专业：自动化
姓名：
学号：
日期：2022/07/15
地点：浙江大学玉泉校区

浙江大学实验报告

课程名称：机器人与智能系统综合实践 指导老师：_____ 成绩：_____
实验名称：移动机器人导航规划 实验类型：_____ 同组学生姓名：_____

装 订 线

一、实验目的和要求（必填）

实验目的：

1. 掌握 ROS 框架及相关工具的使用；
2. 掌握路径规划算法；
3. 掌握速度规划相关算法；
4. 掌握运动学求解相关算法
5. 使用 Python 等编程语言利用 ROS 实现导航规划算法

实验要求：

1. 实现 RRT 或其他算法在仿真中实现路径规划功能并在 RViz 中可视化；
2. 实现 DWA 或其他算法使机器人跟随路径；
3. 实现运动学分解算法使机器人移动；
4. 完成实验后需提交实验报告电子版 1 份，页数不超过 8 页 A4 纸，报告命名规则为：学号-姓名-移动机器人.docx；
5. 完成实验后需提交 ROS 环境的实验代码包 1 份，命名规则为：学号-姓名-移动机器人.zip

二、主要仪器设备

1. 基于 Gazebo 的移动机器人仿真环境
2. ROS 分布式通讯框架
3. Python + VScode

三、实验内容和原理（必填）

1 路径规划（RRT 和 RRT*）

1.1 原理

RRT Algorithm

Input : M, x_{init}, x_{goal}
Result : A path Γ from x_{init} to x_{goal}
$\Gamma.init()$ 以 x_{init} 为根节点，建立搜索树
for $i = 1$ *to* n *do*
  $x_{rand} \leftarrow Sample(M)$ 在可行空间中随机采样
  $x_{near} \leftarrow Near(x_{rand}, \Gamma)$ 找到树中离采样点最近的树节点
  $x_{near} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$ 根据机器人的执行能力生成新的节点和新的路径
  $E_i \leftarrow Edge(x_{new}, x_{near})$ 将最近点与新点相连接
  if $CollisionFree(x_{near})$**then** 无碰撞检测，根据成本进行树结构优化
    $\Gamma.addNode(x_{new})$ 添加节点
    $\Gamma.addEdge(E_i)$ 添加路径
  if $x_{new} = x_{goal}$ 若到达终点
    $Success()$;

RRT* Algorithm

Input : M, x_{init}, x_{goal}

Result : A path Γ from x_{init} to x_{goal}

$\Gamma.init()$ 以 x_{init} 为根节点，建立搜索树

for $i = 1$ *to* n *do*

- $x_{rand} \leftarrow Sample(M)$ 在可行空间中随机采样
- $x_{near} \leftarrow Near(x_{rand}, \Gamma)$ 找到树中离采样点最近的树节点
- $x_{near} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$ 根据机器人的执行能力生成新的节点和新的路径
- **if** $CollisionFree(x_{near})$**then** 无碰撞检测，根据成本进行树结构优化
  - $X_{near} \leftarrow NearC(\Gamma, x_{near})$ 选择多个邻节点
  - $X_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$ 评估各邻节点作为父节点下的总路径长度
  - $\Gamma.addNodEdge(x_{min}, x_{new})$ 根据总最短路径选择父节点
  - $\Gamma.rewire()$ 优化邻节点路径

RRT 算法是一个相对高效率，同时可以较好的处理带有非完整约束的路径规划问题的算法，并且在很多方面有很大的优势，但是 RRT 算法并不能保证所得出的可行路径是最优的。RRT*算法的主要特征是能快速的找出初始路径，之后随着采样点的增加，通过 rewire() 函数修改连接过程，使路径不断优化，直到找到目标点或者达到设定的最大循环次数。

1.2 程序框架（以 RRT*为例）

```
def __init__(self, ox, oy, avoid_buffer, robot_radius):
```

本函数初始化了一些基本参数，包括从 global_planner 中接收到障碍物信息 ox, oy，机器人大小和避障距离，同时自定义了场地在坐标系中的边界值、每次的距离增量等参数。

```
def plan(self, start_x, start_y, goal_x, goal_y):
```

本函数为 RRT*算法的主要函数，接收到起始点和目的地的坐标信息，输出为 path_x[] 与 path_y[]的规划路径点列。其实现过程与上文的伪代码基本一致。

```
def Sample(self):
```

