



学科实践（一）：智能机器人理论与实践

智能小车实验报告

姓 名	刘本宸 (第一组)
学 号	22920202200764
日 期	2023 年 02 月 27 日
学 院	厦门大学信息学院
班 级	人工智能二班
课 程	学科实践（一）：智能 机器人理论与实践

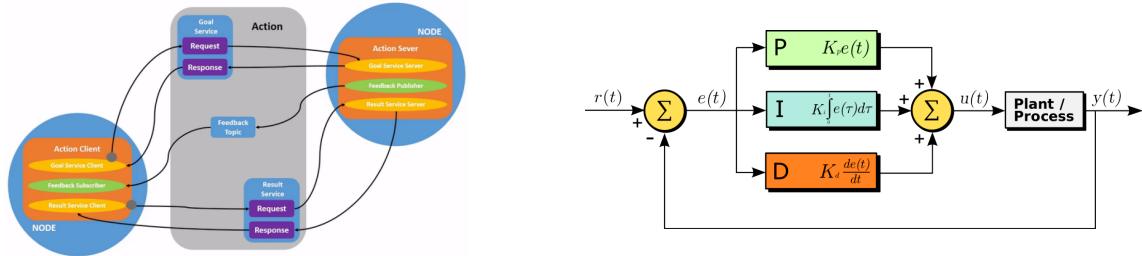
目录

1 选题以及选题依据	3
2 小组分工和工作计划表	4
2.1 小组分工	4
2.2 工作计划	4
3 实验过程	5
3.1 硬件模块	5
3.2 软件模块	7
3.3 设计技术参数	7
3.4 程序架构以及实现思路	8
4 实验结果检测	9
4.1 巡线模块	9
4.2 巡航模块	9
4.3 视觉模块	10
4.4 Yolov3-Tiny 神经网络训练	11
4.5 抓取模块	13
4.6 模块整合后实验结果	13
5 出现的问题以及解决方案	14
5.1 硬件问题	14
5.2 软件问题	14
5.3 网络问题	14
6 课程体会	15
A 附录	16
A.1 图表	16
A.2 Task1–巡线关键代码	18
A.3 Task2–寻物关键代码	19

1 选题以及选题依据

在课程的开始，我们先认识了 ROS2 系统和给定的小车的一些基本模块，后面我们开始循着老师给的选题进行实践。选题实现了两个项目。这两个项目的关系是递进的。第一个项目是基于小车的智能巡线，另外一个是基于 SLAM、视觉系统以及抓取的自动寻物小车。

ROS(Robot Operating System) 是我们本次机器人实践的软件平台。它提供一系列程序库和工具以帮助软件开发者创建机器人应用软件。它提供了硬件抽象、设备驱动、库函数、可视化、消息传递和软件包管理等诸多功能。ROS 遵守 BSD 开源许可协议，是开源的机器人控制框架。我们使用的 ROS2 是 ROS 的升级版本，ROS2 引入了数据分发服务通信协议，各个软硬件模块通过节点进行通信，而且减少了框架的内存使用。同时基于 ROS2 系统，我们还可以利用自带的 Rviz 等程序进行小车的状态可视化，这大大方便了我们的学习。并且我们使用的 ROS2 有对应的开源社区¹，在开源社区内已经有了一部分成熟，易用，鲁棒性强的“轮子”供我们学习和调用。同时我也通过 Github, Youtube 等多个网站上搜集了相关资料进行学习和吸收，这些学习资料也帮助了我快速完成后面的任务。



图片 1: ROS2 通信方式

图片 2: PID 控制算法

小车智能巡线主要是实现一个可以根据小车前部的视觉输入，自动研判当前位置，自动进行车轮速度调节和转向，并且判断是否达到了寻迹终点，到达终点时自动停止关闭相关服务，终止相关进程。在实践寻迹小车之前我们已经进行了小车基本模块的认识，而且还组内阅读，讨论了相关代码。在老师给定的代码框架之下，我们进行了部署，在小车上进行了测试。整体上是一次项目的练兵，从框架设计，到功能和逻辑的拆分，到代码实现，再到部署。整个流程我们小组完整的走了一遍。在这次学习和实现巡线小车的时候，我们还学习了 PID 算法。PID 算法是工业应用中最广泛算法之一，也是一种非常古老的算法（有一百多年的历史了），在对系统的控制中，可自动对控制系统进行准确且迅速的校正。在我们的小车实践里面我们使用了 PID 算法，巧妙的处理了给定地图中比较“大”的转弯，这样，在转弯时，我们的小车的前摄像头不会跟丢贴在地板上的“实线”，出现站在拐弯处原地不动的情况。

自动寻物小车是在自动巡线小车的升级版。在已有的控制算法基础上，我们在实现小车最终的功能时，尝试综合使用了小车上已有的多个模块，比如：激光雷达，摄像头模块，机械臂上的驱动模块以及机械臂上的视觉模块。而且在软件上面，我们基于已有的硬件模块进行了多个小车功能节点的编写，比如：位姿定位模块，物体识别模块，驱动模块，抓取模块，巡航模块等等。我们通过激光模块对整个地图进行建模，方便小车在地图中定位自己以及规划路径。并且我们尝试了训练一个 tiny-YOLOv3 的神经网络用于进行物体的识别。这个神经网络不仅在巡航时给出了物体相对于车的位置，为车靠近物体提供了方向，而且还给出了物体相对于机械臂的位置，这个对于后面机械臂抓取物体方面有很大的帮助。在完成抓取任务之后，小车会在我们的设定之下完成“回家操作”，这一步也是基于地图和导航的。自此，小车的各个部分相互关联工作，形成了我组的自动寻物小车。

¹<http://www.ros.org/> <http://wiki.ros.org/>

2 小组分工和工作计划表

2.1 小组分工

- 刘本宸 (组长) 负责整个组的统筹工作，代码框架设计、代码具体实现、部署和优化，保证可以完成课程选题以及整体论文撰写。
- 孟 媛 负责制作物体训练集，完成 PPT 报告的制作
- 姜小康 负责后勤工作，解决小车硬件故障，小车充电等
- 刘宇菲 训练 tiny-YOLOv3 神经网络，生成权重文件，部署神经网络
- 贾惠卿 负责管理和维护小车机械臂，保证机械臂基本功能正常

