

基于ros的图像识别小车

1850061 阮辰伟
1951328 曹峰源
1951095 梁伊雯

基于ros的图像识别小车

项目说明

运行环境

开发环境

环境依赖

运行方式

系统架构

系统架构图

架构详解

后端

中间层

前端

界面设计

WEB端

WEB端界面设计

APP端

APP端界面设计

存在的问题

效果展示

功能实现

运动控制

功能描述

效果展示

建图

gmapping 2D建图

功能描述

原理简介

rtabmap_ros + kinect 3D建图

图

功能描述

原理简介

效果展示

建图结果

自动建图

功能描述

效果展示

自主导航+自动避障

功能描述

原理简介

定位 (amcl)

全局路径规划 (astar)

局部路径规划 (TEB)

效果演示

yolov4 目标检测

功能描述

原理简介

优化

效果演示

项目说明

本项目选题为“基于ROS的图像识别小车“，根据题意，项目详细设计方案主要包括”ROS“、”图像识别“、”人机交互“三大方面。在”ROS“方面，我们主要在ROS上做了二维建图、三维建图、自动建图、导航、边建图边导航等工作；在”图像识别“方面，我们收集了自定义的交通数据集，并在此基础上对比了Faster RCNN、YoloX、YoloV4三个模型，最后在YoloV4模型上作出了相应改进；在“人机交互”方面，我们能Web和手机APP两种方式对小车的运动进行可视化，同时还可以通过APP来实现前后端数据互传，实现对小车的基本控制。在这三个方面上，我们都取得了相当不错的表现。同时，本项目完成了当时设置的基本要求与加分要求，并做出了很多合理的尝试和改进。总而言之，经过了这一个学期的学习、实践与汇报，本项目产出了一个较为完整的交互系统，项目完成度能够达到甚至超出了我们的初始预期。

| 要求 | 详情 | 完成情况 |
|--------|---------------|--|
| 基本要求 | 小车正常运动 | 1. 电脑键盘控制 2. app按键控制 |
| | 使用摄像头捕捉画面 | 完成 |
| | 识别面前物体并做出动作 | 完成 |
| 加分项 | 一次性接受多条命令统一执行 | 1. 上左按键同时按下实现左转 2. 边运动边识别物体 3. 边导航边建图 4. 等等 |
| | 对比不同识别算法 | 1. Faster R-CNN 2. YoloX 3. YoloV4 4. YOLOv4-kmeans-anchor 5. YOLOv4-DE-anchor |
| | 自己训练数据集 | 交通数据集，由三大类组成： 1. 交通灯：红灯、黄灯、绿灯 2. 交警手势：向前、停止、直走 3. 道路标志牌：限速、人行横道、停止 |
| 其他合理改进 | 建图 | 1. Gmapping 2D建图 2. rtabmap_ros + kinect 3D建图 |
| | 导航 | 1. 能够实现边导航边建图 2. 自动规划路径 3. 自动避障 4. 自动建图 |
| | 前端展示 | 1. WEB端 2. APP端 |
| | app智能交互 | 1. 天气查询 2. 语音控制 3. 路况查询 4. 新闻获取 5. 尾号限行 6. 气象灾害 |

运行环境

开发环境

- 开发环境: *Ubuntu 18.04* + *ros* (melodic版本)

- 开发软件:

vscode

- 开发语言

Python 2.8 , c++ , rospy , roscpp

环境依赖

```
sudo apt install ros-melodic-desktop-full
sudo apt-get install ros-melodic-rtabmap-ros
sudo apt-get install ros-melodic-rosbridge-suite
sudo apt-get install ros-melodic-driver-base
sudo apt-get install ros--melodic-gazebo-ros-control
sudo apt-get install ros--melodic-effort-controllers
sudo apt-get install ros--melodic-joint-state-controller
sudo apt-get install ros-melodic-ackermann-msgs
sudo apt-get install ros-melodic-global-planner
sudo apt-get install ros-melodic-teb-local-planner
...
```

