标志
2. 缺点：全扫描用时 4.4s，未优化情况下不可接受
3. 与分类器模型一样，依赖数据，在未标注的数据上表现不佳，检测与分类都有误判，150 张图片训练远远不够

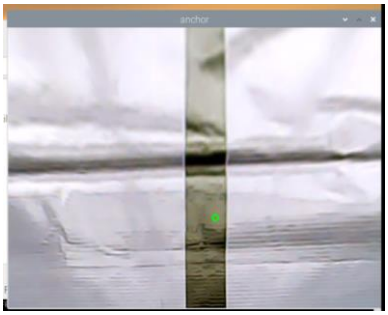
6. 总结

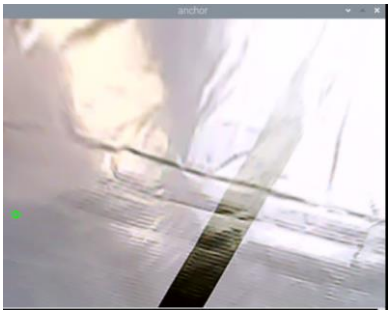
本周主要对标志物的识别进行了探究与实践，尝试（踩雷）了多种方法，其中简化的分类器模型用时最短，Yolo 方法精度最高，最后得出此树莓派的性能在不超频的情况下想要既保证运行周期又保证高精度较为困难，因而需要一定的取舍。

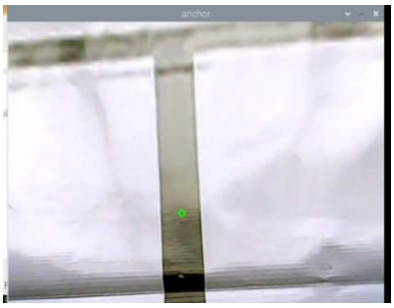
另外，本周小车电池由于不知名原因，无法充电，因此本周所有的实验都是在 USB 供电情况下完成，总提示 Low Voltage，不确定是否会对树莓派性能产生影响，也无法进行巡线测试。

同时，在实践过程中，我们发现 SD 卡空间不足，编译困难，因而使用自己的 64GSD 卡重刷了系统，并进行了相关的配置。

最后，本周在前去测试的时间为下午，因而发现了场地的光线可能会对小车的巡线产生极大的影响，实摄光线的影响效果如下所示。由于电池问题，暂时无法进行实际测试，但预计会产生极大干扰，可能需要对算法进行大面积修正。


anchor


anchor


anchor


anchor

更新日期：2021 年 3 月 11 日　　第一周

本周是第一周，我们小组尝试完成智能车控制的基本任务有：相机标定，图像处理算法测试，控制算法研究。

批注 [HJ5]: 这一周完成的任务还蛮多的，有想法就可以尽快去场地调试。

1. 相机标定

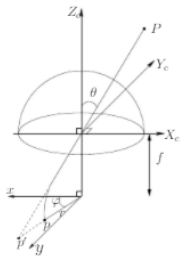
首先，要完成获得智能车周围场景特征，我们需要进行相机标定，重构场景。标定相机内参及畸变系数（左图：标定现场，右图：前置摄像头实摄的 20 张棋盘图）。

观察到该摄像头畸变特性及其明显，如果选择普通相机的标定方法进行标定，不能保证去畸变后的效果是最好的。查阅文献资料后，我们最终选用了鱼眼相机的标定方法。阅读文献《A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses》后，我们总结了鱼眼相机 fisheye 的标定原理如下：

1. 相机标定的过程通常分为两个部分。第一步是从世界坐标系转为相机坐标系，从三维真实坐标转换为三维相机坐标；第二步是从相机坐标系转换为成像平面坐标系，从三维相机坐标转换为二维平面坐标。完成相机标定。

2. 鱼眼镜头可覆盖相机前方的整个半球形视场，视角约 180°，然而不可能通过透视投影在三维图像平面上投射半球形视场，因此鱼眼镜头不适用透视投影模型。由于存在固有畸变，鱼眼镜头的图像服从某种投影模型。考虑更灵活变化的径向对称投影模型。