本函数在空间内随机取点采样，返回采样点对象。

```
def CollisionFree(self, new_node, nearest_node, obstree):
```

本函数采用增量法判断是否碰到障碍物，从起始点不断伸长小段步长检测碰撞，直到到达目标点仍然无碰返回 True。

```
def Steer(self, rnd_node, nearest_node, goal_node, path_x, path_y, path_dict,
pathtree, obstree):
```

本函数根据机器人的执行能力生成新的节点和路径，如果新节点足够接近目标节点，则将新节点的坐标更改为目标点的坐标。

```
def NearC(self, new_node, pathtree, path_dict, path_x, path_y, k):
```

本函数找到现有路径树上距离新节点最近的 k 个近邻节点。

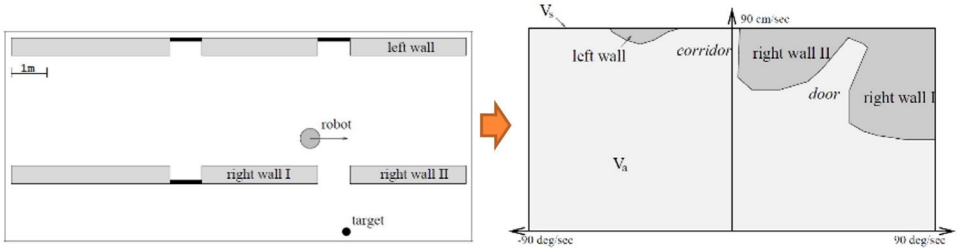
```
def rewire(self, NEAR_NODES, new_node, obstree, path_x, path_y):
```

本函数的主要作用是修改连接过程，优化邻节点路径。

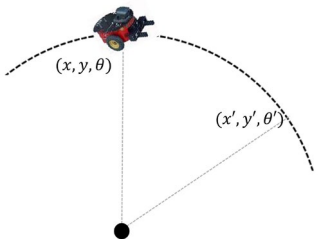
2 避障规划（DWA）

2.1 原理

动态窗格法的基本思想是在速度空间中搜索适当的平移速度和旋转速度指令(v,w)，实现从几何空间搜索转化为速度空间搜索。



首先，基于速度控制运动模型，构建可行的速度空间。



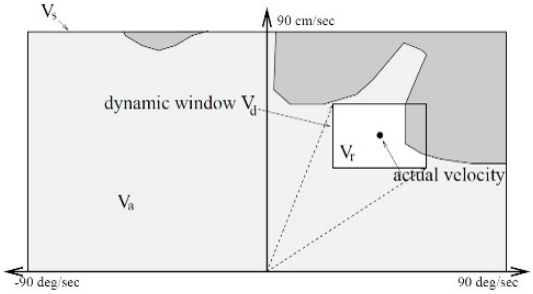
位姿更新公式为：

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x - \frac{v}{\omega}\sin(\theta) + \frac{y}{\omega}\sin(\theta + \omega\Delta t) \\ y + \frac{y}{\omega}\cos(\theta) - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \theta + \omega\Delta t \end{bmatrix}$$

不同的速度指令(v,w)会得到不同的运动半径，同样的时间间隔到达不同的终止位置。有些位置是安全的，有些会与障碍物发生碰撞。可以让机器人停止不与障碍物相碰的可行速度集合为：

$$V_a = \{(v,\omega) \mid v \le \sqrt{2 \cdot \mathrm{dist}(v,\omega) \cdot \dot{v}_b} \wedge \omega \le \sqrt{2 \cdot \mathrm{dist}(v,\omega) \cdot \dot{\omega}_b}\}$$

其中，$\mathrm{dist}(v,\omega)$表示速度配置(v,ω)所对应圆弧上最近障碍物的距离，$\dot{v}_b$为刹车平移加速度，$\dot{\omega}_b$为刹车旋转加速度。



考虑到机器人在运动过程中最大加速度的约束，在当前速度配置处以固定的小时间间隔开一个速度窗口空间：

$$V_d = \{(v,\omega) \mid v \in [v_l, v_h] \wedge \omega \in [\omega_l, \omega_h]\}$$

$$\begin{cases} v_l = v_a - a_{vmax} \times \Delta t \\ v_h = v_a + a_{vmax} \times \Delta t \\ \omega_l = \omega_a - a_{\omega max} \times \Delta t \\ \omega_l = \omega_a - a_{\omega max} \times \Delta t \end{cases}$$