上述依赖可能存在包含关系，这里都列出只是为了避免漏装的情况发生

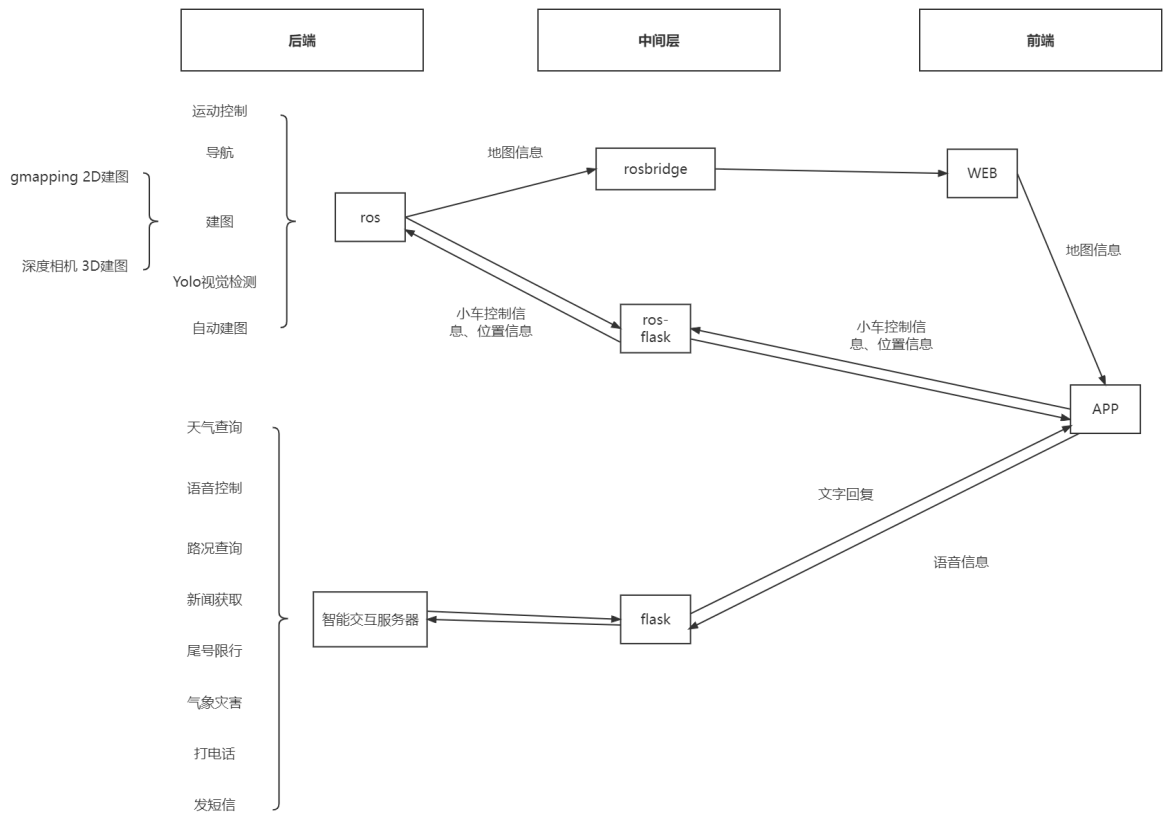
运行方式

```
# 打开gazebo, 开启建图、导航、目标检测等功能
roslaunch racecar_gazebo mango_auto.launch
# 启动ros-flask, 使得app发送的程序能够被ros接受
roslaunch racecar_gazebo ros_web_socket.py
# 打开rviz, 观察小车状态
roslaunch racecar_gazebo auto_run.launch
# 打开web端, 使得能通过访问特定网页获取小车信息、地图信息等
roslaunch rvizweb rvizweb.launch

# 目标网页
http://localhost:8001/rvizweb/www/index.html
```

系统架构

系统架构图



架构详解

后端

后端中最重要的部分为ros部分，其功能包括导航（附带自主避障模块），建图（包括gmapping 2D建图以及深度相机 3D建图）以及Yolo视觉检测模块。

后端还包括一个智能交互服务器，他为app提供了天气查询、语音交互、路况查询等智能服务

中间层

中间层为rosbridge以及ros-flask。

rosbridge负责连接ros端与WEB端，主要负责地图信息的单向传输。

ros-flask负责连接ros端与APP端，主要负责控制信息以及小车位置信息的双向传输。

前端

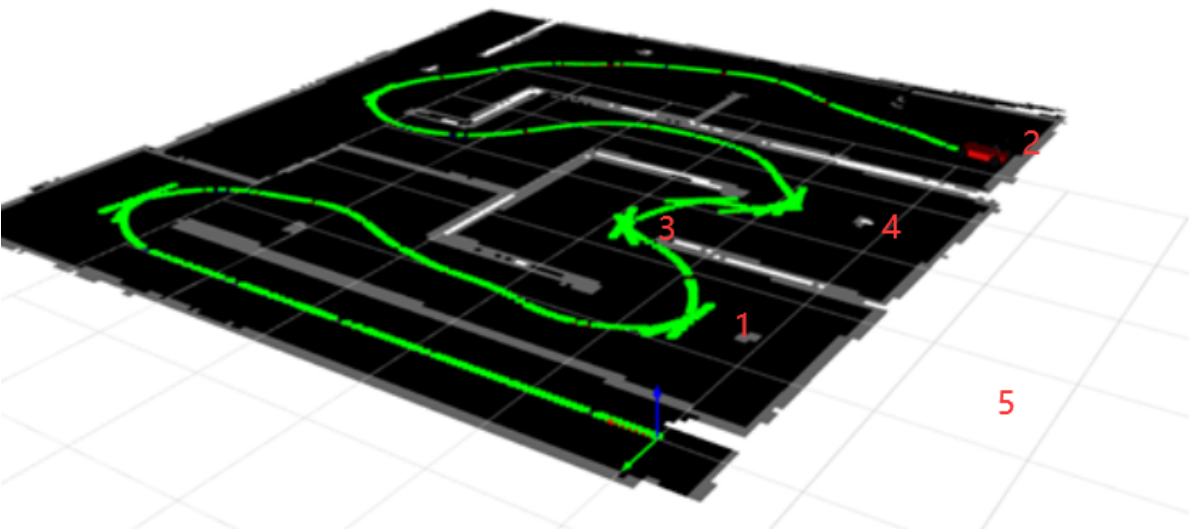
WEB端主要负责渲染地图信息、小车的信息等信息。这些信息由WEB端汇聚后再传输至APP端，因为APP端不方便直接渲染此类信息。