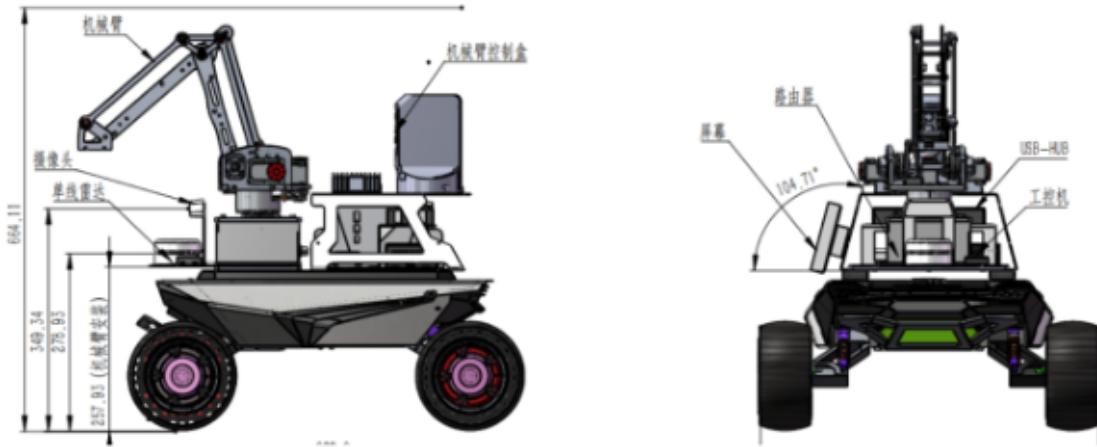
2.2 工作计划

- 1-4 周 熟悉 ROS2 机器人操作系统，熟练控制机器人，对各个小模块进行编程和功能测试
- 5-6 周 编程实现巡线功能，并且实现参数的调优，并且记录相关参数
- 7-8 周 编程实现寻找、抓取功能，优化程序运行流程

3 实验过程

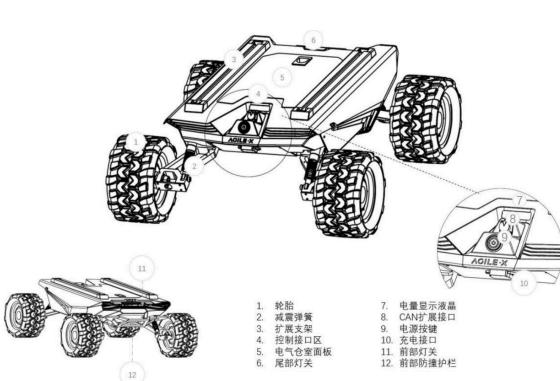
3.1 硬件模块

机器人整体外观如下图所示，框架采用钣金作为支撑件，在底盘上有四个电机在机器人的左边配备有肩屏，在正上方后部是机器人的中央处理器和机械臂控制模块，在正上方的前部是机械臂模块。最前面是 G4 激光雷达与奥比中光大白相机。还有一个遥控器但是并没有在图中给出。

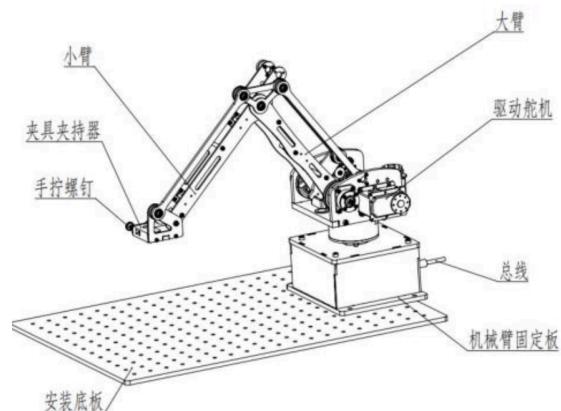


图片 3: 小车基本外观

机器人机械部分 SCOUT MINI 智能移动底盘是四轮驱动的，性能优越。该底盘在尾部增加了充电接口，而且在后部还有扩展接口，交互模块，供电模块等等。足以成为其他模块的底座。处理器采用的是英伟达的 **Nvidia Jetson Nano** 小型计算机，据英伟达的描述，该计算机专为支持入门级边缘 AI 应用程序和设备而设计。但是在实际应用中我发现这个小型计算机的性能并没有那么强劲，需要程序员亲自进行资源分配，而且要在程序设计时做好容灾处理。机械臂是配备的 **Alpha** 机械臂整体外形紧凑，安装便捷。PC 端提供了 BrobotStudio 集成软件，该软件融合了鼠标控制、图形化编程等多种功能，并提供了多种控制机械臂的运动形式，而且它支持多种编程语言，方便我们使用 Python3 和 ROS2 框架进行编程控制。

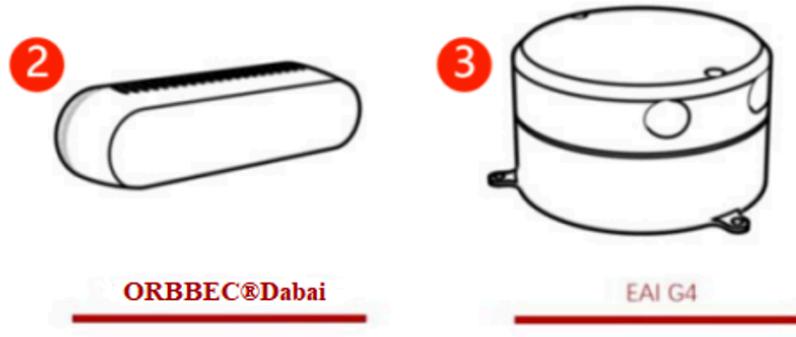


图片 4: Scout Mini 底盘



图片 5: Alpha 机械臂

机器人传感部分 ORBBEC®Dabai 相机是基于双目结构光 3D 成像技术的深度相机，有左右红外相机，可以计算物体对应深度，而且还有基本的图像接收功能。YDLIDAR G4 激光雷达是基于三角测距原理的一款激光雷达。不仅可以测距，并且在测距的同时，机械结构 360 度旋转，不断获取角度信息，从而实现了 360 度扫描测距，输出扫描环境的点云数据，这个点云数据可以帮助我们小车构建地图，获取小车位置。而且摄像头信息还可以通过 Rviz 进行可视化。摄像头（附于 Alpha 机械臂上）这个摄像头主要是在抓取的时候可以获取物体的和机械臂的相对位置，在抓取的时候可以准确定位，并且进行抓取，这个摄像头依附于机械臂夹具之上，故图片不再给出。



图片 6: 基本传感器

机器人遥控部分 遥控主要是使用遥控器，是随着机器人配备的。在对应的 SWB 模式置为 ON 就可以实现遥控器对于小车底盘控制，反之则由机器人上面的程序进行运动控制。小车的油门是在左手边，类似我们小时候玩的遥控汽车。



图片 7: 遥控器

(-5.72, -7.78, 0.0, -33.56), (-1.97, -2.31, 0.0, 53.65), (-5.03, -1.08, 0.0, 173.53),
(-2.76, -0.78, 0.0, 52.78), (-3.04, 2.49, 0.0, 96),]

四元组分别为 (**x, y, z, Orientation**)

- Yolov3-tiny 权重文件：同级列表中有权重文件，名为：yolov3-tiny-Vision-bear-2000.weights

3.4 程序架构以及实现思路

巡线实验—Task1 第一个巡线实验主要是需要协调好运动模块和视觉模块。首先视觉模块需要持续接受信号，并且将信号转化为对应的二值化图像，即把线从背景中分离开来，让最后的图像中只有线，之后通过计算对应的线的视觉中心，通过计算视觉中心的偏移和 PID 算法来把视觉信号转化为运动信号和转向信号。最后将运动信号输出到底盘上。当没有线的输入信号的时候，机器就判断已经走到终点，程序停止。程序框图在附录中（图片 20）。

