机器人导论作业

使用 PRM 算法进行路径规划

中山大学计算机学院　计算机科学与技术

19335174　施天予

目录

一、实验目标

绿色方块代表起始位置，红色方块代表目标位置，要求在已知地图全局信息的情况下，规划一条尽可能短的轨迹，控制机器人从绿色走到红色。

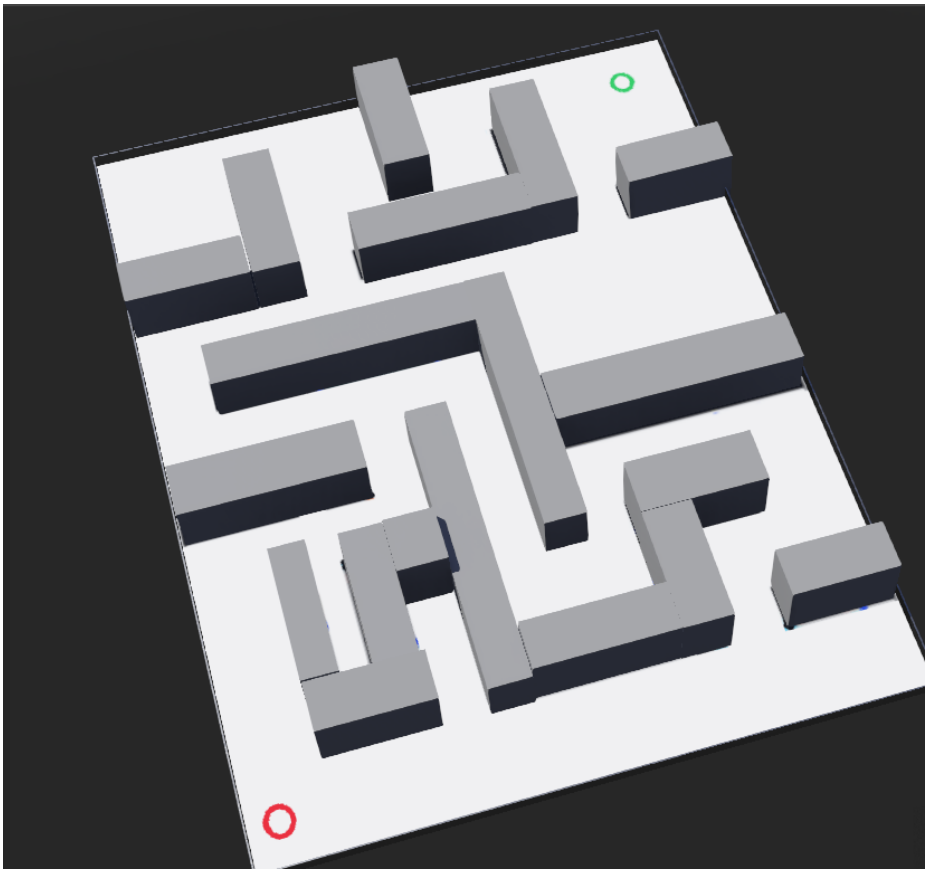

图 1: 迷宫地图

二、实验原理

路径规划作为机器人完成各种任务的基础，一直是研究的热点。研究人员提出了许多规划方法：如人工势场法、单元分解法、随机路标图（PRM）法、快速搜索树（RRT）法等。传统的人工势场、单元分解法需要对空间中的障碍物进行精确建模，当环境中的障碍物较为复杂时，将导致规划算法计算量较大。基于随机采样技术的 PRM 法可以有效解决高维空间和复杂约束中的路径规划问题。

PRM 是一种基于图搜索的方法，它将连续空间转换成离散空间，再利用 A* 等搜索算法在路线图上寻找路径，以提高搜索效率。这种方法能用相对少的随机采样点来找到一个解，对多数问题而言，相对少的样本足以覆盖大部分可行的空间，并且找到路径的概率为 1（随着采样数增加，P（找到一条路径）指数的趋向于 1）。显然，当采样点太少，或者分布不合理时，PRM 算法是不完备的，但是随着采用点的增加，也可以达到完备。