3. 右图是鱼眼镜头模型。鱼眼相机的投影模型的一般形式如下：$r(\theta) = k_1\theta + k_2\theta^3 + k_3\theta^5 + k_4\theta^7 + k_5\theta^9 + \ldots$ ，。我们需要估计上式中的各项参数，即可获得鱼眼镜头模型。



(b)

4. 鱼眼镜头标定步骤：

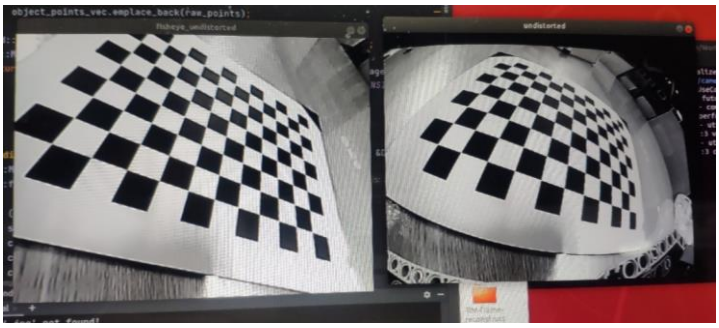
（1）采集标定图像。我们使用智能车上的前置摄像头进行现场拍摄制作好的棋盘图，获得 20 张标定图片。

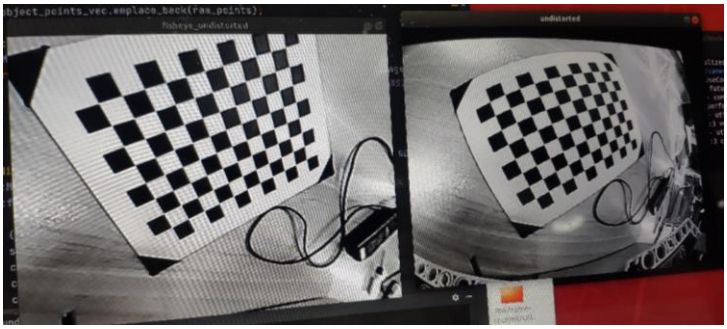
（2）进行图像的形态学处理。我们对采集到的标定图像进行去噪滤波操作，去除噪声干扰（主要是受场地影响产生的光线的干

扰）。该部分算法在 Part2 中说明。

（3）使用 OpenCV，检测每一张图片中的格子交点坐标，排序后调用函数进行计算，得到摄像头响应的内参与畸变系数（本周报中不展示具体数据）。

（4）计算误差，检查畸变矫正的效果。对其中两幅图片的标定的结果如下所示，可以看到，畸变矫正效果很好，虽然一定程度上减少了摄像头的可观察区间，但仍在可接受范围内。

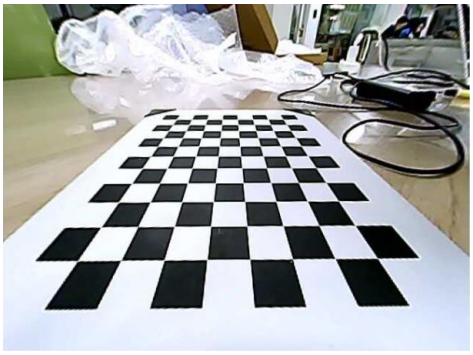




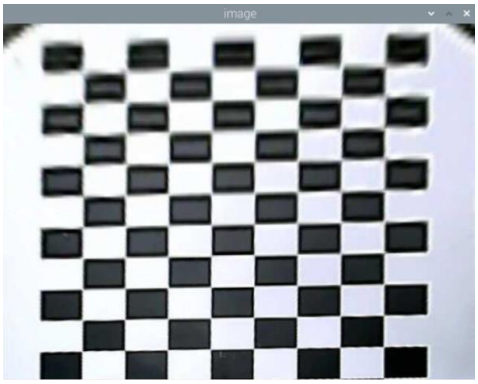
（5）透视变换。对畸变矫正后的图像进行透视变换，获得目标图像。以拍摄的一张棋盘照片为例，完整的变换过程如下所示（原图-畸变矫正后-透视变换后）：