寻物实验—Task2 第二个寻物实验会用到多种的传感器。首先要保证地图可以建成，地图的搭建使用的是激光雷达进行建图。建成的图片在文件附录中可以看到。并且小车需要通过对于点云的扫描来确定自己的位置和朝向。在寻找物体的过程中，小车需要开启导航模块，需要循着我们既定的路线进行寻址，以保证小车可以遍历地图上所有的点。加之，我们需要一直启动摄像机节点并且启动物体检测服务，如果在小车的巡航的时候发现了物体，就要进入寻找物体并且抓取的逻辑。而且在抓取逻辑结束之后需要回到起点，这也是需要导航服务工作的。程序框图在附录中（图片 21）。

4 实验结果检测

4.1 巡线模块

巡线模块主要是 OpenCV 的使用，使用这个视觉库对于输入画面先进行二值化再进行判断。如果路线转弯则向底盘发送转弯信息，否则就是直线行走如果摄像头丢失了图像的信息，说明已经走到了尽头，开始进入下一个任务（即 Task2）。

当然这个地方有一个比较 tricky 的地方就是，可以对输入画面先进行分割，比如在本次实验中，我们在判断巡线轨迹的时候只截取了摄像头（480*720）视野内的最下方的 60 条像素。因为我们看到的“线”大多数都在视野的下半区。这样可以减少二值化的计算次数，减少性能的浪费。

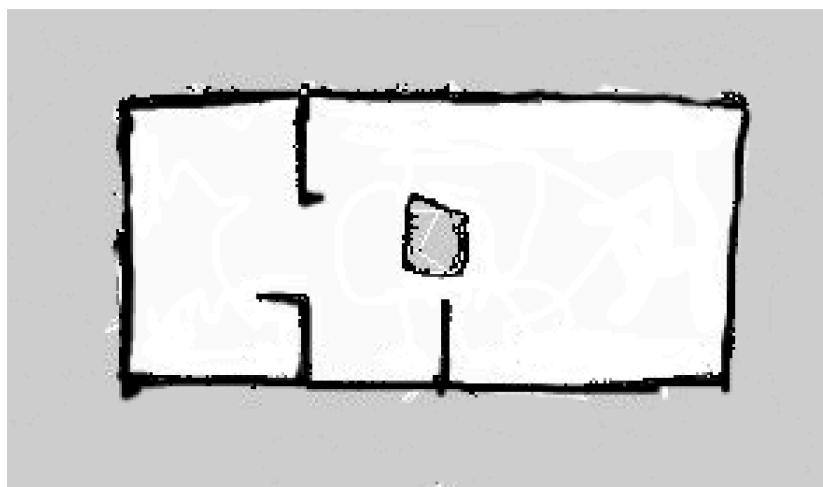
然后需要在轨迹寻迹的时候加入 PID 算法判断，使得小车转弯的时候尽量平滑。当然这一部分的参数调节需要自己手动。后来我发现其实如果要追求较快的速度，也可以在直线的时候使用 PID 算法，在接受的画面中通过计算直线在画面中的占比来确定是否可以加速。

适配于巡线小车的 PID 控制算法公式在下面已经给出， K_p 是比例增益， K_i 是积分增益， K_d 是微分增益， $e(t)$ 是我们小车的偏转状态变量。其中三个增益量是我们可以调整的参数。在本次实验中，我们经过多次的实验已经在最后调整出了最优的参数。这个参数被适用于我们的巡线模块。

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (1)$$

4.2 巡航模块

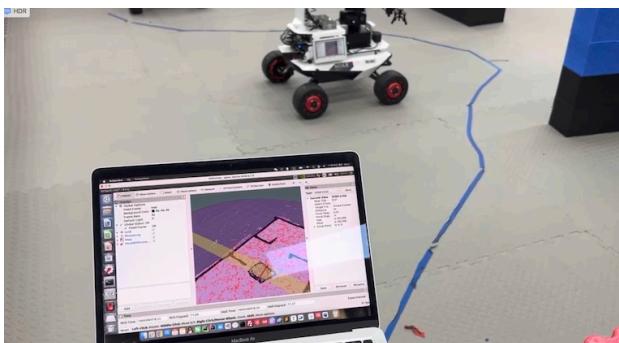
巡航模块是基于地图和激光雷达的。主要是 SLAM (Simultaneous Localization and Mapping, 简称 SLAM) 技术的应用。一开始我们会开启雷达，并且用遥控器控制机器人遍历一遍整个地图，直到整个地图都被雷达扫描完毕，生成一个数字地图（图片：10）。在我们建立好地图之后，会以我们的开始作为原点建立二维坐标系。并且在建图时还会生成对应的转向角度，这个角度在后面会用于搭建地图服务，让机器人知道自己在哪个地方。我们用一个三元组（X, Y, Orientation）来描述机器人当前位姿。



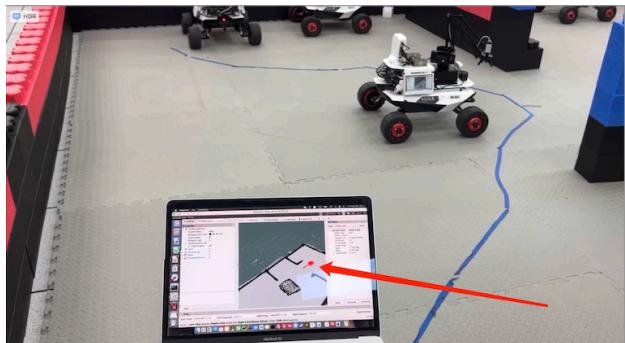
图片 10：使用激光雷达建模的地图

在巡线结束后，机器人进行 Global Localization 的操作，机器人会启动激光雷达，扫描接收到的激光信号，并且通过 AMCL（蒙特卡洛）粒子滤波定位算法来进行自身的全局重定位，在围绕场地后机器人会将自己的方向向量置为正确的方向。在此之后，机器人会开始自动按照巡航节点队列依次访问巡航，在巡航过程中如果视觉模块检测到了物体的出现，就会抛弃后面的巡航任务。转而进行抓取、回到起点的操作。

还有一点要注意就是，机器人在接受到一个目标节点的巡航任务时，会分别调用两个函数 global_planner 和 local_planner，这两个函数有一个参数 tolerance 控制小车在行进的过程中不会碰撞到墙壁。如果这个小车在巡航的时候靠近墙壁太近的话，会导致小车被墙“卡”住。所以在编程实践时，一方面要注意巡航节点的选取要远离墙壁，另一方面要在 scout_param 中调小 tolerance 的值，让小车不会离墙过远的时候认为自己遇到了障碍。