APP端为最终的展示与交互终端，其展示信息包括小车位置信息，地图信息，小车信息等。他可以与小车进行语音交互以及按键交互。

界面设计

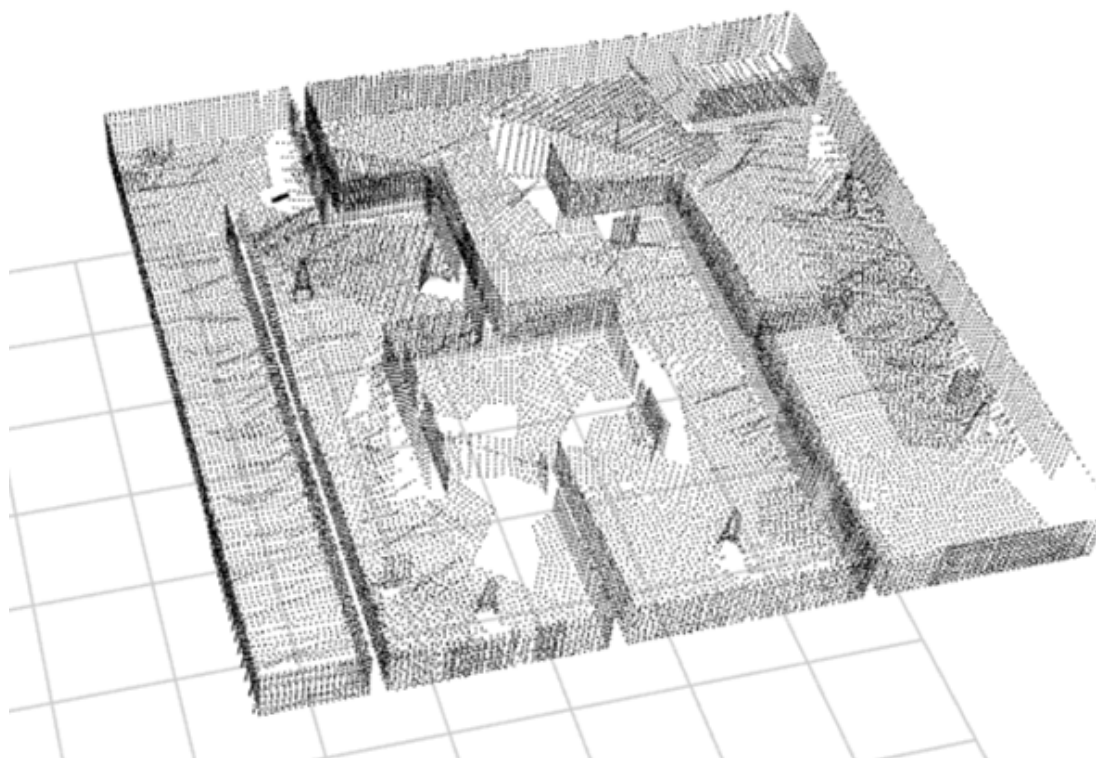
WEB端

WEB端界面设计



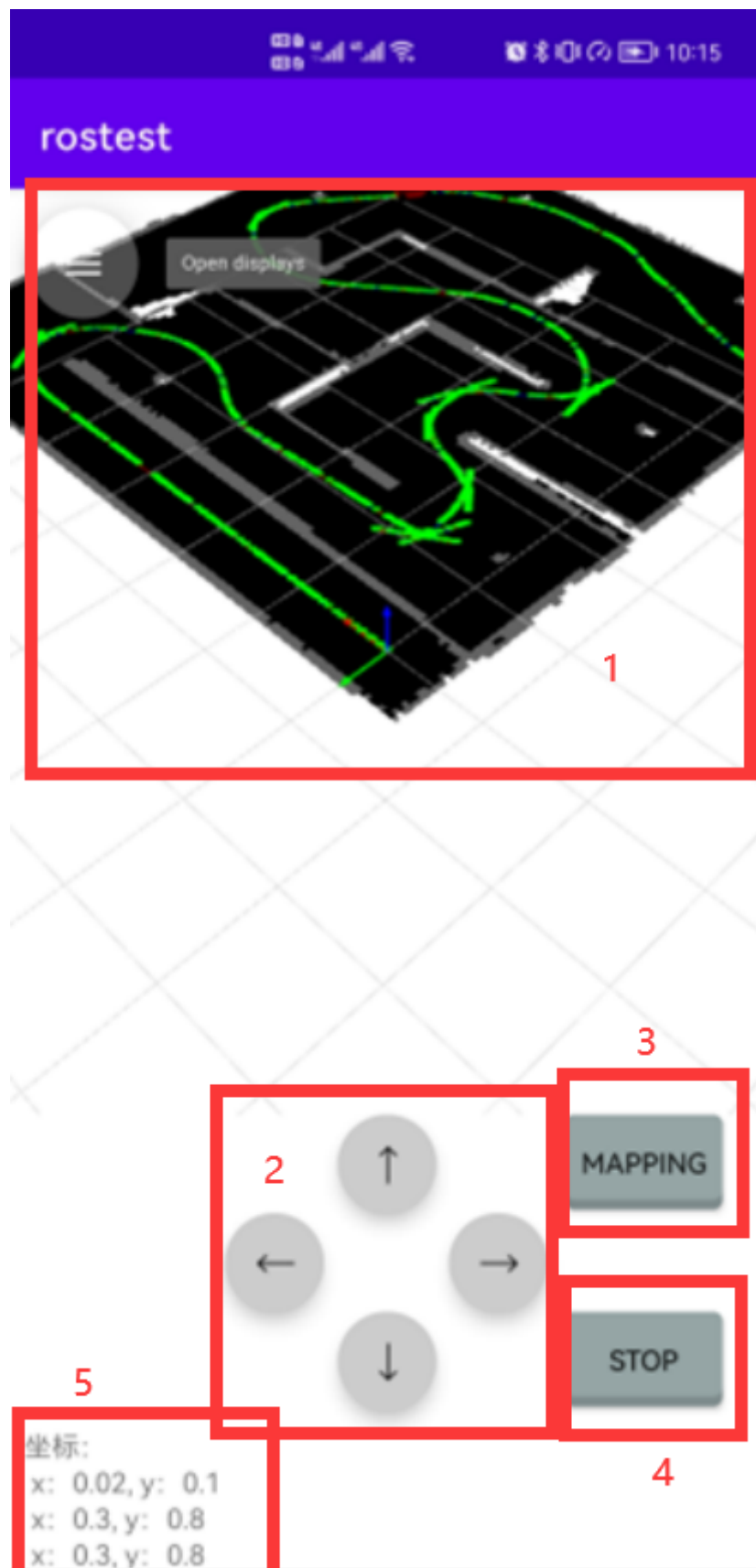
| 序号 | 物体 | 解释 |
|----|------|---------------------------------------|
| 1 | 地图 | 图中黑色部分为地图已知部分状况，即可以行驶的部分 |
| 2 | 小车 | 途中红色长方体状的物体代表我们的设备小车，其上有着深度相机、激光雷达等设备 |
| 3 | 轨迹 | 图中绿色部分为小车行驶的轨迹，即小车行驶过的历史轨迹 |
| 4 | 地图边界 | 图中灰色部分为地图边界，或者是障碍物的边缘位置 |
| 5 | 未知区域 | 图中白色部分为地图未知区域 |

上图展示了gmapping建图时WEB端的状态，下面展示 `rtabmap_ros + kinect` 3D建图 时web端的样子。由于大多数元素与上图相似，我就不一一介绍了。



APP端

APP端界面设计



| 序号 | 物体 | 解释 |
|----|--------|-------------------------|
| 1 | web端信息 | WEB端传输来的地图信息、轨迹信息、小车信息等 |
| 2 | 小车控制按键 | 可以通过这个案件控制小车移动 |