结合机器人速度约束，获得可行速度空间为：

$$V_r = V_a \cap V_d \cap V_s$$

其中：

$$V_a = \{(v,\omega) \mid v \le \sqrt{2 \cdot \mathrm{dist}\,(v,\omega) \cdot \dot{v}_b} \wedge \omega \le \sqrt{2 \cdot \mathrm{dist}\,(v,\omega) \cdot \dot{\omega}_b}\}$$
$$V_d = \{(v,\omega) \mid v \in [v_l, v_h] \wedge \omega \in [\omega_l, \omega_h]\}$$
$$V_s = \{(v,\omega) \mid v \in [-v_{max}, v_{max}] \wedge \omega \in [-\omega_{max}, \omega_{max}]\}$$

在可行速度空间中选择最优的速度控制指令：

$$\mathrm{evaluation}(v,\omega) = \alpha \cdot \mathrm{heading}(v,\omega) + \beta \cdot \mathrm{dist}(v,\omega) + \gamma \cdot velocity(v,\omega)$$

其中，$\alpha + \beta + \gamma = 1 (\alpha \ge 0, \beta \ge 0, \gamma \ge 0)$；$\mathrm{heading}(v,\omega)$代表朝向目标点，保证机器人朝目标点运动；$\mathrm{dist}(v,\omega)$代表远离障碍物，保证机器人避开障碍物，安全不碰撞；$velocity(v,\omega)$代表速度最大化，保证机器人以最大速度运动。

算法的伪代码如下所示：

```
allowable_v = generateWindow(robotV, robotModel)
allowable_w  = generateWindow(robotW, robotModel)      # 首先在V_m∩V_d的范围内采样速度
for each v in allowable_v:
    for each w in allowable_w:
       dist = find_dist(v,w,laserscan,robotModel)
       if (v in v_canStop and w in w_canStop)          # 如果能够及时刹车，该对速度可接受
          CalculateEvaluation(v,w)                     # 利用评估函数对其评价
          FindBest(v,w)                                # 找到最优的速度组
```

2.2 程序框架

```
def __init__(self, config):
```

本函数利用 config 类初始化定义了速度空间中的最大速度、最小速度、最大加速度、预测步数、预测步长、机器人半径以及三个指标的系数。

```
def plan(self, x, goal, ob):
```

本函数为 DWA 算法的主要函数，在这个函数中，用两个 for 循环对速度和角速度空间进行了遍历搜索，对每一个v、ω进行碰撞检测，对每一个无碰的v、ω计算代价，并取加权平均和，最后找到一个最小代价对应的v、ω，以此作为规划结果。

```
def calc_trajectory(self, x, u, config):
```

本函数的作用是计算未来 n 步的轨迹点。

```
def motion(self, x, u, dt):
```

本函数的作用是利用运动学模型对未来 dt 时间后机器人所在位置进行计算。

```
def calc_cost(self, trajectory, goal, dist):
```

本函数的作用是根据输入的轨迹、目标点、障碍物距离集合计算代价。其中，to_go_cost 计算的是预测时间内的最后点与目标点距离，direction_cost 计算的是预测时间内最后一点朝向与机器人和目标点连线的夹角绝对值，speed_cost 计算的是速度与最大速度差值的平方，collision_cost 计算的是机器人与障碍物之间距离的倒数。将这四个代价取加权平均和，代价越小，越倾向于选择这样的速度和角速度。

```
def calc_dist(self, trajectory, goal, ob):
```

本函数的作用是计算距离机器人最近的障碍物的距离，主要在 cal_cost()函数中被调用。

3 运动控制

差分驱动机器人的行进速度、角速度和两轮轮速之间的关系为：

$$\dot{X}_I = R(\theta)^{-1}\dot{X}_R = R(\theta)^{-1}\begin{bmatrix} \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2} \\ 0 \\ \frac{r\dot{\varphi}_1}{2l} + \frac{-r\dot{\varphi}_2}{2l} \end{bmatrix} = R(\theta)^{-1}\begin{bmatrix} v \\ 0 \\ \omega \end{bmatrix}$$