原图

畸变矫正



透视变换



可见相机标定基本达到了我们的目标要求。我们在实际赛道中进行测试，结果如下：

①左弯道：

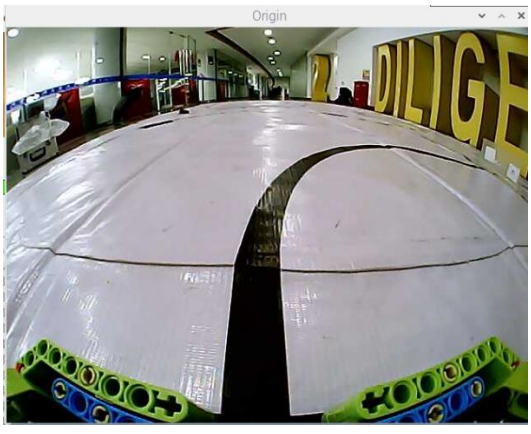
原图

畸变矫正

透视变换

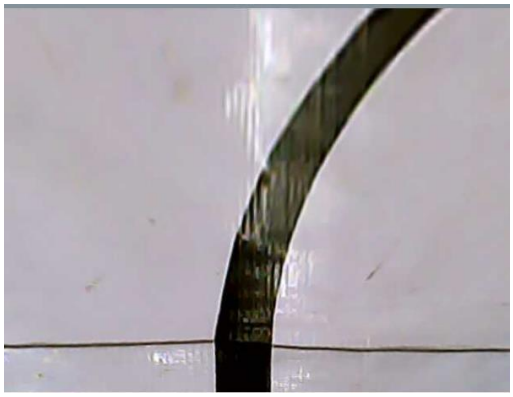
②右弯道

原图



畸变矫正

透视变换



可以看到，我们的透视变换足以满足小车运行的需求。

2. 图像处理算法

由于成像角度、光线干扰等因素影响，在处理视觉传感器获取的数据时，我们需要可靠而准确的算法来更清晰地检测出图像中的车道线特征。

在对赛道的处理上，我们主要使用了高斯滤波与形态学运算等滤波处理算法，并尝试使用了二值化、Canny 算子等算子进行处理如下：

1）高斯滤波

对整幅图像进行加权平均，每一个像素点的值都由其本身和邻域内的其他像素值经过加权平均后得到。具体操作是：设置一个模板扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值来替代模板中心像素点的值。

2）直方图均衡

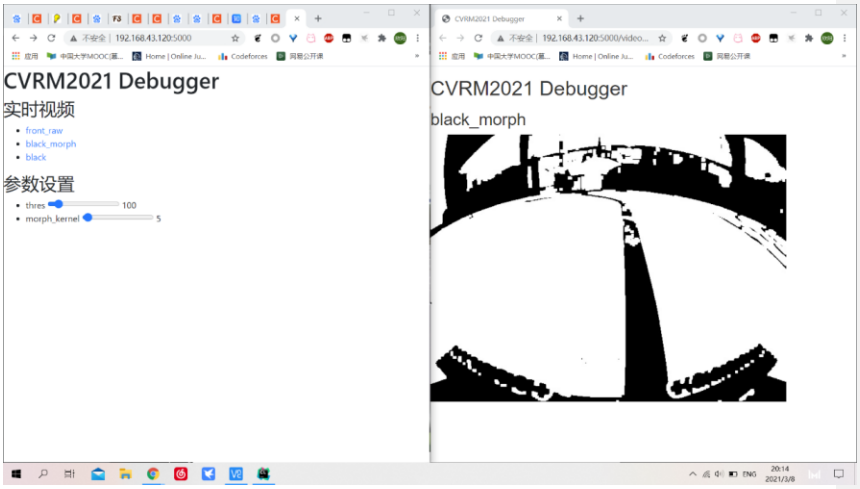
用以解决图像光照不均问题，提高对比度。

3）形态学变换

我们尝试了图像的开闭运算和膨胀腐蚀运算，但发现效果并不明显，于是最终未采用。

4）二值化

为了获得赛道的形状，我们对高斯滤波与直方图均衡后的图像采取二值化操作，将图像转化为黑与白两种颜色。同时，我们组内成员将参数的调节与整定过程可视化与可调化，调节过程如下图所示，可以在左侧滑动条方便的进行参数调节，并在右侧实时观察到调整的结果。