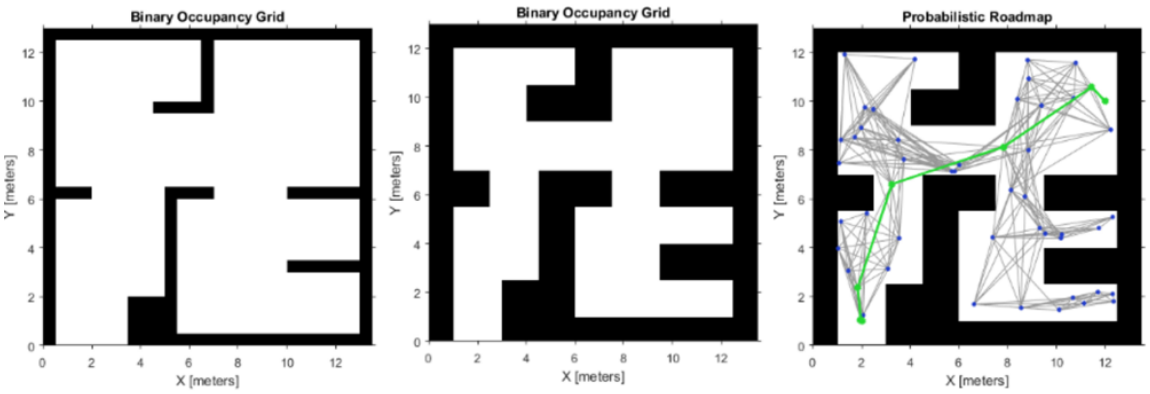

图 2: PRM 算法示图

用概率路径图法（PRM）寻找给定地图中两点之间的路径

- 学习阶段 (learning phase)：在给定图的自由空间里随机撒点（自定义个数），构建一个路径网络图。
- 查询阶段 (query phase)：搜索一条从起点到终点的可行路径。

具体考虑以下四个问题：

1. 怎样随机撒点?
    - 必须是自由空间里的随机点
    - 每个点都要确保机器人与障碍物无碰撞
2. 怎么构造区域规划器，连接两点?
    - 保证区域规划器的确定性和快速性
    - 离散化局部路径，进行防撞检查
3. 通过什么规则来选取邻域点?
    - 领域点的距离在一定范围
    - 领域点的个数有上限
4. 如何选择距离函数 D?

$$D(c,n) = \max_{x \in \text{robot}} \|x(n) - x(c)\|$$

学习阶段构建无向路径网络图的伪代码如下：

Algorithm 6 Roadmap Construction Algorithm

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

```
1:  V ← ∅
2:  E ← ∅
3:  while |V| < n do
4:     repeat
5:        q ← a random configuration in Q
6:     until q is collision-free
7:     V ← V ∪ {q}
8:  end while
9:  for all q ∈ V do
10:    N_q ← the k closest neighbors of q chosen from V according to dist
11:    for all q' ∈ N_q do
12:       if (q, q') ∉ E and Δ(q, q') ≠ NIL then
13:          E ← E ∪ {(q, q')}
14:       end if
15:    end for
16: end for
```

图 3: PRM 算法学习阶段伪代码

PRM 查询阶段：学习阶段已经构造了无向路径网络图 R=（N,E），进入查询阶段时需根据设定的起点 s 和终点 g，选择合适的路径：

1. 将 s 和 g 点与路径网络中的两个点 x,y 分别连接；
2. 寻找无向路径网络图中 x 与 y 连接的最短路径，这样就可以将起点和终点连接起来，构成全局路径；
3. 得到全局路径后，可使用平滑的方法优化路径。

主要难点在于寻找 s 到 x 的路径，g 到 y 点的路径及 x 到 y 点的路径，可以采用启发式搜索算法：A* 算法、D* 算法等，也可以采用一般的搜索算法：dijkstra 算法、广度优先/深度优先算法等。

三、核心代码

在具体实现上，我使用了 Python 编程语言来完成 PRM 算法，绘制一条从起点到终点的最短路径。其中对于图像处理我使用了 PIL 的库函数，可对输入的迷宫图转为灰度图并且绘制出随机撒点、最短路径等；对于无向图的构建我用了 networkx 的库函数，用于构建和操作复杂的图结构，提供分析图的算法。

因为 PRM 算法的整体代码过长，这里就展示几个关键函数，并加以文字辅助说明。首先说明几个要点：

- 因为迷宫 maze.png 是 600*800（长 * 宽）像素的图片，将其变为 800*600（先列后行）的二维数组后，经过不断测试发现起点位置约为（50，528），终点位置约为（730，30）。
- 随机采样点最终选取个数为 1000，邻接距离是 200，在后面部分会有关于这两个参数的讨论。
- 距离度量采用欧氏距离。

首先使用 PIL 库中的 Image 类将读入的图像转为灰度图，然后用其构建二维数组：黑色的障碍部分为‘#’，其余白色部分为‘.’。

```
    img = Image.open(img_file)
    img_gray = img.convert('L') # 地图灰度化
    img_arr = np.array(img_gray)
    img_binary = np.where(img_arr<127,0,255)
    for x in range(img_binary.shape[0]):
        temp_row = []
        for y in range(img_binary.shape[1]):
            status = '#' if img_binary[x,y]==0 else '.'
            temp_row.append(status)
        test_map.append(temp_row)
```

