轮式机器人：ROS仿真导航规划

周炜 3210103790

Part 1 路径规划

路径规划是在给定的环境中寻找从起点到目标点的路径的过程，同时考虑到可能存在的任何障碍或限制(只考虑工作空间的几何约束，不考虑机器人的运动学约束)

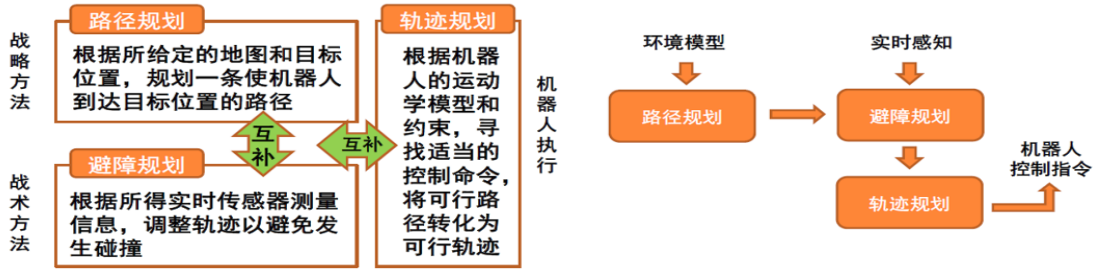


图1 路径规划、避障规划、轨迹规划三者关系

RRT算法

以起始点作为树的根节点，然后在可行空间中随机采样，找到树中离采样点最近的树节点，根据机器人执行能力生成新节点和新的路径，根据无碰检测加入树中，不断重复该过程，直到树节点生长到重点区域

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  $E_i \leftarrow Edge(x_{new}, x_{near})$;
  if $CollisionFree(\mathcal{M}, E_i)$ **then**
    $\mathcal{T}.addNode(x_{new})$;
    $\mathcal{T}.addEdge(E_i)$;
  if $x_{new} = x_{goal}$ **then**
    Success();

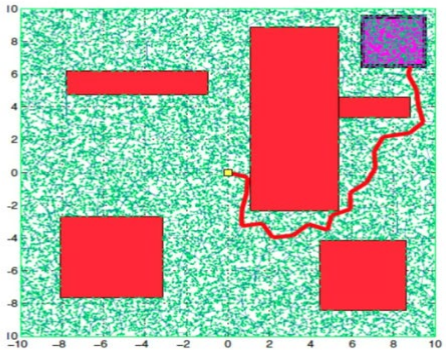


图2 RRT伪代码和效果图

AStar算法[1]

基于优先级定义的广度优先搜索，根据启发式评估函数[2]

$$f(n) = g(n) + h(n)$$

选择路径代价最小的结点作为下一步探索结点而跳转其上

实验步骤

在 Astar.py 和 RRT.py 中分别实现对应的算法，然后在助教所给的 main.py 中调用

遇到的困难

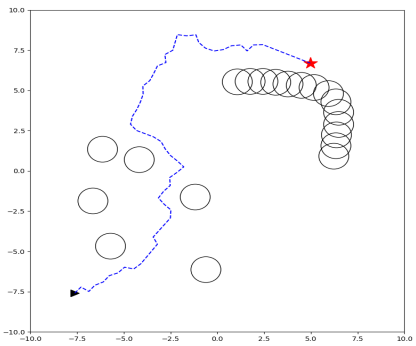
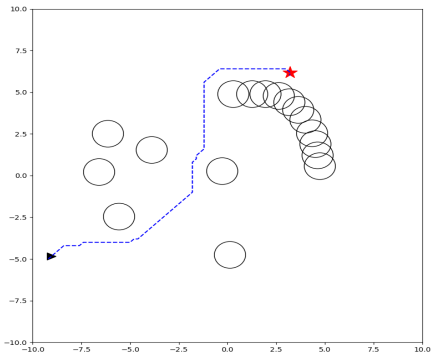
主要出现在调参上，比如膨胀设大了，小车有时候会撞到设置的障碍；小车的速度和加速度过大或者过小也都不合适

---

[1] 上课的时候没有提到 A*算法的具体实现，但是由于 A*也是常用的路径规划算法因此我也进行了实现

[2] n表示节点；$g(n)$表示从起始点到节点的实际代价，即 Dijkstra 算法中的代价；$h(n)$为从节点到目标点的最佳路径的估计代价，即节点到终点的欧式距离