图片 11: 全局重定位开始



图片 12: 全局重定位结束

4.3 视觉模块

视觉模块主要是运用 Dabai 前置摄像头接收图像后，用我们的订阅节点来订阅机器人所发送出来的图像，并且使用计算机视觉亦或是深度学习、神经网络的方法对已有图像进行处理，并且提取图像中的信息。在本次实验中，我们对两种方法都有涉猎。实践证明，深度学习方法识别物体的准确度是相当的高的，但是深度学习框架是非常耗费资源的；计算机视觉方法是需要不断调整一些参数，但是计算速度快，节省系统资源特别在我们这种边缘计算的场景之下，是可以大大提高系统其他进程的运行速度的。当然还有一种对象检测算法可以应用在这个视觉模块上面，但是这个需要的资源也是相当多的，我们组只是做了相关的尝试并没有在最后任务的实现中使用这个物体检测模块。

深度学习 深度学习方法我们是训练了一个属于“物体”自己的神经网络。我们先对小熊进行了视频拍摄，然后将视频每一帧分出来，并且使用精灵标注助手²这个标注软件，制作了一个“小熊”的 VOC 数据集^[4]。然后我们用 Colab³白嫖了一些计算资源，训练了一个自己的 tiny-YOLOv3 神经网络^[5]。识别效果还是相当好的。下面有识别的效果图，图中的物体识别置信度为 **0.92**。但是有一个问题就是，如果开启了神经网络对于摄像头图片进行实时识别，那样剩余可用的内存容量会很少，无法开启其他的服务（如地图服务）。

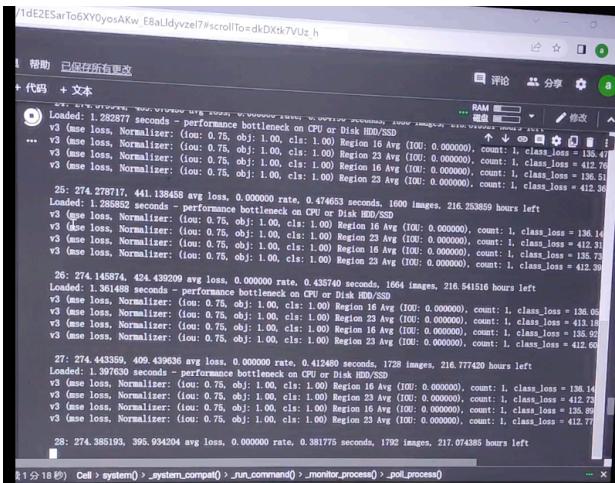
计算机视觉 这一种图像处理方法主要是使用 OpenCV 库对小熊的一些颜色特征进行提取。这种方法带有先验知识影响，即小熊是偏黄色的而我们的迷宫挡板多数都是黑色、蓝色、红色，使用类似图像遮罩的方法可以将小熊的颜色从背景板里面提取出来。然后我们一旦获取到了小熊对应的颜色信息，而且它的颜色占据了整

² 下载地址：<https://www.xiazaiba.com/html/115079.html>

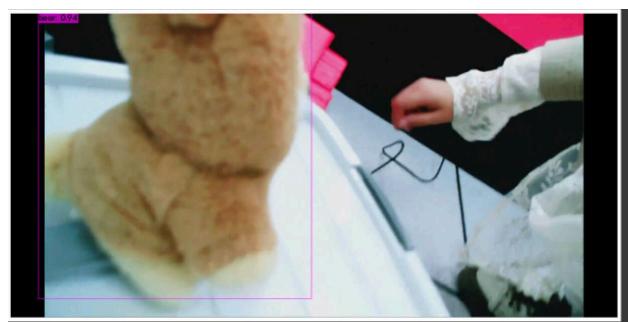
³ <https://colab.research.google.com>

个摄像头足够大的区域，我们就认为我们的小车已经在小熊周围了，接下来就会去启动靠近小熊并且抓取的功能。（可惜这部分实验拍摄的图片已经丢失，但是代码还在，会在附录一并给出）

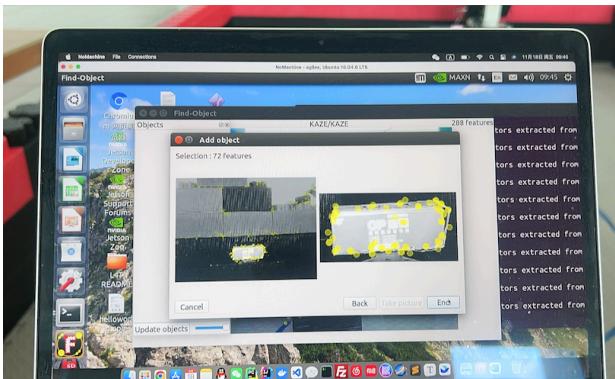
物体检测算法（模板匹配） 模板匹配主要是使用了 SURF、SIFT、ORB、FAST 等一系列算法，这一部分算法被适配进了 ROS 2 开源的库⁴ 进行物体的特征提取并且进行检测。我们在给定目标物体的图片的情况下，提取特征点，然后将特征点与视觉传入的物体的特征进行比对。而且这种算法还可以返回物体在图像中的具体位置在用单应性矩阵的变换^[3] 之后就可以得到物体在摄像头画面中的具体位置了。但是实践证明这种方法也不是特别好，因为基于特征点的匹配对光线的要求特别高，再者如果物体姿态与上一秒采样的姿势不同，算法就无法正确识别。（可惜这部分实验拍摄的图片已经丢失，但是代码还在，会在附录一并给出）



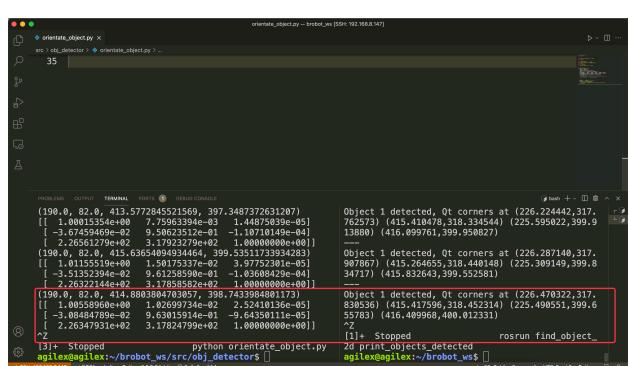
图片 13: 使用 Colab 进行神经网络训练



图片 14: 使用新训练的 tiny-YOLOv3 进行识别



图片 15: 模板匹配结果



图片 16: 计算物体的具体位置

4.4 Yolov3-Tiny 神经网络训练

数据集准备 通过 google 云端硬盘导入 images、ImageSets、annoutations 文件夹到 darknet 文件夹下。images 是图片集合，ImageSets 里面有一个次级文件夹：Main，Main 里有三个空文件 train，test，trainval，val，annoutations 是刚通过精灵标注助手打完且未变成 voc 所需的标签。并且通过代码将 annotation 的 xml 文件转化为 voc 需要的模式。

⁴http://wiki.ros.org/find_object_2d

配置训练前的参数 对训练集和测试集图片进行随机划分，并将路径写入四个 txt 文件中，形成了 txt 列表。并且生成对应的 label 文件。之后要修改 cfg 文件