通过速度和角速度解两轮轮速，得：

$$\begin{cases} \dot{\varphi}_1 = \dfrac{v + \omega l}{r} \\ \dot{\varphi}_2 = \dfrac{v - \omega l}{r} \end{cases}$$

其中，r为轮子半径，l为轮子到两轮中间中点 P 的距离（l=小车宽度 width/2+轮子半径 r/6），$\dot{\varphi}_1$、$\dot{\varphi}_2$分别为两轮旋转速度。将两轮轮速下发至驱动器即可实现运动控制。

四、实验结果与分析（必填）

1.实验结果分析

1.1 路径规划（RRT*）

我们实现了 RRT、RRT*、RRT* with Path Smoothing 三种算法，规划结果如下图所示：

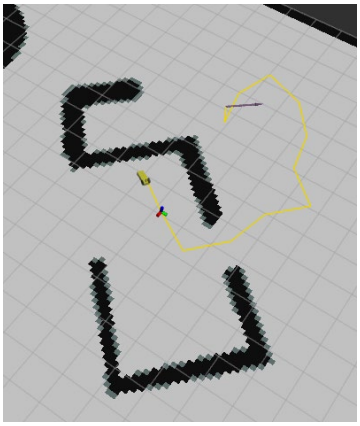


图 1：RRT 规划效果

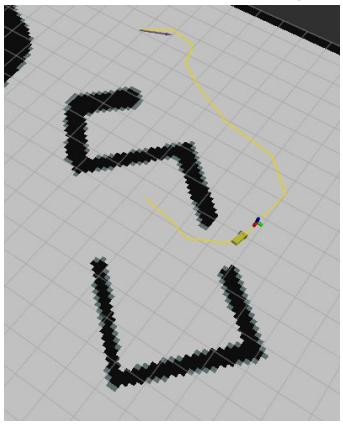


图 2：RRT*规划效果

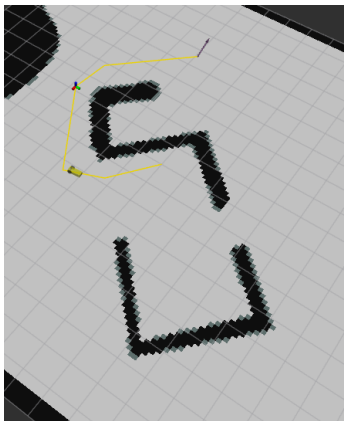


图 3：RRT* with P-S 规划效果

我们也分别对其进行了定量测试，测试结果如下表所示：

表 1：算法比较

算法	采样点数	规划用时	路径长度	路径转角绝对值和	规划失败次数
RRT	142.78	0.29732s	9.60	234.12°	0/100
RRT*	156.87	0.37678s	9.37	214.37°	0/100
RRT* with Path Smoothing	**154.57**	**0.38934s**	**7.46**	**132.76°**	**0/100**

根据规划速度以及路径质量，我们选择规划速度较快、路径质量较好的 RRT* with Path Smoothing 作为我们的路径规划算法。在此基础上，我们还探究了采样步长和偏执概率对路径规划的影响，如下表所示。

表 2：采样步长对路径规划的影响（以 RRT*为例）

采样步长	总采样点数	规划用时	路径总长度	规划失败次数
0.5	175.38	0.41569s	9.56	0/100
0.7	**156.87**	**0.37678s**	**9.37**	**0/100**
0.9	129.63	0.32689s	9.42	1/100

表 3：偏执概率对路径规划的影响（以 RRT*为例）

偏执概率	总采样点数	规划用时	路径总长度	规划失败次数
0.2	182.36	0.46342s	9.63	0/100
0.4	**156.87**	**0.37678s**	**9.37**	**0/100**
0.6	142.36	0.33876s	9.46	2/100

总之，当步长太大时，可能无法成功绕过障碍物或无法搜索到终点；当步长太小时，生长的速度减慢，收敛速度慢，且采样点数增加，运算速度减慢。经过多次尝试，最终选定采样步长为 0.7。

当偏执概率P_g较小时，随机树生长的目标性不强，产生了过多的随机采样点，导致规划速度较慢；当偏执概率P_g较大时，绕过障碍物的性能很差，且容易将目标点直接作为随机采样点。