5）Canny 算子

我们使用 Canny 算子进行图像边缘检测，以获得赛道。Canny 边缘检测算子的原理如下：

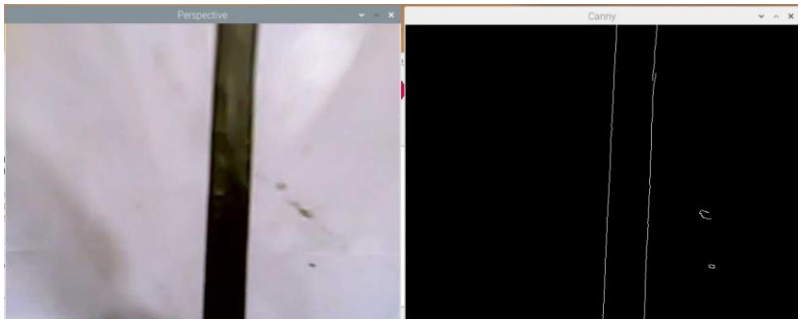
①高斯平滑：与高斯滤波类似，由于前步已处理，我们实际跳过此步。

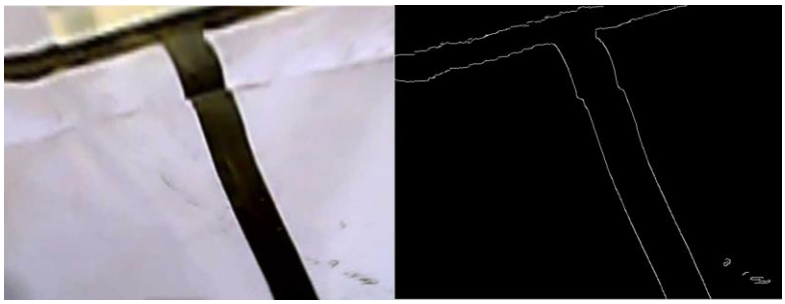
②计算梯度大小与方向：使用 prewitt 模板来计算图像梯度的大小与方向。

③根据角度对梯度的大小进行非极大抑制。

④使用双阈值法来检测边缘。

初步使用的 Canny 算子效果如下所示（透视变换-Canny 算子）。可见，Canny 算子的边缘检测效果优异。





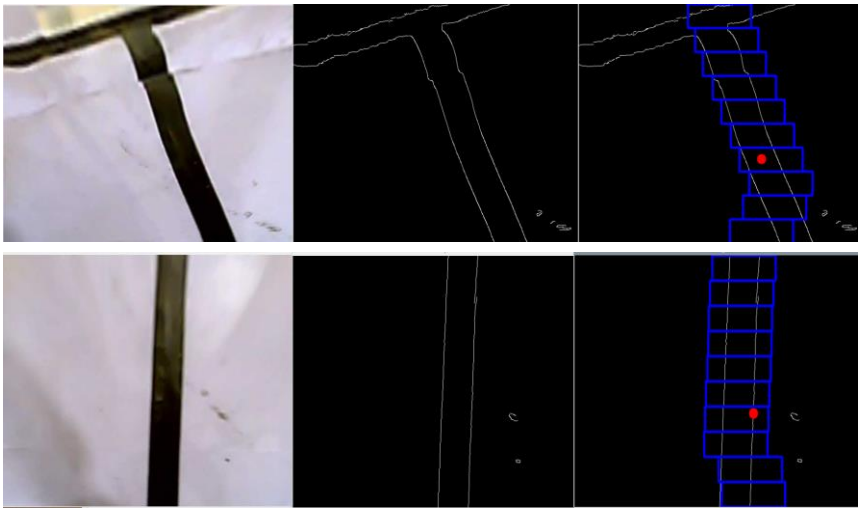
6）滑动窗口算法 Sliding Window Algorithm

我们已经通过 canny 算子获得赛道的边缘，接下来需要进行赛道寻线操作。我们采用的是滑动窗口算法，该算法通常应用于数组和字符串上，我们将其移植到我们的图像处理中，用于赛道寻线。步骤如下：