1. 修改 cfg 文件夹，创建.data 和.names 文件
2. .names 文件中写入各个类型（要按之前标注标签时的类型顺序额写），这里只有 bear 一类，因此只写入 bear .data 文件中写入 train valid names 文件所在的位置，且需要修改 classes(类别数) 的参数。
3. yolo-tiny-vision_bear.cfg_train 中，将 net 修改为 train 模式，把步长设为 2000，lr 设为 0.001. 且将 classes 和 filters 进行修改，classes 修改为 1, filters 修改为 (classes+5)*3。yolo-tiny-vision_bear.cfg_test 中，只需把.cfg_train 中的 Testing 部分取消注释，Training 部分注释即可。
4. 将.data 文件，.names 文件移入 Vision_bear 文件下

训练 先下载预训练模型，之后只需要调用对应的训练命令即可。即训练是对于原有模型的微调。在训练之后我们会使用我们的测试图片进行测试。从我们的实验结果可以看出我们训练的网络识别准确率还是很高的。

代码 1: 模型测试输出

```

1 [yolo] params: iou_loss : mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00,
2 delta_norm: 1.00, scale_x_y: 1.00
3 Total BFLOPS 5.448
4 avg_outputs = 324846
5 Allocate additional workspace_size = 52.44 MB
6 Loading weights from /content/drive/MyDrive/Vision_bear/yolov3-tiny-Vision_bear_2000.weights...
7 seen 64, trained : 128 K-images (2 Kilo-batches_64)
8 Done! Loaded 24 layers from weights-file
9 Detection layer : 16 - type = 28
10 Detection layer : 23 - type = 28
11 /content/drive/MyDrive/Vision_bear/darknet/images/000056.jpg: Predicted in 4.850000 milli-seconds.
12 bear: 93%

```



图片 17: 模型测试的图片输出

其他具体的代码已经在分享链接中给出⁵。

4.5 抓取模块

抓取模块主要是调取给定软件包中的接口 (Brobot_SDK)^[7]。在我们靠近小熊之后，系统会自动启动在机械臂上面的摄像头，并且计算接收到的画面中小熊物体的中心。这个中心就是机械臂需要达到并且抓取的坐标。在接受坐标之后，机器人只需要调用 linear_movement() 函数进行运动即可。

4.6 模块整合后实验结果

整合后的实验流程框架就是在附录中给到的流程框图 (图片 21)。

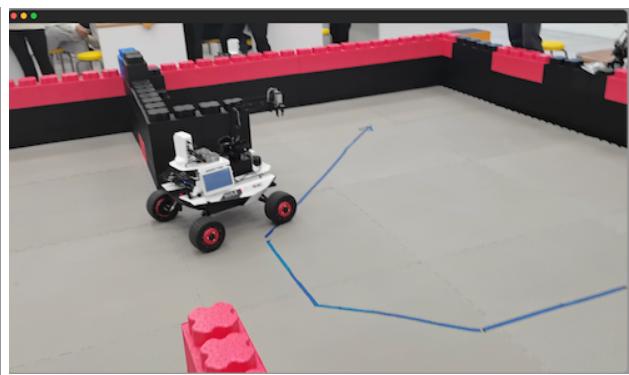
在本次实验中，我们把已有的流程拼接起来，就形成了整个的寻物的任务流程。我们在本次寻物任务中。先进行了巡线行走的任务，然后开始打开寻物所需要的服务节点，有：机械臂节点、前置摄像头视觉节点、底盘节点、雷达节点、地图服务节点。都打开之后就可以通过既定的程序进行运行了。

在整个实验中，我们在第 19 秒完成了巡线任务。此时，我们的程序调用了底层的 shell，开启了其他节点的服务。在第 50 秒的时候，我们的所有服务都开始运行，小车开始遍历所有规定好的所有节点。在小车巡航到第三个地图的标定的节点的时候 (1 分 44 秒)，视觉模块发现了小熊的存在。自此，小车程序将剩余的巡航节点任务都关掉了，开始进行靠近的任务。这个地方我们发现了一个问题就是靠近的时候，小车的速度可以调整的更快一些，这样可以减少我们在整个任务中所花费的时间。

在 2 分 25 秒的时候，我们的小车开始尝试抓取小熊，但是很可惜，因为机械臂标定的问题，我们的机械臂前爪都没有碰到小熊，在平常的实验中是可以抓起来的，究其原因是因为上次标定太长时间了，太长的时间间隔让机械臂丢失了精度。之后从 2 分 40 秒开始我们的小车开始执行回到起点的操作。在返回起点的过程中我们也卡壳了，因为对于碰撞的容忍度调整的有点低，所以中间在门柱的时候被卡住了一段时间。索性最后还是完成了，虽然存在一些失误，但是也算是比较成功的实验。



图片 18: 小车尝试抓取



图片 19: 小车回到出发点

关键部分具体代码详见附录。

⁵https://drive.google.com/file/d/1YCLXn2tdLbZ4lEXn0jRRH7mI_b9Qm38Z/view?usp=sharing

5 出现的问题以及解决方案

5.1 硬件问题

1. **问题:** 车载路由器因为接触不良经常出现断连、没有 5Gwifi 信号、信号质量不好的问题。因此 No Machine 并不能很好的链接机器人 (因为 No Machine) 需要传输画面信息。

解决方法: 后面我尝试使用 vscode 进行链接，vscode 不仅有很好的 ROS 2 扩展支持，而且还可以很清晰地展示文件架构，也可以在 vscode 上面控制终端的操作。^[2]

2. **问题:** 机械臂经常失去刚性，导致机械臂无法正常响应控制信号。

解决方法: 在每一次开机的之前，先将机械臂摆正到正确的初始化为止，再开机。这样机械臂的控制器就会在机械臂初始化的时候给予机械臂一定的刚性，并且机械臂可以正确响应。如果以上的操作无法解决机械臂的问题那就将机械臂从机器人上面写下来，重新进行标定，标定后机械臂即可恢复原状。

5.2 软件问题

1. **问题:** 开启过多的服务时，内存不够用，处理机资源不够用，可能导致一些程序出现致命错误，操作系统可能卡到崩溃。

解决方法: 后台存在着五个主要的进程，分别是 Ros-core, Map-server, Yolov3-tiny, Radar, Movement-Base。在这些进程中，Yolov3-tiny 需要的资源是最多的。所以可以给出两种解决方法：

- (a) 增大系统 swap 分区的大小，让更多不需要系统资源的程序交换到系统的硬盘的虚拟内存空间去。
- (b) 尽量减少神经网络的使用，比如我们可以在检测到黄色色块之后再开启神经网络，让神经网络检测我们看到的是不是小熊。

2. **问题:** 因为完成整个的任务需要开启过多的服务进程，但是中途又不可以使用鼠标键盘进行操作，找不到一种方法来控制各个进程的启动与关闭。

解决方法: 使用 Python 进行进程的控制。使用 Python 的 os 库^[6]，调用系统 shell 接口，利用 shell 开启其他的服务程序，并且记录返回的进程 ID 号在后面我们会用这些 ID 号来关闭某些进程，以节约系统资源。比如：我们在抓取任务完成之后，根据神经网络和视觉节点的 Process ID 关闭他们对应的进程。