| 3 | 开始建图按键 | 点击此按键可以开始建图 |
| 4 | 停止建图按键 | 点击此按键可以停止建图 |
| 5 | 小车坐标信息 | 通过该单元可以查看小车的当前坐标与历史坐标 |

存在的问题

1. app端延迟比较大，尤其是轨迹模块的显示。导致其控制信息与ros端的信息都无法及时得到反馈与展示
2. 前端暂时没实现导航功能，在以后的设计中会逐步加入

效果展示

见文件夹

- 演示视频/app_2d建图.mp4
- 演示视频/app_3d自动建图.mp4
- 演示视频/app 按键控制 运动与建图.mp4
- 演示视频/app_语音控制.mp4

功能实现

运动控制

功能描述

可以通过上下左右键控制小车的运动以及转向

效果展示

为了更加清晰地展示运动控制效果，这里采用了按键输入0, 1, 2, 3的方式进行运动控制。实际上还可以通过按钮交互或是直接按wasd等方式进行运动控制。

详情见演示视频文件夹下的的运动控制.mp4文件

- 演示视频/运动控制.mp4
- 演示视频/app 按键控制 运动与建图.mp4

建图

gmapping 2D建图

功能描述

通过该功能，我们能够得到一张2D的地图。地图由**安全区域**、**未知区域**以及**障碍物**这三个部分组成。我们可以根据该结果进行导航。

原理简介

Gmapping是一个基于**2D激光雷达**使用**RBPF**（Rao-Blackwellized Particle Filters）算法完成**二维栅格地图**构建的SLAM算法。