RRT和Astar对比

	RRT	AStar
算法思想	随机采样	启发式算法
时间复杂度[3]	$O(n^2)$	$O(bd)$
存储消耗	较少	较多
适用情境	少量节点和已知图形	复杂环境和高维空间
Python仿真		

Part 2 避障规划

DWA 算法

DWA（Dynamic Window Algorithm），即动态窗口法，在基于速度控制运动，构建可行的速度空间，**在可行的速度空间种选择最优的速度控制指令：**

$$evaluation(v,w)=\alpha\cdot heading(v,w)+\beta\cdot dist(v,w)+\gamma\cdot velocity(v,w)$$

$$\alpha+\beta+\gamma=1\ (\alpha\geq 0,\beta\geq 0,\gamma\geq 0)$$

$\alpha\cdot heading(v,w)$ 朝向目标点：保证机器人朝目标点运动

```
def heading_cost(self, xf, yf, xt, yt, robot_inf,):
    angle = np.arctan2((yt - yf),(xt - xf)) - robot_inf[2]
    dist = np.sqrt(math.pow((xt - xf),2) + math.pow((yt - yf),2))
   return 2*np.abs(angle)+dist
```

$\beta\cdot dist(v,w)$ 远离障碍物：保证机器人避开障碍物，安全不碰撞

```
def dist_cost(self, dwa, robot_inf, plan, rad):
    dist = 100000
    for i in range(int(dwa.predict_time / dwa.dt)):
      for obs in plan:
        now_dist = np.sqrt(math.pow((obs[0] - robot_inf[0]),2) +
math.pow((obs[1] - robot_inf[1]),2)) - rad
        if now_dist < 0:
          return 100000
        else:
          dist = min(now_dist,dist)
        robot_info = self.motion(robot_info, dwa)
   return 1.0 / dist
```

[3] n 代表 RRT 树中的节点数；b 代表搜索树的分支因子；d 代表搜索树中最优解的深度

$\gamma \cdot velocity(v,w)$速度最大化：保证机器人以最大速度运动

```
def velocity_cost(self, now_v, max_v):
        return max_v - now_v
```

实验步骤

实现 dwaplanner.py，然后调用上个实验中所写的 Astar.py(我使用的)或 RRT.py

遇到的困难

主要的问题仍然是出现在调参上，具体来说是

1. 小车转角过大，修改时考虑的因素：
   障碍的半径膨胀过大
   小车角速度范围太大，速度上限太大
   小车前面预测点距离当前位置较远
2. 当小车更偏向倒车或者转圈而不是从侧面绕过障碍：
   障碍物放置位置离小车太近
   小车给的角速度范围不够大