PRM 算法学习阶段：先在图上随机撒点，如果其位置不在障碍物上，就将其加入无向图的点集。接下来对点集中的所有点两两匹配，如果两点的连线不经过障碍物且小于邻接距离，那么就把这条边也加入无向图中。这样，无向路径网络图构建完成，学习阶段就结束了。

```
    def learn(self):
        '''
        num_sample: 随机采样的点个数
        distance_neighbor: 邻接的距离
        is_valid_xy: 判断点是否在600*800的图像范围内
        not_obstacle: 判断点是否不在障碍物上
        EuclidenDistance: 计算两点间的欧式距离
        check_path: 判断两点间的路径是否不通过障碍物
        '''
        while len(self.G.nodes)<self.num_sample:
```

```
            # 随机取点
            XY = (np.random.randint(0, self.rows),np.random.randint(0, self.cols))
            # 不是障碍物点，则加入无向图
            if self.is_valid_xy(XY[0],XY[1]) and self.not_obstacle(XY[0],XY[1]):
                self.G.add_node(XY)
        # 邻域范围内进行碰撞检测，加边
        for node1 in self.G.nodes:
            for node2 in self.G.nodes:
                if node1==node2:
                    continue
                dis = self.EuclidenDistance(node1,node2)
                if dis<self.distance_neighbor and self.check_path(node1,node2):
                    self.G.add_edge(node1,node2,weight=dis)
```

PRM 算法查询阶段：先将起点和终点加入无向图的点集中，再把它们与原点集的边按学习阶段的方式判断是否加入无向图的边集。查询阶段需要寻找最短路径，通常用 A* 搜索、D* 搜索等启发式搜索算法提高查询效率。我这里调用了 networkx 中的 shortest_path 函数来求最短路径，查看源码发现这个函数包含了 dijkstra 算法、广度优先/深度优先算法等多种算法思想。

```
    def find_path(self,startXY=None,endXY=None):
        '''
        EuclidenDistance：计算两点间的欧式距离
        check_path：判断两点间的路径是否不通过障碍物
        construct_path：将最短路径中每两点中的所有点都加入路径，用于绘图
        '''
        temp_G = copy.deepcopy(self.G)
        startXY = tuple(startXY)
        endXY = tuple(endXY)
        # 查询阶段将起点和终点添加进图中
        temp_G.add_node(startXY)
        temp_G.add_node(endXY)
        # 将起点和终点连接到图中
        for node1 in [startXY, endXY]:
            for node2 in temp_G.nodes:
                dis = self.EuclidenDistance(node1,node2)
                if dis<self.distance_neighbor and self.check_path(node1,node2):
                    temp_G.add_edge(node1,node2,weight=dis)
        # 调用networkx中求最短路径的方法
        path = nx.shortest_path(temp_G, source=startXY, target=endXY)
        return self.construct_path(path)
```

PRM 算法得到的随机撒点 100 个和 1000 个的无向路径网络图、以及最终最短路径图如图4所示。

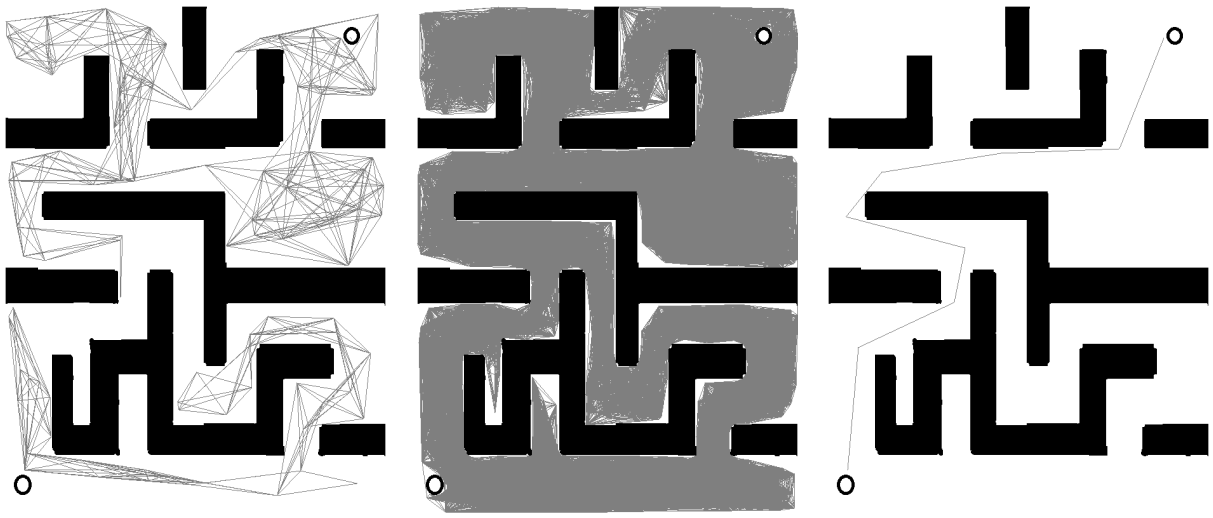

图 4: 随机撒点 100 个和 1000 个的无向路径网络图、最短路径图

四、实验过程

1. 插入路径图

通过 PRM 算法得到规划路径后，需要做的就是将规划好的路径图插入 webots 中，然后让机器人巡线完成。这里由于绘图的最短路径非常细，我就将其描粗了一些，如图5所示。