优点：

1. gmapping可以实时构建室内环境地图，在小场景中计算量少，且地图精度较高
2. 对激光雷达扫描频率要求较低。

缺点：

1. 随着环境的增大，构建地图所需的内存和计算量就会变得巨大，所以gmapping不适合大场景构图。

rtabmap_ros + kinect 3D建图

功能描述

通过该功能，我们能够实现3D建图，相比于gmapping 2D建图，这种建图方式能够保留更多的地图信息。这样的结果便于我们进行yolo目标检测

原理简介

RTAB-Map (Real-Time Appearance-Based Mapping)是一种基于全局贝叶斯闭环检测的RGB-D Graph SLAM方法。它可以用kinect的深度信息结合kinect变换得到的激光数据进行即时定位与建图，而gmapping算法只用到了kinect转换得到的激光数据，而把深度信息丢弃了。


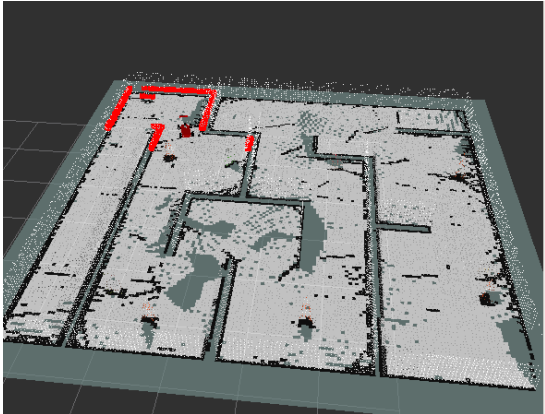
效果展示

详情见演示视频文件夹下的的建图+web端.mp4文件

- 演示视频/建图+web端.mp4

这里展示的是gmapping建图以及对应的WEB端变化。rtabmap_ros + kinect 3D建图这种建图方式为了保证建图质量，需要在地图上来回运动多次。导致视频总大小超过500mb，canvas难以上传。因此我们只选择提交了gmapping建图的demo展示。

建图结果

| gmapping建图结果 | rtabmap_ros + kinect 3D建图结果 |
|---|--|
|  |  |

可以看出，gmapping算法的出的地图是2d的，但比较完整

rtabmap_ros + kinect的见图结果是3d的，包括更多信息，但是更加耗时。而且因为视觉死角等问题，地图上死角比较多。

自动建图

功能描述

我们使用explore_lite包实现了自动建图技术，该包接受地图信息/map，同时根据贪心的方法决定接下来的运动方向。

但是由于我们参数调节的问题，小车在转角处会反复徘徊，不知道接下来的方向应该选择哪里。这有待进一步调整。

效果展示

- 演示视频/自动建图.mp4

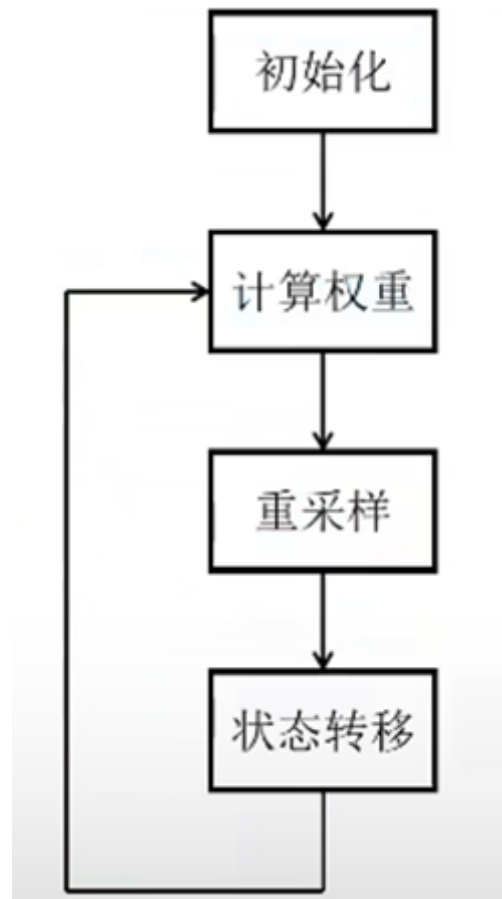
自主导航+自动避障

功能描述

我们可以为小车指定目标点以及目标位姿。此时，系统就会帮助我们规划出一条路径前往目标点。中途遇到障碍物会进行自主避障。

原理简介

定位 (amcl)



接下来是步骤理解

(1)初始化粒子群

在整个场地内，将总粒子数 N 进行均匀分布。

(2)小车开始运动(同时每运动一步进行一次测量)

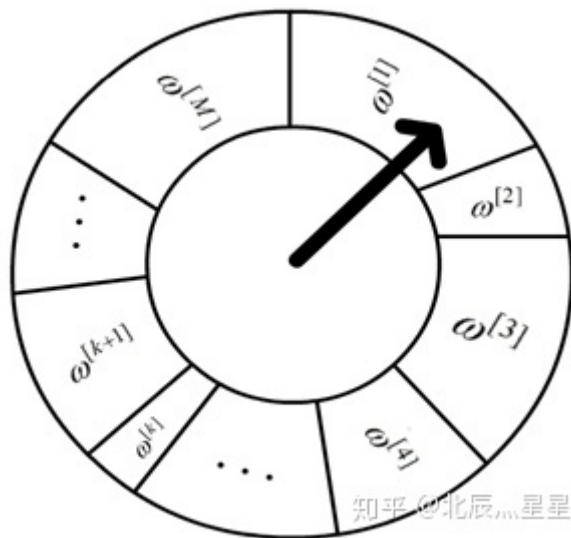
小车按照控制给定的叠加了控制噪声的运动方程进行运动，运动达到下一位置后，传感器对当前位置进行测量，并得到一个测量的位置(不准确,含有测量噪声)