我还认识到了python是一种脚本语言，因此当解释器无法看懂中文注释时也会出错

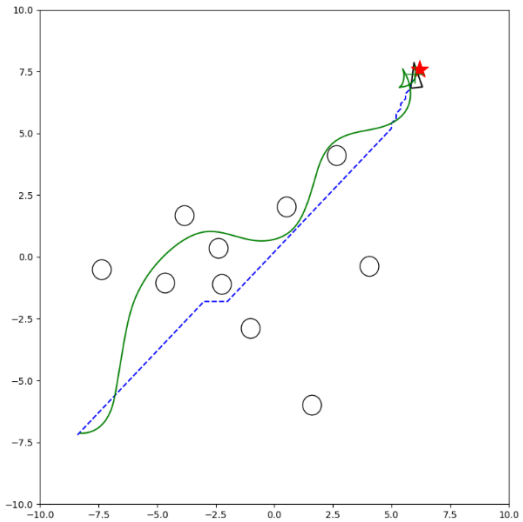


图 3 调参前的 DWA

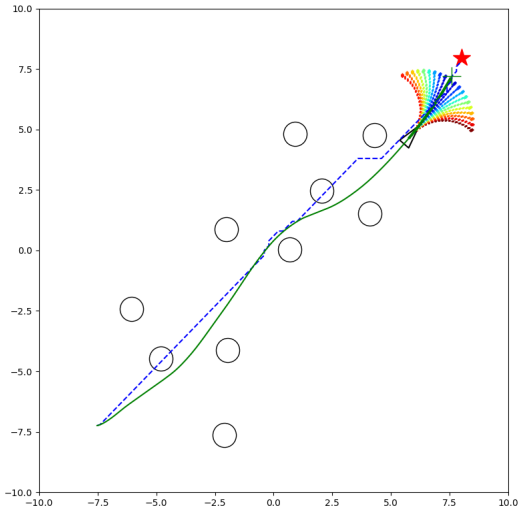


图 4 调参后并且带雷达的 DWA

Part 3 轨迹生成与仿真

根据机器人的运动学模型和约束，寻找适当的控制命令，将可行路径转化为gazebo和Rviz中的可行轨迹

本实验主要需要完成a_star.py和dwa.py这两个文件，不能直接套用实验5，6中的参数，要和小车的尺寸匹配起来，就需要许多的调参。由于之前的实验中就已经在Astar.py和RRT.py中分别实现对应的算法，a_star.py只需要一些参数的改动

文件组织

```
└─── course_agv_nav
    ├── CMakeLists.txt
    ├── config
    ├── launch
    │   ├── nav.launch
    │   └── nav.rviz
    ├── msg
    ├── package.xml
    ├── scripts
    │   ├── global_planner.py
    │   ├── a_star.py
    │   ├── local_planner.py
    │   └── dwa.py
    └── srv
```

遇到的困难

翻车

急刹车急走，动态窗口开得过大

倒车幅度过大

避障的奖励调小，障碍物膨胀半径调小一些，实操中，调小奖励函数会更为有效

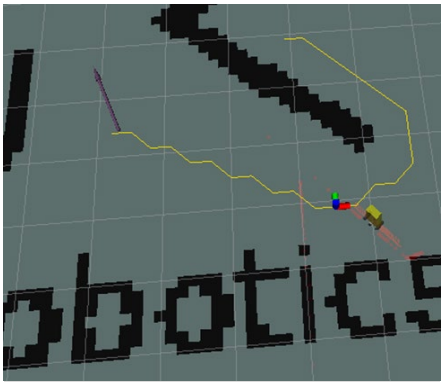


图5 翻车

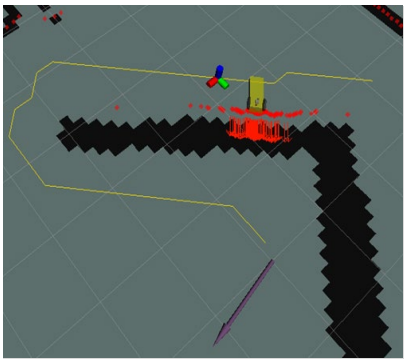


图6 转弯幅度过大

转弯幅度过大

不仅仅是角加速度不能过大，并且最大的线速度也不能过大；另一方面，障碍物的膨胀半径不能过大，否则也会导致转弯半径较大，在狭小空间内转弯不一定能回到正常轨迹上

计算速度较慢

需要对部分计算语句进行优化，对大量障碍物坐标进行运算会导致代码执行速度较慢，发生卡顿现象。可以多使用numpy库中的运算代替普通的运算符号；同时，可以考虑只对机器人视角范围内部分障碍物点进行计算，减少计算量和对内存的消耗