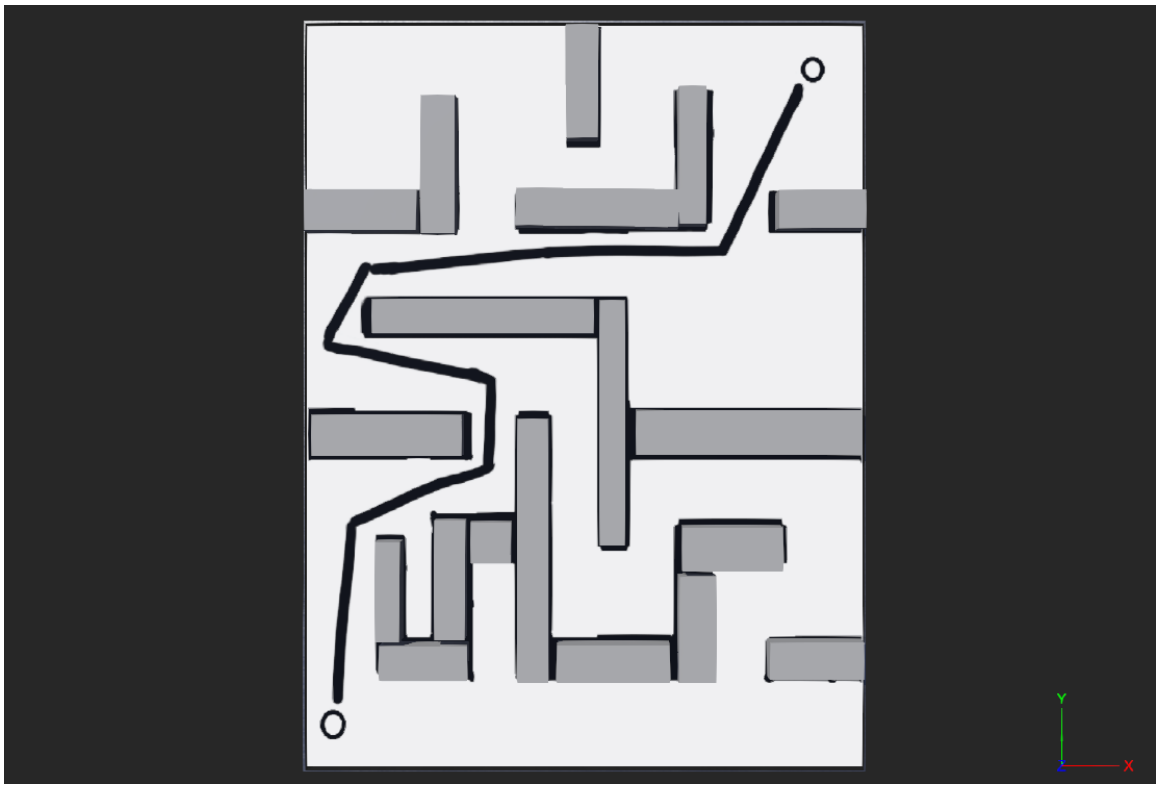

图 5: webots 中添加路径图

2. 添加机器人

接下来将上一次作业的巡线小车复制到这次的 webots 中，调整起始位置，使小车能够开始巡线，如图6所示。

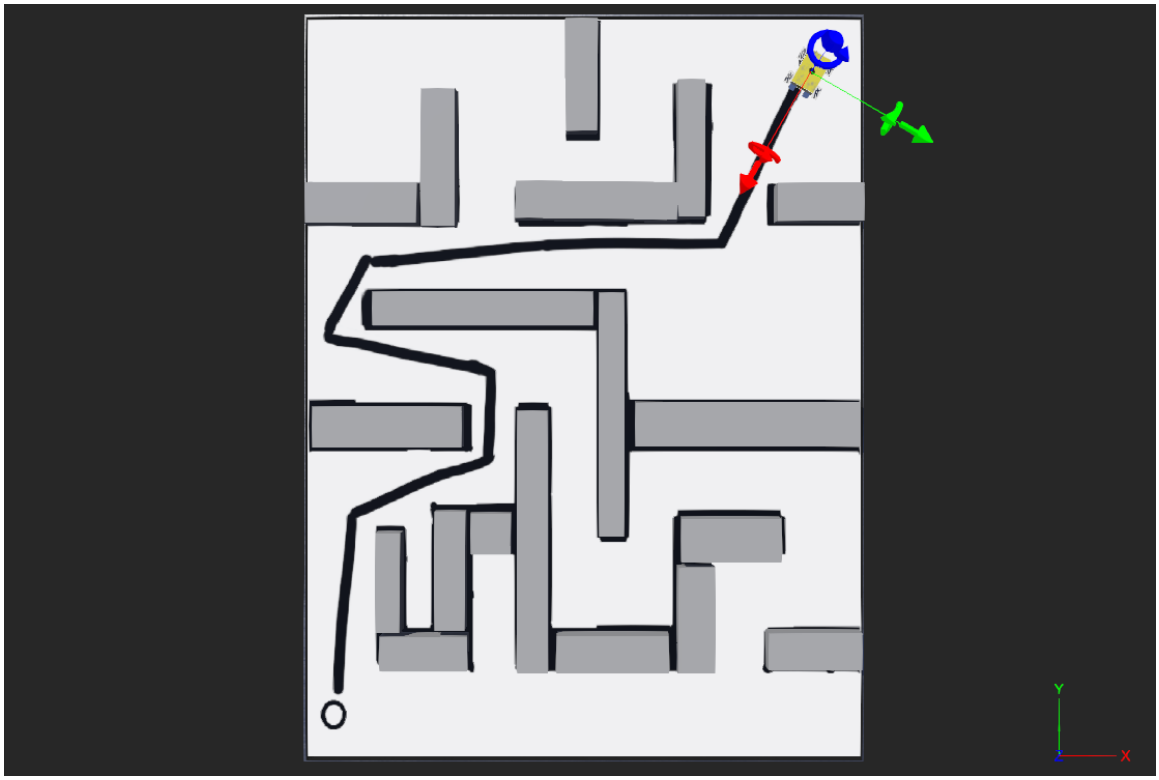

图 6: webots 中添加小车

3. 添加代码

这里我也直接以上一次作业 C++ 的代码为基础，进行了一定的修改。主要是关于几个参数的修改：为了提高机器人走迷宫的精确度，我将其直行的速度大大减小，并按比例调整转弯时机器人轮子的转速；另外由于地图的改变，我也调整了灰度阈值的参数，使机器人能够适应新地图的巡线。webots 中的 C++ 代码因为与上次作业类似就不再展示，最后修改得到的参数如下所示：