1.2 DWA

在完成 DWA 程序及其调试之后，我们对各项参数进行了整定。首先是各损失项的加权系数。如下表所示，我们选择了几组参数进行了尝试。

表 4：各项系数对控制效果的影响

各损失项(heading、vel、obs)的加权系数	运行时间	速度标准差	角速度标准差
0.2、0.3、0.5	13.54s	0.349	0.905
0.3、0.6、0.1	**11.32s**	**0.342**	**0.743**
0.6、0.3、0.1	12.69s	0.351	0.869

最终，我们选定各损失项(heading、velocity、obstacle)的加权系数分别为 0.3、0.6、0.1，在这一参数下，小车基本能在远离障碍物的情况下以最大速度朝目标点前进。

在 DWA 预测步长及步数上，若预测步长很大，则中间有的点可能未被考虑到，有撞上障碍物的可能，若预测步长很小，则计算量很大，不利于控制的实时性。若预测步数很少，则容易找不到有效路径，会在原地转圈；若预测步数很多，则计算量大，且轨迹会很接近圆，会转着圈前进，延长行进时间。最终，我们经过多次尝试，选择预测步长 0.1s，预测步数 200 步。

同时，我们也发现，小车在接近中间点时会运动得很慢或是绕着中间点转圈，其原因在于 demo 程序对中间点打卡要求过于严格：只有当小车距离中间点很近才认为到达中间点，目标点才会更新至下一个路径中间点。于是，我们将打卡要求放宽：当小车距离中间点 0.7m 时即切换至下一个目标点，小车能够比较顺滑地通过中间点。

下图是 DWA 控制下的行进速度、角速度曲线图，可以发现整体上速度均匀，加速度不会突变，比较平滑，控制效果较好。

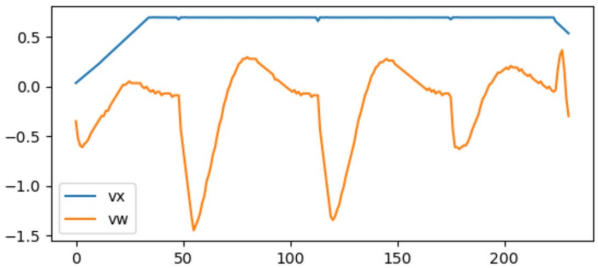


图 4：DWA 控制下的行进速度、角速度曲线图

2 算法改进

2.1 RRT*的改进

2.1.1 KDtree 加速搜索

RRT*的搜索树结构为多叉树，使用一个含有父节点和所有子节点信息的 Node 类来构建双向链表以存储所有节点，在规划时，由于涉及到大量的搜索和比较，采用遍历方法十分耗时，于是单独创建一个字典 node_dict，以及用于字典索引和生成 KD-Tree 的列表 path_x, path_y。通过这样的定义，在进行最近节点搜索的时候使用字典数据结构，在找到 x_goal 之后，通过对 x_goal 进行回溯找到路径，这样的做法大大提高了计算速度。

表 5：KDtree 加速搜索优化效果

	规划用时
无加速搜索	0.98732s
KDtree 加速搜索	**0.37678s**

2.1.2 RRT* with Path Smoothing

RRT*算法是渐进优化的，随着迭代次数的增加，路径逐渐优化，但不能得到最优路径。虽然 rewire()函数已经针对 RRT 随机小步拓展导致路径曲折、成本高的问题进行了优化，但是最后生成的路径仍有可以优化的部分，我们构造了一个 PathSmooth()函数对生成的路径进行了优化，该方法的主要思想为：取 3 个相邻路径点，如果第 1 个路径点和第 3 个路径点之间没有障碍物，则删除中间点。这样的做法带来的优化是显著的，大大减小了路径中间点的个数和路径总长度。具体数值如下表所示：