最终效果

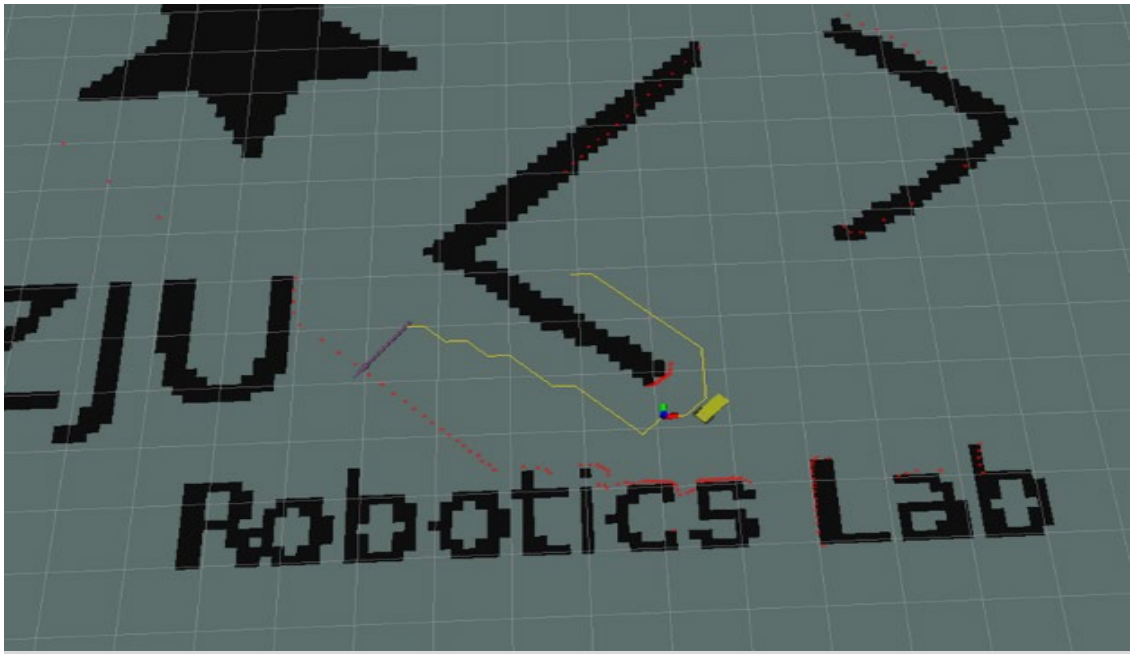


图7 Rviz下轨迹规划仿真

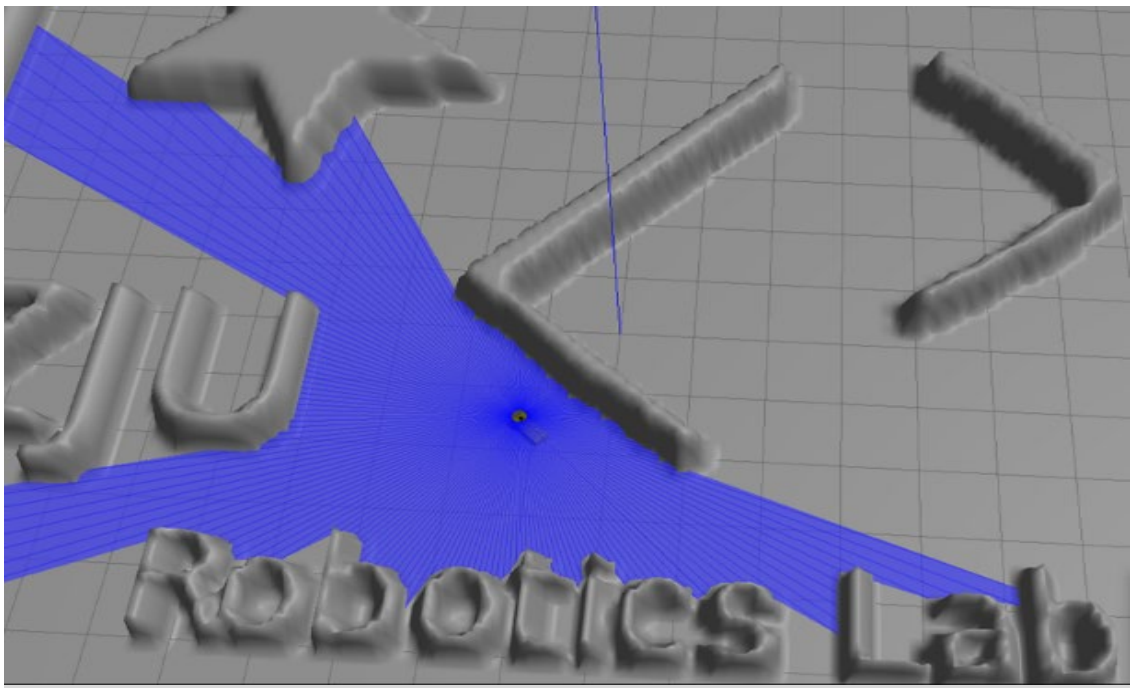


图8 Gazebo下轨迹规划仿真