- speed1 = 0.8（直行时的轮子转速）
- speed2 = -0.3（转弯时内侧轮子的转速）
- gray = 200（灰度阈值）

五、结果分析

1. webots 机器人的运动结果

最后实现的机器人成功从起点到达了终点，完整视频在 video.mkv 中。

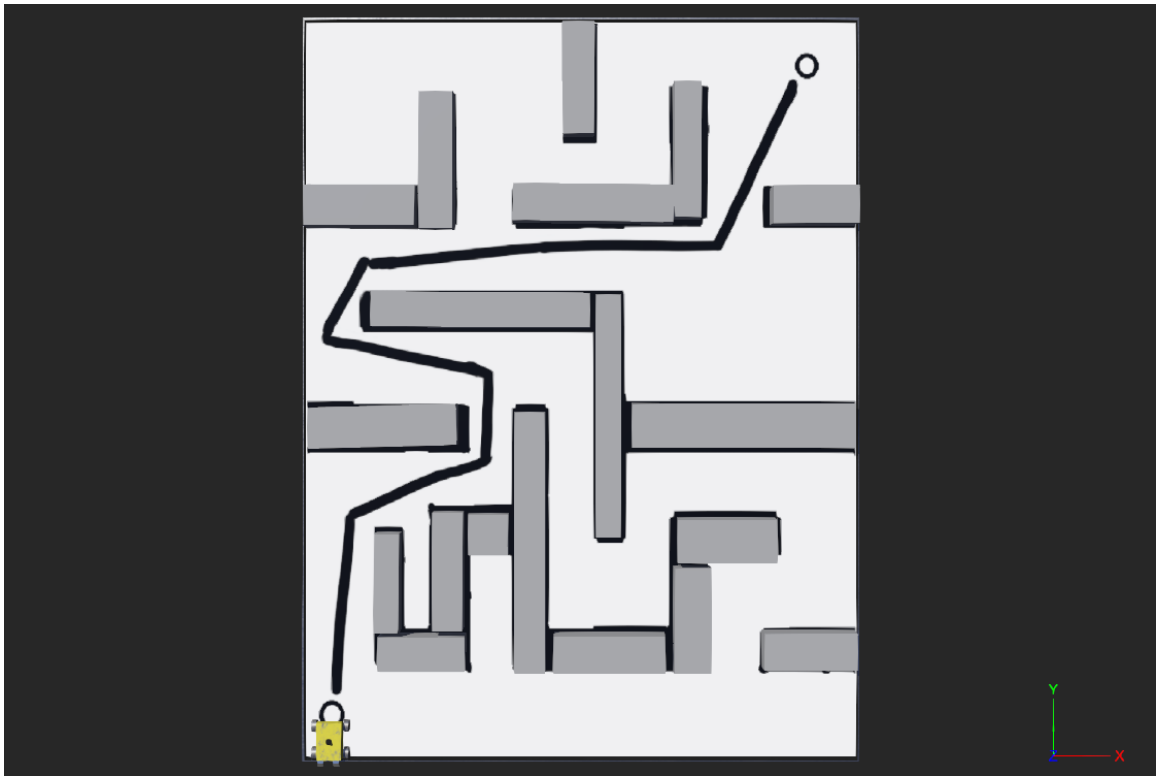

图 7: webots 中小车到达终点

2. PRM 算法的参数分析

(i) 采样数量的影响

为了探究随机采样点的数量对 PRM 算法的影响，我首先固定邻接距离为 200，分别进行采样点数量为 100、500、1000 的实验，得到结果如图8所示。

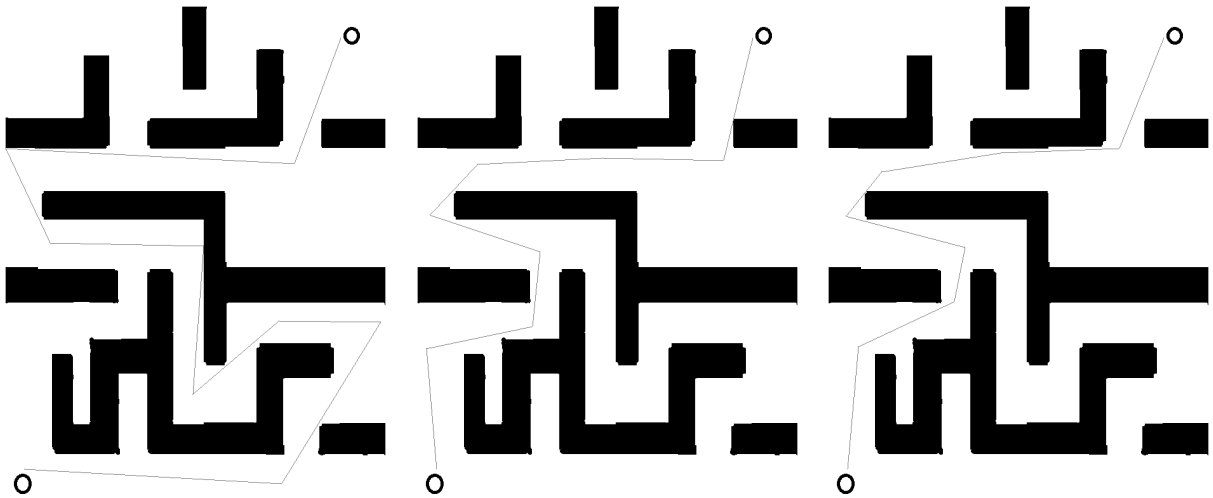

图 8: 随机采样 100 个点、500 个点、1000 个点的路径

如果只设置 100 个采样点，发现得到的路径并不是最短路径，而 500 个和 1000 个采样点得到的结果就比较合理。这也说明了抽样规划算法存在完备性弱的问题。显然对于同一地图，**采样点的数量越多，找到合理路径以及更优路径的概率就越大**。但同时，采样点数量越多，计算与搜索时间也会更长。表1是三组实验的运行时间。