5.3 网络问题

1. **问题:** 有的时候，我们的组上课的时候会到的比较晚，但是实验室内的 WiFi 带宽有限，甚至链接同一个 wifi 人多的时候会出现掉线的现象。所以有的时候会出现机器人无法连接到网络的问题。

解决方法: 一方面是可以开启手机热点，另一方面是可以早点去，连接到别的教室的 wifi，别的教室的 wifi 信号相对较好，而且使用的人也比较少。

2. **问题:** 下载某些软件包的时候需要访问国外的一些网站，比如 www.github.com, wiki.ros.org 等等，就会出现无法连接掉线的现象。

解决方法: 一方面我们小组使用清华等镜像软件站，通过访问国内网站下载相同版本的软件。另一方面我们学习了科学上网的方法，可以访问国外部分的网站学习和下载软件。

6 课程体会

在本次课程中，我们学习了基本的机器人的编程方法。通过两个课题实践学习到了如何统筹设计机器人控制框架。而且在课题二中我们学习了多个模块:Map_server, Move_base, Radar, SLAM 等等。我们先对这些模块分别进行编程实践，对模块的功能有了感性的认识。之后将模块糅合起来，组成了一个适合课题目标的小机器人。

其实我一开始小看了这个课程设计，感觉这个项目应该是很快就可以完成的，但是做到后面发现并不是那么简单。想要一键解决所有问题实在是太难了，不仅是编程的问题，而且还涉及了性能优化，语言特性，库函数调用，CMake 等等问题。一些问题是曾经没有遇到过的。只能是在实践中学习，摸索着石头过河。在这门课的前面四周，我上课的目标是完成文档中给定的任务就好，但是到了后面我发现仅仅使用上课的知识并不可以完全解决我。于是我开始寻找网上的学习资料。在 YouTube 上找到了一个教程⁶，是相当好的。同时可以多阅读一下文档，比如上面提到的 wiki.ros.org，里面提供了很多 API 的具体调用方法和用途。

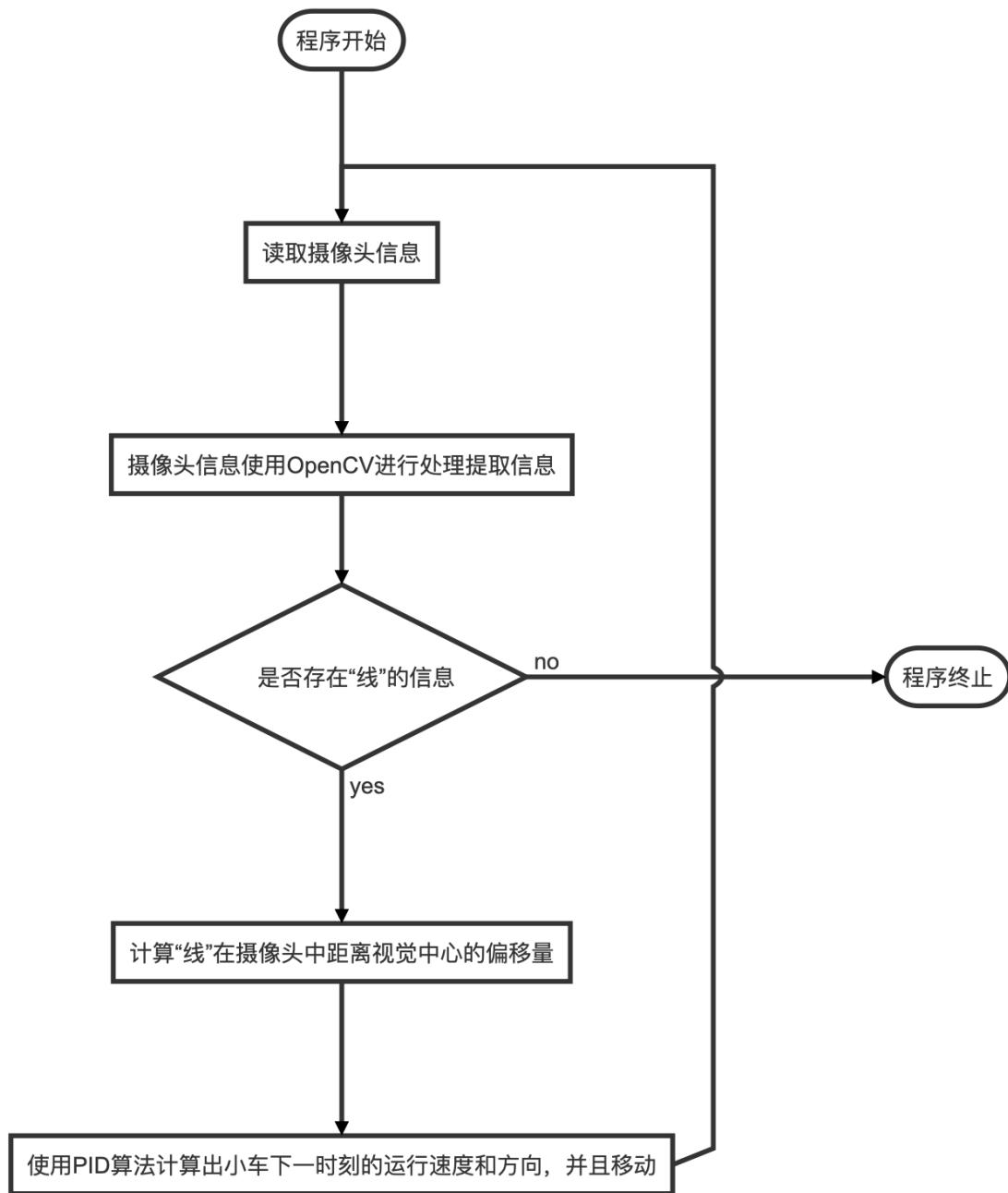
到了后面几周，基本就都扑在这个课程设计上面了，基本周末都回去实验室泡着调试机器人，也和很多别的组的大佬一起研究怎么可以把机器人跑起来。第一版的代码也就是这么产生的。当时也搜了很多网上的代码仓库想看看有没有借鉴的，结果还是没有。中间遇到了很多问题，一开始遇到了小机器人无法定位自我位姿的问题，后面是使用了 global_localization 的函数解决的，在后面考虑要从一个主进程作为入口，开启其他进程。这个使用的 python 的 os 库解决的。还学会了如何解决 CMake 的报错，以及如何安排 CMake。还解决了很多其他的问题，就不一一细说了。但是从这些解决问题的过程中我学到了很多。比如如何高效地利用搜索引擎，如何优雅的使用 ROS2 自带的文档等等。当然也学到了很多编码方面的能力，比如如何组织大型的工程文件，如何进行代码的版本管理等等。