(3)粒子群更新(预测步骤)

把粒子群中的全部粒子逐个带入小车的运动方程中，得到粒子群的下一步位置。同时，计算此时每个粒子的位置和测量得到的小车位置 这两个位置之间的几何距离，按照距离的不同给每个粒子添加一个权重，用于重采样。显然，距离越近，权重越大（权重和距离关系的函数，这里采用高斯分布钟形曲线的右侧，即随距离增大，权重单调递减）。下一步，得到全部粒子的权重后，将它们进行归一化。

(4)重采样

对于全部 M 个粒子，它们的归一化权重集合为 ω ，第 i 个粒子的权重为 $\omega[i]$ 。则重采样过程可理解为旋转轮盘抽奖。如下图。



由于不同粒子的归一化权重不同，它们占轮盘的面积也不同，因此权重大的粒子更容易被抽中。

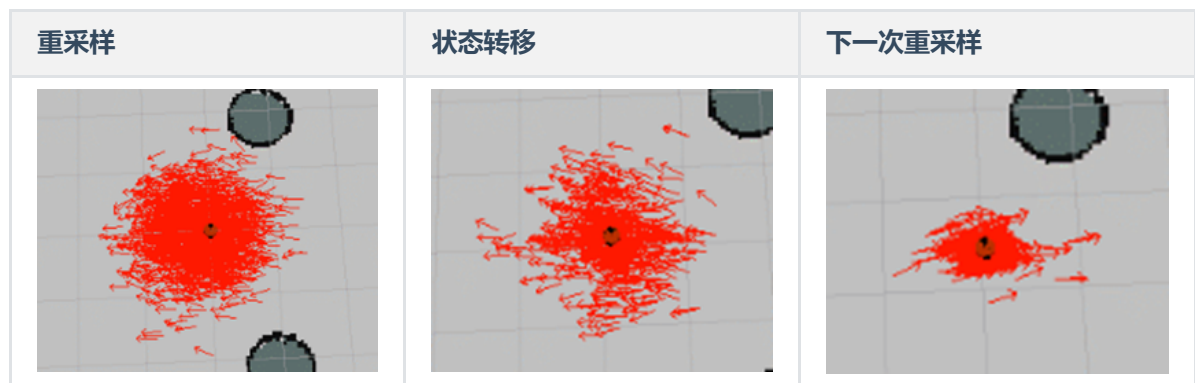
现在，我们需要重采样得到新的M个粒子组成的粒子群。因此，我们按照上述轮盘，抽M次，抽到某个权重 $\omega[i]$ ，则把该权重对应的粒子放入新的M个粒子组成的粒子群中。这样，那些权重大的粒子可能被反复抽到，从而重复出现在新的粒子群中，而那些权重小的粒子可能会在新的粒子群中被丢弃。