随机点个数	100	500	1000
时间	2.83s	8.34s	28.56s

表 1: 不同随机采样点个数的运行时间

(ii) 邻接距离的影响

为了探究不同邻接距离对 PRM 算法的影响，我首先固定随机采样点个数为 1000，分别进行邻接距离为 50、200、1000 的实验，得到结果如图9所示。

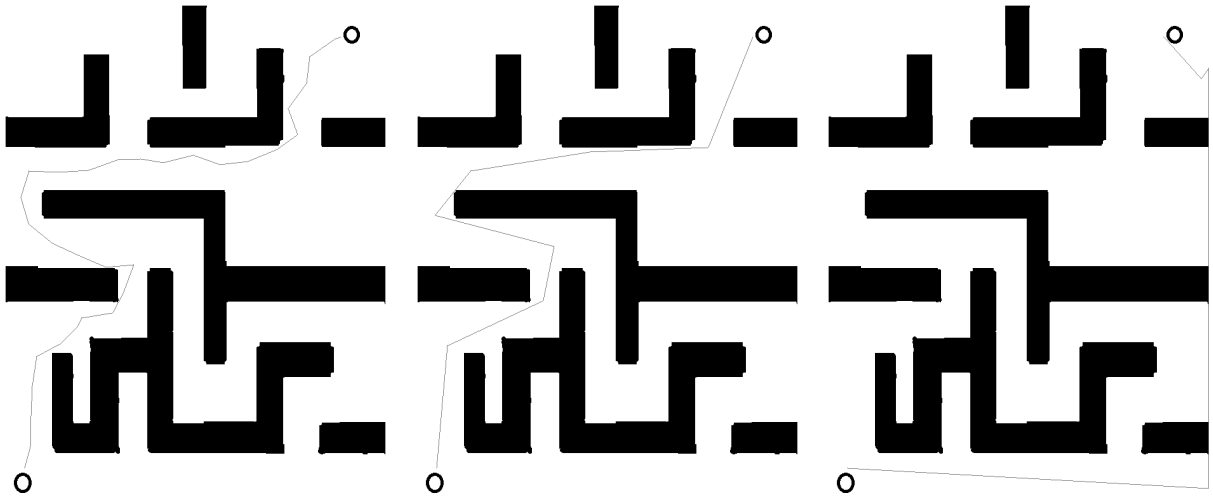

图 9: 邻接距离为 50、200、1000 的路径

邻接距离的设置影响着连线的建立与检测。当邻接距离过小时，由于连线路径太少，可能找不到解；当邻接距离太大时，会检测太多较远的点之间的连线，而增加耗时。上图中邻接距离为 50 时，最后得到的路径比较曲折蜿蜒，也就是说会增加路径的长度，不是特别合理；而当邻接距离为 1000 时，路径直接从图的边沿细缝中穿过（因为原图中的黑色障碍区域其实没有连到图像边沿），直通终点，显然也是错误的。**因此选择适中的邻接距离，才能得到比较好的结果**。