①下图中蓝色框为滑窗，在每一个小窗口里面检测线的分布并取均值。

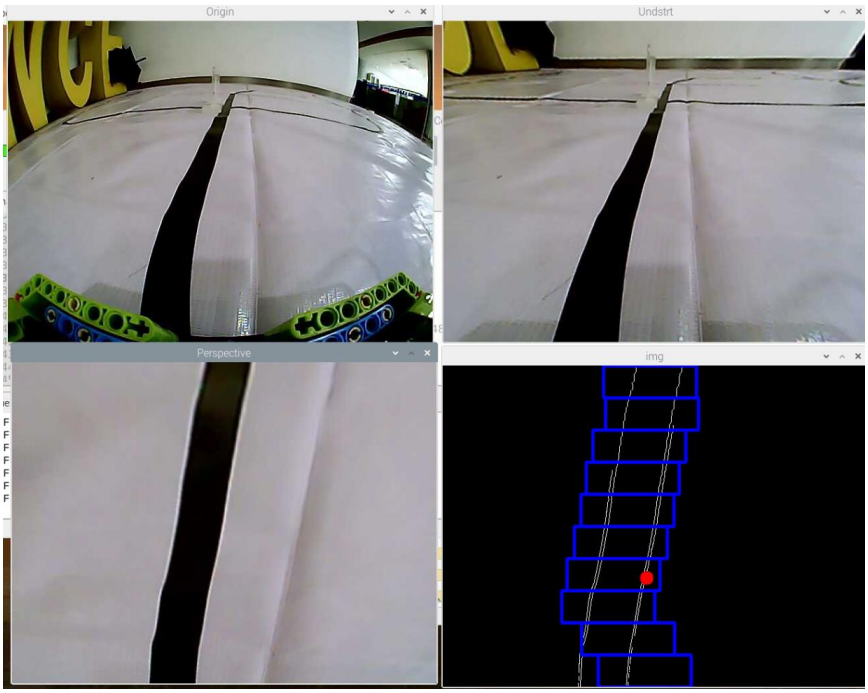
②将均值作为下一个窗口的起始滑窗位置，左右搜索，依次滑窗。

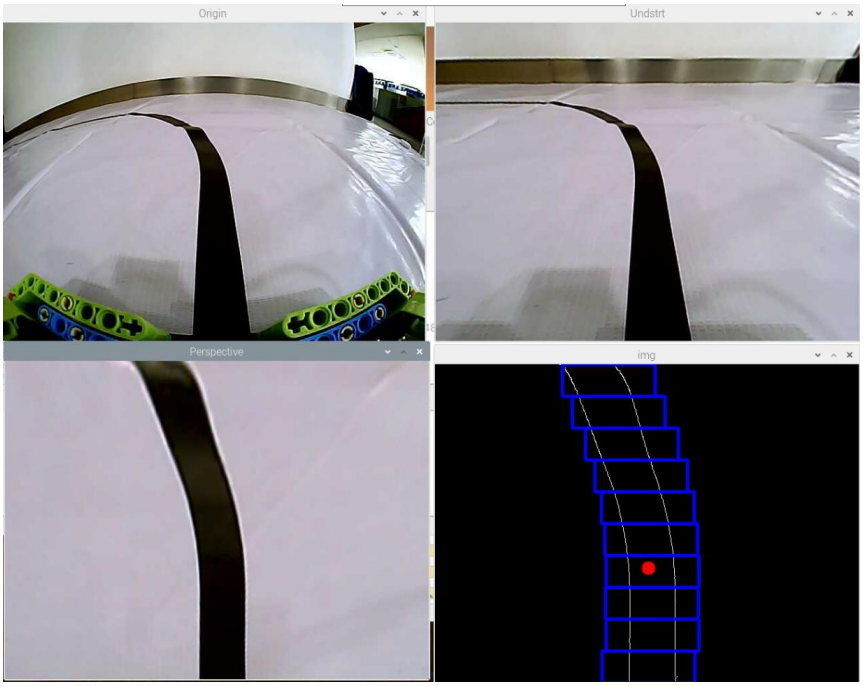
效果如下图所示



3. 完整算法效果

我们将以上算法结合，得到完整算法效果如下。

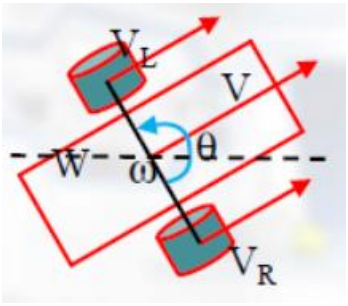

Origin
Undstrt
Perspective
img


Origin
Undstrt
Perspective
img

4. 控制模型

1）小车模型

我们使用 Tank-like 小车模型进行运动学分析。小车简化后的模型如下：



图中 VL 和 VR 分别表示小车的左轮和右轮的前进速度，W 表示轮距， θ 表示方位角（与 x 轴正向的夹角）， ω 是绕 Z 轴的角速度。运动关系如下：

$$V = \frac{1}{2}(V_L + V_R)$$

$$\omega = (V_R - V_L) / W$$

$$\dot{x} = V\cos\theta$$

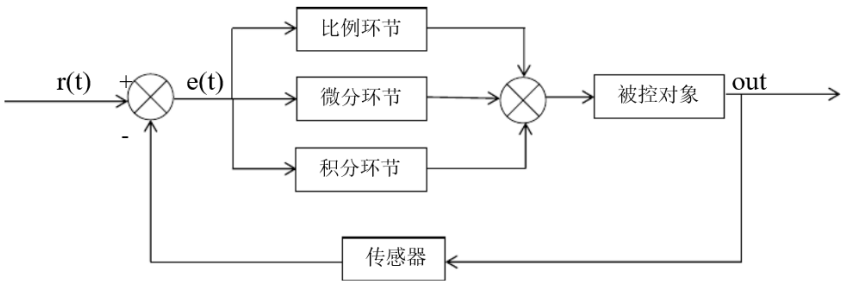
$$\dot{y} = V\sin\theta$$

$$\dot{\theta} = \omega$$

因此我们可以根据小车速度和角速度求出小车位置。

2）PID 算法

我们采用 PID 控制器来控制小车的运动。PID 控制器包括比例环节 P、微分环节 I、积分环节 D。PID 控制流程图如下：