某一次运动过程结束后，可得出小车实际位置、重采样后粒子群位置、以及粒子群的几何中心位置

(5)重复步骤(2)~(4)，直到结束。

最后，我们可以得到整个运动过程中，小车实际路径、测量得到的路径、粒子群中心位置构成的路径。

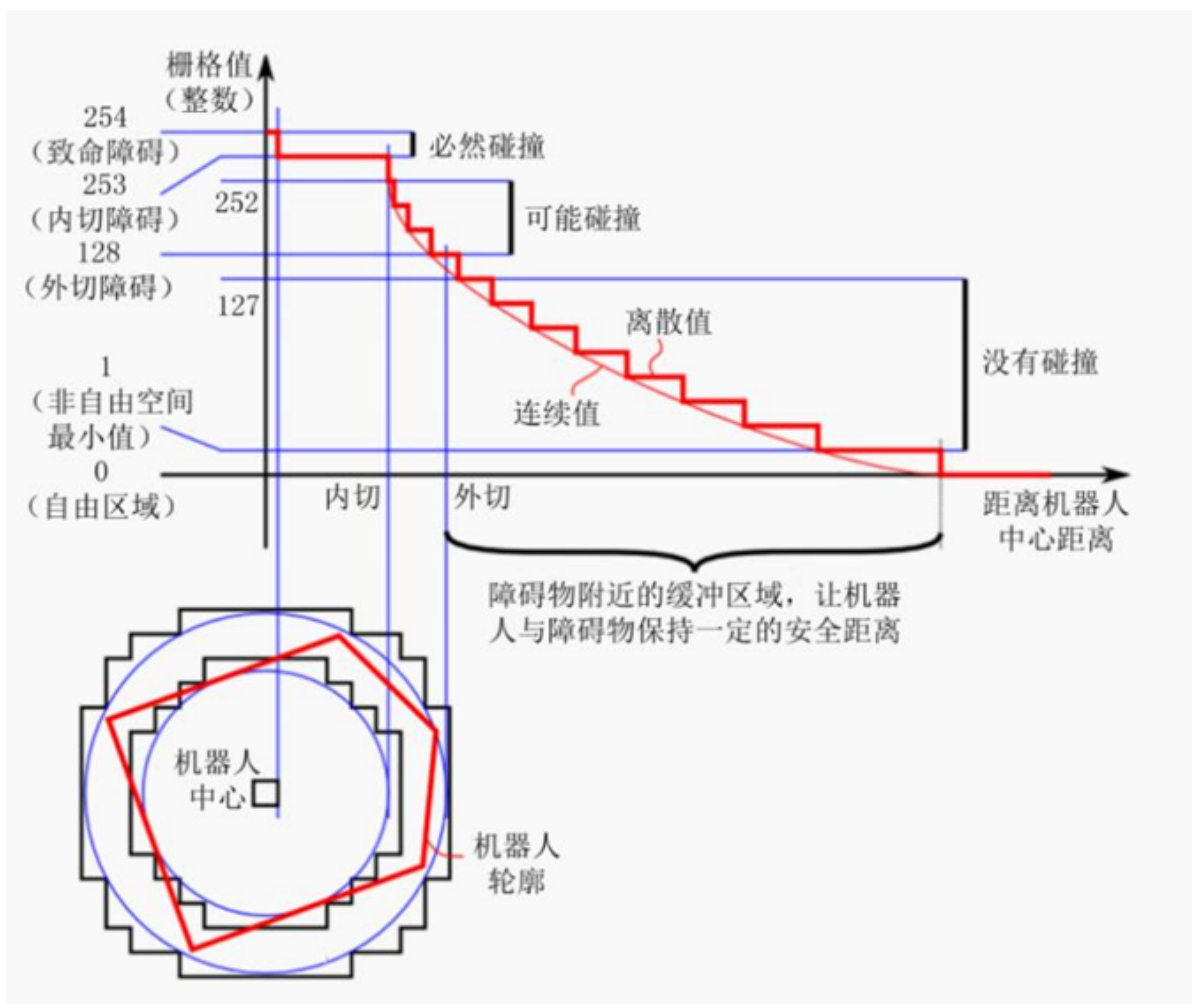
实际的过程大概如下所示：



全局路径规划 (astart)

a*算法较为常见，这里就不多加介绍了。

但这里需要提的一点是，a*算法需要根据路径计算得分，从而选择预期分数最高的路径。在这里，我们使用代价地图的方法计算每个小格子的分数，从而确认路径的分数。而代价地图的计算方式与小车距离障碍物的距离相关。



| 栅格值 | 障碍类型 | 描述 |
|-----------|-------|------------------------------------|
| 254 | 致命障碍 | 障碍物与机器人中心重叠，必然发生碰撞 |
| 253 | 内切障碍 | 障碍物处于机器人的内切圆内，必然发生碰撞 |
| [128,252] | 外切障碍 | 障碍物处于其机器人的外切圆内，处于碰撞临界，不一定发生碰撞 |
| (0,127] | 非自由空间 | 机器人处于障碍物附近，属于危险警戒区，进入此区域，将来可能会发生碰撞 |
| 0 | 自由区域 | 此处机器人可以自由通过 |
| 255 | 未知区域 | 还没探明是否有障碍物 |

局部路径规划 (TEB)

TEB算法就是在全局路径中以固定的时间间隔插入N个状态点，让路径变成一条可以形变的橡皮筋，然后再给它施加一个约束。每个约束可以看作橡皮筋的外力，给橡皮筋施加力以后，橡皮筋会形变，这种形变就是它内部的优化算法。通过这个优化变形，就会找到满足各种约束的最终可行路径。

约束条件/目标函数（常见的四种）：

1. 跟随路径+避障
拉回来+推出去
2. 速度/加速度约束
两个状态点之间，直接用差分近似计算

$$V_{min} \leq f_v(B) \leq V_{max}$$

$$a_{min} \leq f_a(B) \leq a_{max}$$
3. 运动学约束
若干弧段组成的平滑轨迹，(最小转弯半径)
4. 最快路径约束（区别于最短路径）

TEB优化的问题

TEB是一个多目标优化的问题，大多数目标都是局部的，故只与一小部分参数相关。因此它只依赖于几个连续的机器人状态。

优化算法使用的时开源框架g2o : nodes、edges.

g2o优化算法目标 即这些离散的位姿组成的轨迹能够达到时间最短、路径最短、远离障碍物等目标，同时限制速度/加速度，使轨迹满足机器人运动学。

TEB整个工作流程：

全局路径——>加入约束条件——>g2o优化——>速度指令
(等时间间隔插入N个状态点，把它变成橡皮筋)
(给橡皮筋施加外力)