邻接距离越大，耗时越长，具体如表2所示。

邻接距离	50	200	1000
时间	6.12s	28.56s	132.77s

表 2: 不同邻接距离的运行时间

(iii) 随机性的影响

在实验过程中我还发现，即使是同样的参数，PRM 算法也可能得到完全不同的结果。

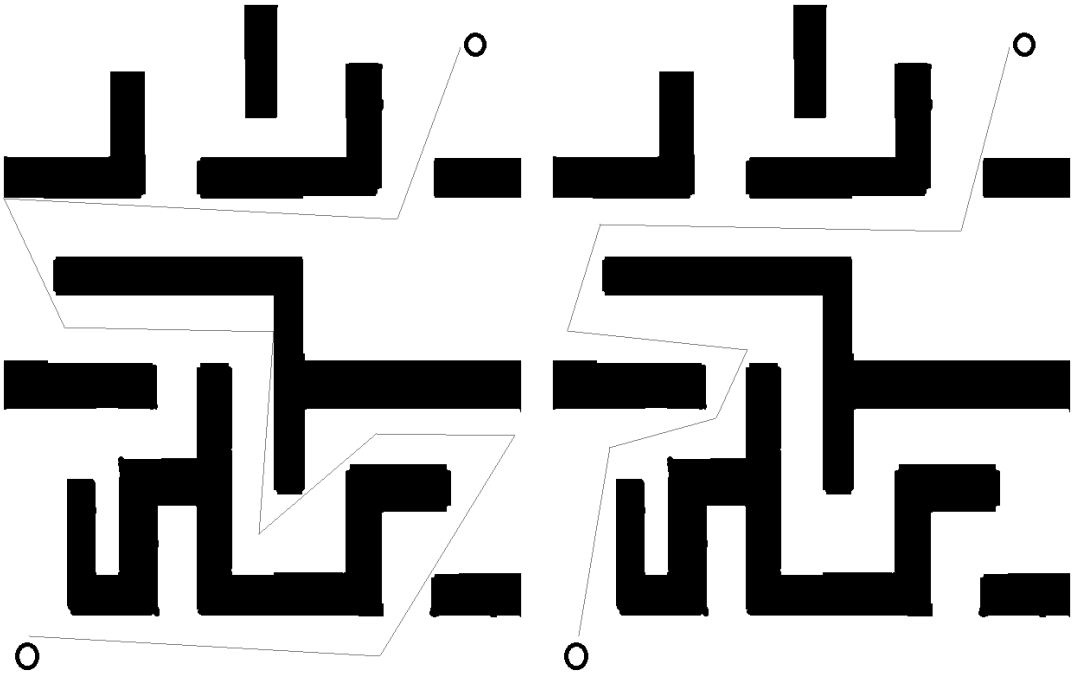

图 10: 随机撒点 100 个，邻接距离为 200 的路径

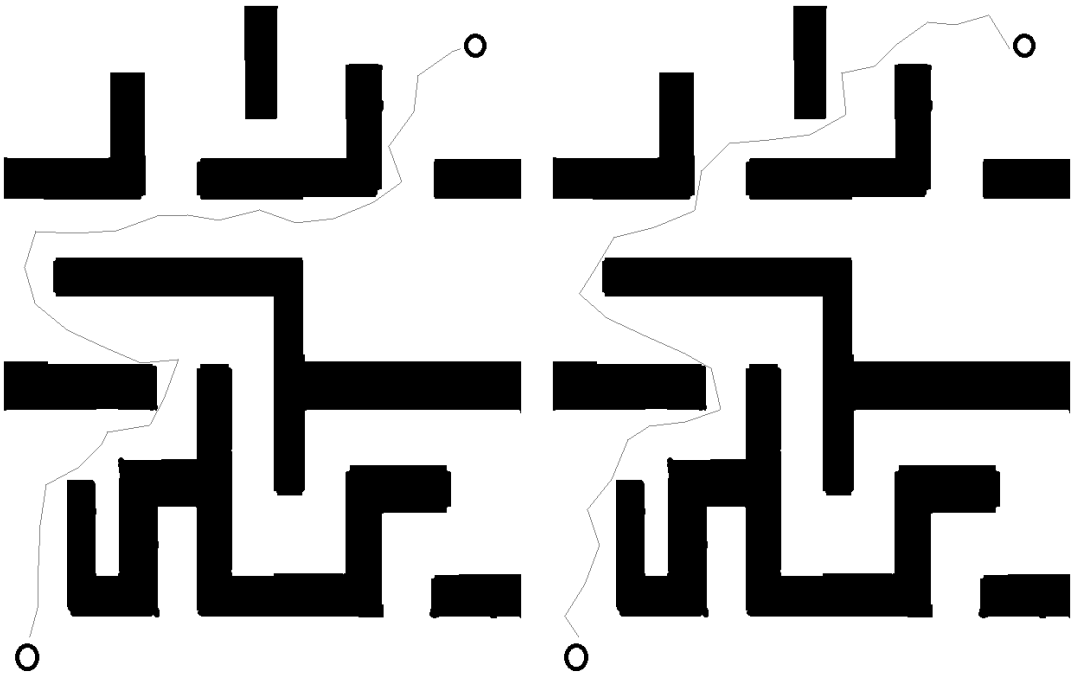

图 11: 随机撒点 1000 个，邻接距离为 50 的路径

因此我们更加需要选择合适的采样点个数和邻接距离才能提高 PRM 算法的稳定性。采样点个数不能太少，邻接距离不能太短。

六、实验总结

本次实验的难度还是相当大的，实验过程中也遇到了很多很多的问题，花费了我整整三天的时间才最终完成。

实验的核心内容就是 PRM 算法，从给定的迷宫地图中规划出一条最短的避障路径，让机器人可以巡线走完迷宫。在 PRM 算法的代码编写过程中我就遇到了很多 bug，比如黑与白的灰度值搞反、无向图的边集添加错误等等，经过反复思考和尝试，才最终完成了 PRM 算法的实现。

然而就算完成了 PRM 算法，以为可以直接让机器人巡线就行了的我还是太天真了。开始的时候我发现其实二维迷宫图 maze.png 和 webots 中的三维地图还是有一定偏差的，这让我十分头疼，具体如图12和图13所示：