最后，很感谢系里可以开这样的一门课程，作为为数不多的工程实践课，感觉学到的东西远比其他理论课学到的东西要更加生动，更加扎实。

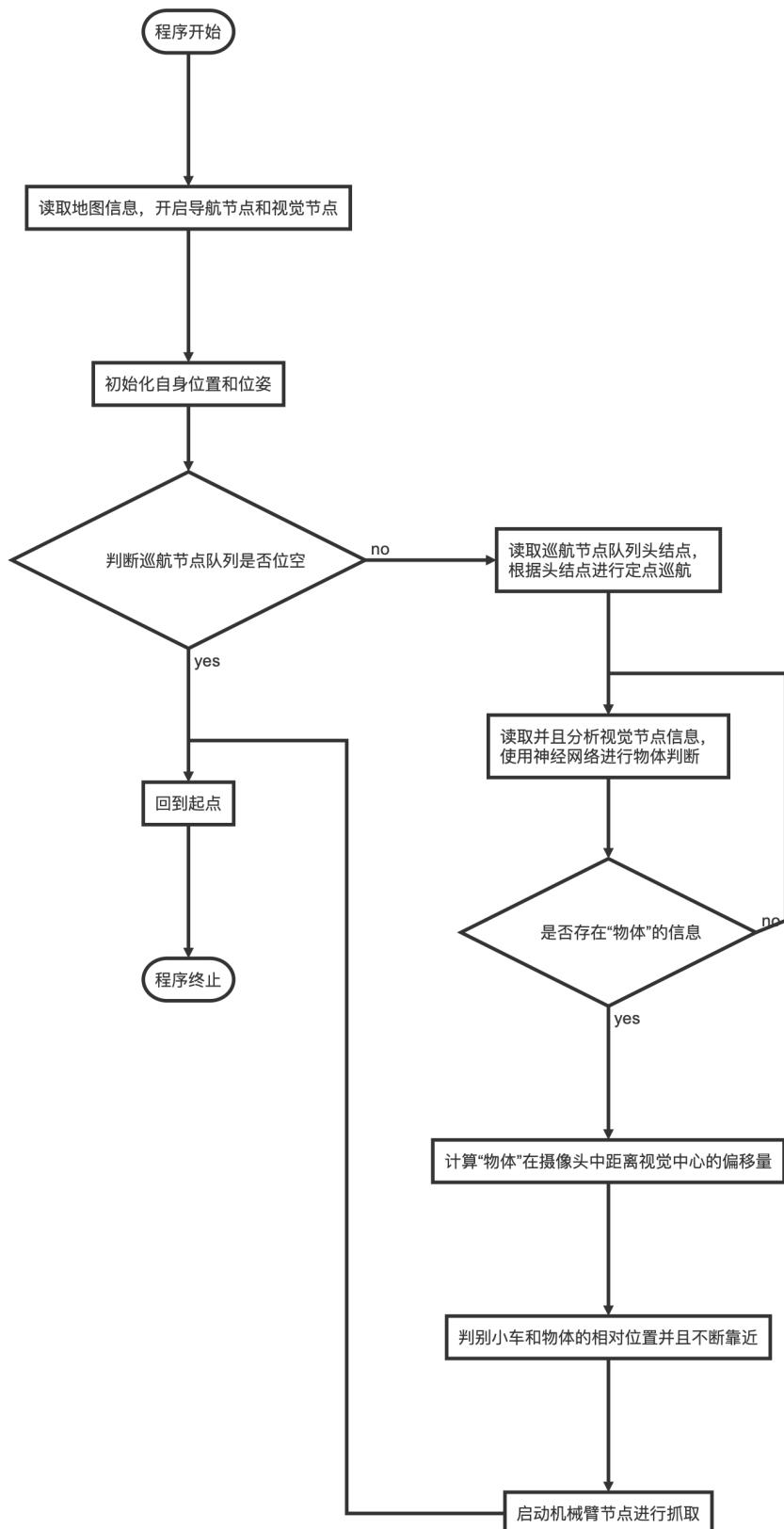
⁶<https://www.youtube.com/watch?v=0aPbWsyENA8&list=PLLSegLrePWgJupPUof4-nVFHGkB62Izy>

A 附录

A.1 图表



图片 20: task1: 巡线框架



图片 21: task2: 寻物框架

A.2 Task1—巡线关键代码

巡线关键代码

```

1 def image_callback( self , msg):
2     global last_erro , col_black = (73,43,46,124,255,255) # 蓝色的相反色调
3     image = self . bridge . imgmsg_to_cv2(msg, desired_encoding='bgr8')
4     image = cv2.resize(image, (320,240), interpolation =cv2.INTER_AREA) #提高帧率
5     hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV) # 转化为HSV
6     kernel = numpy.ones((5,5),numpy.uint8)
7     hsv_erode = cv2.erode(hsv,kernel , iterations =1)
8     hsv_dilate = cv2. dilate (hsv_erode,kernel , iterations =1)
9     lowerbH,lowerbS,lowerbV=col_black[0],col_black [1], col_black [2]
10    upperbH,upperbS,upperbV=col_black[3],col_black[4],col_black[5]
11    mask=cv2.inRange(hsv_dilate,(lowerbH,lowerbS,lowerbV),(upperbH,upperbS,upperbV))
12    masked = cv2.bitwise_and(image, image, mask=mask)
13    # 在图像某处绘制一个指示，因为只考虑20行宽的图像，所以使用切片将以外的空间区域清空
14    h, w, d = image.shape
15    search_bot = search_top = h
16    mask[0:search_top, 0:w] = 0
17    mask[search_bot:h, 0:w] = 0
18    M = cv2.moments(mask) # 计算mask图像的重心，即几何中心
19    if M['m00'] > 0:
20        cx = int(M['m10']/M['m00'])
21        cy = int(M['m01']/M['m00'])
22        cv2. circle (image, (cx, cy), 10, (255, 0, 255), -1)
23        if cv2. circle :      # 计算图像中心线和目标指示线中心的距离
24            erro = cx - w/2-15
25            d_erro=erro-last_erro# 计算误差
26            self . twist . linear .x = 0.18
27            if erro!=0: self . twist . angular.z = -float(erro)*0.004-float(d_erro)*0.000 #转弯
28            else : self . twist . angular.z = 0 # 不转弯
29            last_erro=erro
30        else :
31            self . twist . linear .x = 0
32            self . twist . angular.z = 0
33            rospy.signal_shutdown("closed!")
34            self . cmd_vel_pub.publish(self . twist )