效果演示

详情见演示视频文件夹下的的导航展示.mp4文件

- 演示视频//导航展示.mp4
- 演示视频//导航+建图.mp4

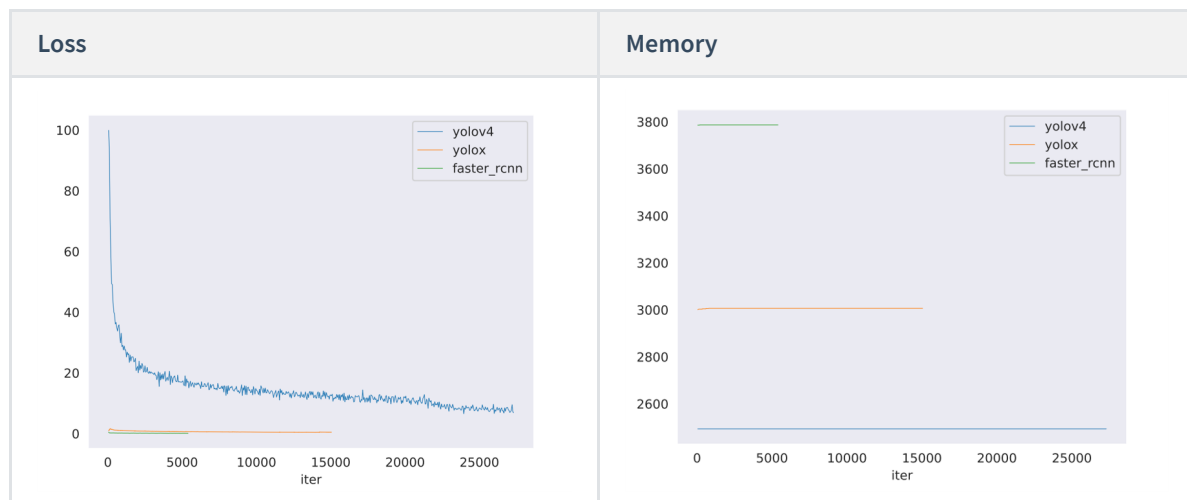
我们把导航与建图进行了结合。用户可以选择已经建图完成的区域作为导航的目标，当车辆行驶到该位置时，构建的地图会被更新，从而可以选择新的导航目标。通过不断重复这个过程实现半自动建图。

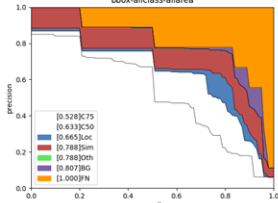
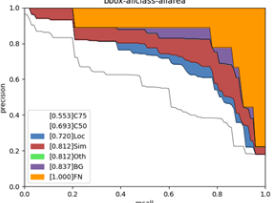
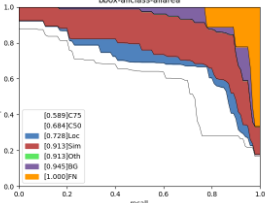
yolov4 目标检测

功能描述

可以通过yolov4算法对路径上的物体进行检测，然后再通过深度相机判断与物体的距离。最后，根据距离与检测到的物体做出相对应的反应。

原理简介



| 模型 | Faster R-CNN | YoloX | Yolov4 |
|------------------|---|--|---|
| fps(img/s) | 15.4 | 43.0 | 53.5 |
| time(ms/img) | 64.8 | 23.3 | 18.7 |
| PR曲线 |  |  |  |

我们分析比较了不同的不同的目标检测算法，最终我们认为yolov4再速度，内存以及精度等方面都更加符合我们的要求。

在此基础上，我们重新收集、标定并训练了数据集，使得模型能够识别左转、右转、停车、红灯、黄灯、绿灯等多种物体，合计3160张。

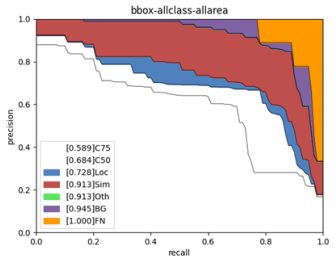
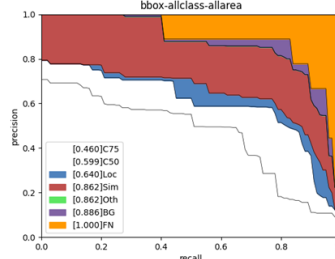
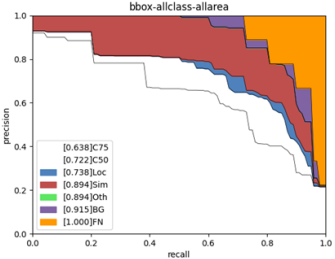
数据集具体信息如下所示：

| 数据集格式 | VOC |
|-------|------|
| 图片格式 | jpg |
| 图片数量 | 3160 |
| 训练集数量 | 2844 |
| 测试集数量 | 316 |

优化

我们的优化方向调整anchor，主要有以下两个方案

1. K-means聚类
2. DE算法

| Yolov4-common-anchor | Yolov4-kmeans-anchor | Yolov4-DE-anchor |
|---|---|---|
|  |  |  |

我们尝试采用不同的优化算法后，计算其PR曲线。最后发现Yolov4-DE-anchor的方案效果最好。因此把他作为最后的优化方案。

效果演示

详情见演示视频文件夹下的的目标检测.mp4文件

演示视频//目标检测.mp4