表 6：PathSmooth 优化效果

规划算法	路径中间点个数	路径总长度
RRT*	24	9.37
RRT* with Path Smoothing	**4**	**7.96**

2.2 DWA 的改进

2.2.1 双 heading 项指标

一开始，我们 DWA 的 heading 项指标采用的是预测时间内的最后点与目标点距离，这样做虽然可以保证小车是朝着目标点前进的，但小车在前进过程中并不会走直线，而是会在规划出的路径两侧左右摇摆前进。于是，为了让小车能够尽量朝向目标点直线前进，我们增加一项 heading 项指标：预测时间内最后一点朝向与机器人和目标点连线的夹角绝对值。增加这项指标后，小车前进时的摆动情况大大减小，机器人的朝向也更接近朝向目标点的角度，减少了机器人的一些不必要的转向操作，很大程度上节约了时间。为了定量评估小车前进时的摆动性，我们计算两个路径中间点间小车行进的角速度标准差以及角速度在±0.15 间的时长占总时长的比例，可见新指标的加入带来的改进是很大的。具体数值如下表所示：

表 7：双 heading 项指标优化效果

指标	路径中间点间小车角速度标准差	角速度在±0.15 间的时长占比
单 heading 项指标	0.121	63.8%
双 heading 项指标	**0.057**	**79.3%**

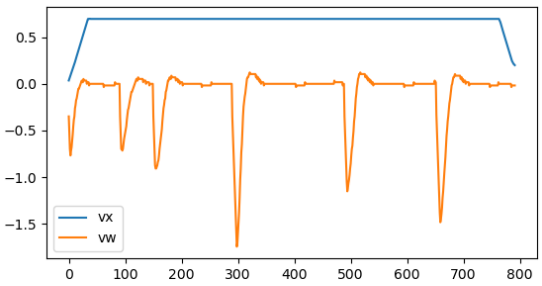


图 5：双 heading 项指标下的行进速度、角速度曲线图

2.2.2 velocity 项指标分段处理

我们的 DWA 在小车过弯时很容易速度过快，导致小车偏出路径太多，即便是调小 velocity 项系数也没有明显改善。为此，我们对 velocity 进行了分段处理，在路径中间段，小车尽量以最大速度行进，在接近目标点时，将 velocity 项中的期望速度减小，同时调大 velocity 项的加权系数，以减小小车的过弯速度。具体数值如下表所示：

表 8：velocity 项指标分段处理优化效果

	最小过弯速度	过弯时相对路径的最大偏移距离
velocity 项指标无分段处理	0.68	0.476
velocity 项指标有分段处理	**0.21**	**0.192**

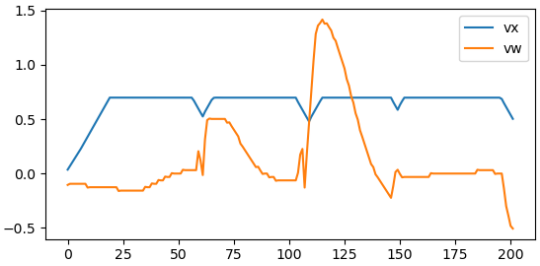


图 6：velocity 项指标分段处理下的行进速度、角速度曲线图

2.2.3 障碍物信息读取

由于本次作业的 demo 代码中删去了激光雷达相关代码，DWA 无法从 local planner 中获取障碍物信息。于是，我们仿照 global planner 的方式从 map_server 中读入障碍物信息，使用转换矩阵将障碍物坐标由世界坐标系转换到机器人坐标系，其中，转换矩阵为：

$$T = \begin{bmatrix} cos(yaw) & -sin(yaw) & x \\ sin(yaw) & cos(yaw) & y \\ 0 & 0 & 1 \end{bmatrix}^{-1}$$

随后，对障碍物进行筛选，将机器人所在点附近的障碍物坐标送入 DWA（距离机器人较远的障碍物不会对 DWA 产生影响）。同时，使用反比例函数（$cost = \frac{1}{dist - radius - 0.2}$）计算代价以保证无碰（当机器人距离最近的障碍物的距离 dist 等于机器人外接圆半径 radius 加余量 0.2 时，代价为无穷大，程序不会选择）。然而，尽管这样的做法在理论上是可行的，但由于我们读入的是全局的障碍物信息，后续还要做筛选和遍历，计算量较大，在虚拟机中跑的较慢，控制周期较长，影响控制效果，所以最终并未采用。于是，为了保证小车在行进的过程中无碰，我们的解决方案是调大 heading 项系数，让机器人尽量沿着规划路径走，这样基本能保证机器人无碰。