图 12: 路径穿过三维地图的障碍

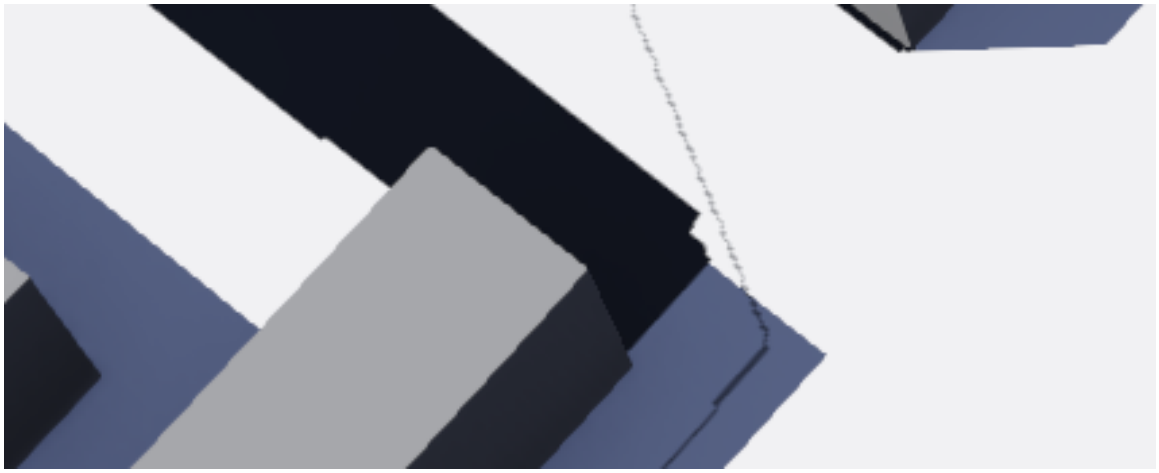

图 13: 路径未穿过二维地图的障碍

好在后来助教也修改了地图，不过也让我深刻体会到了真正工业级机器人工程师们的不易！现实中的场景往往与计划都有一定偏差，可能如果给机器人使用 **PID 算法**让机器人测量出实际与计划发生的偏差，按一定标准来进行纠正，这样效果会好一些。然而由于时间限制，我也就没有实现了。

另外，Python 实现的 PRM 算法因为是把单个像素点连起来，所以绘图的最短路径非常的细，所以我只好在 ipad 上将图描粗了一些，让机器人可以成功巡线。而且机器人其实就算巡线成功，其在运动过程中也会各种遇到问题。由于机器人本身是有一定的体积的，而这我们在用 PRM 算法时没有考虑进去，所以会导致机器人在狭小的空间中转弯轮子卡到墙角，或是直接把墙壁撞翻。于是我只好不断减小机器人的运行速度，修改转弯时两个轮子的转速比例，并且略微调整机器人开始运动时的位置，最终才得以艰难地完成了整个迷宫的巡线。不过我在做完实验后又想了想，是不是可以在使用 PRM 算法时首先**略微扩大障碍物的面积进行二维建模**，这样就把机器人的体积问题考虑进去了呢？希望之后有时间的话可以继续探索。

总而言之，这次实验收获还是比较丰富的。让我对 PRM 算法有了更深入的理解，不仅仅是停留在理论上，而是将算法应用到具体问题中去。虽然过程十分苦涩而艰辛，但同时也给我带来了极大的趣味感和成就感。