PID 控制的基本方法是根据系统输入与预定输出的偏差的大小运用比例、积分、微分计算出一个控制量，将这个控制量输入系统，获得输出量，通过反馈回路再次检测该输出量的偏差，循环上述过程，以使输出达到预定值。

在 PID 算法中，比例环节 P 的作用是成比例地反映控制系统的偏差信号 e(t)，一旦产生偏差，比例控制环节立即产生控制作用以减小偏差。积分环节 I 的作用是消除静差，提高系统的无差度。微分环节 D 的作用是反映偏差信号的变化趋

势，能够在偏差信号变化之前先引入一个有效的早期修正信号来提前修正偏差，加快系统的动作速度，减少调节时间。

我们采用 PID 控制器，对小车进行速度、位置的控制，完成巡线任务。

4. 实验结果

在一个小赛道上进行测试，完整运行两圈需要 28s 左右，稳定速度可以到达 30cm/s 左右。

目前也主要存在以下问题：

1. 图像处理的部分参数仅进行初步估算，并未能进行精细调节，尚有较大的改进空间
2. 在未执行算法时，摄像头的运行帧率为 35FPS 左右，每帧畸变矫正与透视变换用时 0.06s 左右，Canny 算子、Gauss 滤波等操作用时 0.03s，完整滑动窗口用时 0.05s，因而，完整的算法执行下来只有 8FPS 左右，远远未到达摄像头的上限。我们对算法进行了简要的简化后，可以达到 18FPS 左右，还有进一步的优化空间，后续可能会考虑利用 C 语言对算法进行优化加速，或删去一些耗时较长且可有可无的算法。
3. 差速车的控制模型使用的简答的预瞄点控制算法与 Pid 算法，具体参数有待调节