```

A.3 Task2—寻物关键代码

代码 2: 基本巡航框架

```
1 def goal_pose(pose): # <2>
2     goal_pose = MoveBaseGoal()
3     temp = tuple(quaternion_from_euler(0, 0, np.deg2rad(pose[3])))
4     goal_pose.target_pose.header.frame_id = 'map'
5     goal_pose.target_pose.pose.position.x = pose[0]
6     goal_pose.target_pose.pose.position.y = pose[1]
7     goal_pose.target_pose.pose.position.z = pose[2]
8     goal_pose.target_pose.pose.orientation.x = temp[0]
9     goal_pose.target_pose.pose.orientation.y = temp[1]
10    goal_pose.target_pose.pose.orientation.z = temp[2]
11    goal_pose.target_pose.pose.orientation.w = temp[3]
12    return goal_pose
13
14 Flag = 0
15 rospy.init_node('patrol')
16 client = actionlib.SimpleActionClient('move_base', MoveBaseAction) # <3>
17 client.wait_for_server()
18
19 def detect_call_back():
20     global client
21     print("call_back entered.")
22     x, y, w, h, area = xywhs[0]
23     if area > 0.0:
24         print("detected.")
25         client.cancel_goal()
26         client.cancel_all_goals()
27         os.chdir('/home/agilex/zycatkin_ws')
28         open_all_subprocess_in_one_go('/home/agilex/zycatkin_ws')
29         rospy.signal_shutdown("stop going around!")
30     else:
31         print("no detected.")
32
33 if __name__ == '__main__':
34     cmd_vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=1)
35     twist = Twist()
36     f = Follower()
37     client.send_goal(goal_pose(waypoints[0]))
```

```
38 while True:  
39     dector_call_back()  
40     for pose in waypoints[1:]:    # <4>  
41         print("goal:x=%f y=%f" % (pose[0], pose[1]))  
42         goal = goal_pose(pose)  
43         state = client.get_state()  
44         while state != GoalStatus.SUCCEEDED:  
45             state = client.get_state()  
46         if state == GoalStatus.SUCCEEDED:  
47             print("reached!")  
48             client.send_goal(goal)  
49         print(client.get_result())  
50         rospy.sleep(1)
```

代码 3: 位置重定位代码

```
1
2 class localize():
3     def __init__(self):
4         self.pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
5         self.sub = rospy.Subscriber(
6             '/amcl_pose', PoseWithCovarianceStamped, self.callback)
7         self.move = Twist()
8         self.cov = PoseWithCovarianceStamped()
9         rospy.sleep(1)
10
11    def serv_client(self):
12        wait = rospy.wait_for_service('/global_localization')
13        serv_clie = rospy.ServiceProxy('/global_localization', Empty)
14        req = EmptyRequest()
15        result = serv_clie(req)
16
17    def sqr_mov(self):
18        self.serv_client()
19        rospy.sleep(1)
20        print(self.cov.pose.pose.position)
21        z = (self.cov.pose.covariance[0] +
22             self.cov.pose.covariance[7] + self.cov.pose.covariance[-1])
23        print('cov', z)
24        count = 8
25        while count > 0:
26            self.move_straight()
27            rospy.sleep(0.1)
28            self.rotate()
29            rospy.sleep(0.1)
30            count -= 1
31            z = (self.cov.pose.covariance[0] + self.cov.pose.covariance[7] \
32                 + self.cov.pose.covariance[-1])
33            print(self.cov.pose.pose.position)
34            print('cov2', z)
35        if z < COV_THRESHOLD:
36            print('Robot localized itself')
37            print('final position', self.cov.pose.pose.position)
38        else:
39            self.sqr_mov()
```

```
40
41 def move_straight( self ):
42     count = 35
43     while count > 0:
44         self .move.linear.x = 0.15
45         self .move.angular.z = 0
46         self .pub.publish( self .move)
47         rospy.sleep (0.1)
48         count -= 1
49
50 def rotate( self ):
51     count = 26
52     while count > 0:
53         self .move.linear.x = 0.1
54         self .move.angular.z = 0.75
55         self .pub.publish( self .move)
56         rospy.sleep (0.1)
57         count -= 1
58
59 def callback( self , msg):
60     self .cov = msg
61     return self .cov
62
63 if __name__ == "__main__":
64     rospy.init_node('husky_square')
65     x = localize()
66     x.sqr_mov()
```

代码 4: 向着物体的方向靠近

```
1 class Detector:
2     def __init__(self):
3         self.detector = rospy.Subscriber(
4             "/objects", Float32MultiArray, self.dector_call_back, queue_size=10)
5         self.cmd_vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=1)
6         self.twist = Twist()
7
8     def fetch_point(self, msg):
9         """
10         获取object发布节点返回的数据并且通过坐标变换转化成目标在摄像头中的占比
11         image size width = 1280 height = 720
12         """
13
14     def compute_Point(data, x, y):
15         """
16         单应矩阵对应坐标点的变换
17         """
18         _x = data[3] * x + data[6] * y + data[9]
19         _y = data[7] * y + data[4] * x + data[10]
20         return (_x, _y)
21
22     def compute_pct(Point):
23         return (Point[0] / ImageWidth, Point[1] / Imageheight)
24     # 单应性矩阵求解
25     data = list(msg.data)
26     obj_num = data[0]
27     width, height = data[1], data[2]
28     qtTopLeft = compute_Point(data, 0, 0)
29     qtTopRight = compute_Point(data, width, 0)
30     qtBottomLeft = compute_Point(data, 0, height)
31     qtBottomRight = compute_Point(data, width, height)
32     return list(map(compute_pct, [qtTopLeft, qtTopRight, qtBottomLeft, qtBottomRight]))
33
34 @staticmethod
35     def compute_center(Points):
36         """
37         return center_x, center_y
38         """
39         point_set = zip(*Points)
```

```
40     return tuple(map(lambda x: sum(x) / len(x), point_set))
41
42 @staticmethod
43 def compute_coverage(Points):
44     x1, y1 = Points[3]
45     x2, y2 = Points[0]
46     return ((x2-x1)*(y2-y1))
47
48 def dector_call_back(self, msg):
49     Points = self.fetch_point(msg)
50     # print("Points:", Points[0], Points[3])
51     center_x, _ = self.compute_center(Points)
52     Obj_coverage = self.compute_coverage(Points)
53     print("center_x : ", center_x)
54     print("Obj_coverage", Obj_coverage)
55     Bias = center_x - 0.5
56     print("Bias: ", Bias)
57     if abs(Bias) > 0.05:
58         if Bias > 0:
59             print("向右转")
60         else:
61             print("向左转")
62         self.twist.angular.z = -float(Bias) * 0.2
63         self.twist.linear.x = 0.05
64         self.cmd_vel_pub.publish(self.twist)
65     else:
66         self.twist.angular.z = 0
67         self.twist.linear.x = 0
68         self.cmd_vel_pub.publish(self.twist)
69
70
71 if __name__ == '__main__':
72     rospy.init_node("object_detector")
73     dector = Detector()
74     rospy.spin()
```

参考文献

- [1] SLAM 导航机器人零基础实战系列 https://blog.csdn.net/hiram_zhang/article/details/88374512
- [2] 在 VScode 中配置 ROS 环境的详细过程 <https://www.jb51.net/article/255449.htm>
- [3] 单应性矩阵的求解 <https://zhuanlan.zhihu.com/p/163170554>
- [4] 手把手教你自制 VOC, COCO 数据集 <https://zhuanlan.zhihu.com/p/556197423>
- [5] 使用自己的数据训练 yolov3-tiny 模型 <https://blog.csdn.net/fkbiubiubiu/article/details/120295134>
- [6] Python 内置 OS 库使用文档 <https://docs.python.org/3/library/os.html>
- [7] 复合教育机器人套件用户手册 V1.2.pdf