

目录

1. 非线性方程组的求解	1
1.1 修正牛顿法	1
1.2 实践一	1
1.3 实践二	3
2. A*路径规划算法	4
2.1 A*算法概述	4
2.2 A*算法实践	4
2.3 优化过程	6
2.3.1 启发式距离的选择	6
2.3.2 高斯滤波处理	7
2.3.3 方向因子	8
2.4 总结	8
3. RRT 路径规划算法	9
3.1 RRT 算法概述	9
3.2 RRT 算法实践	9
3.3 总结	10
4. 双向 RRT 路径规划算法	11
4.1 双向 RRT 算法概述	11
4.2 双向 RRT 算法实践	11
4.3 总结	12
5. 总结与展望	13
谢辞	14

1. 非线性方程组的求解

1.1 修正牛顿法

对于给定的线性方程组：

$$F(x) = [f_1(x), f_2(x), \dots, f_n(x)]^T = 0$$

我们可以采用牛顿法进行求解，牛顿法的核心思想在于迭代，通过不断的逼近来寻找方程组的解，基本公式为：

$$x^{k+1} = x^k - [F'(x^k)]^{-1} F(x^k)$$

在编程实现牛顿法时，可将上式写作：

$$\begin{cases} x^{k+1} = x^k + \Delta x^k & \text{正常迭代} \\ F'(x^k) \Delta x^k + F(x^k) = 0 & \text{每次解这个线性方程组} \end{cases}$$

通过分析发现，使用牛顿法的主要花费在于运算过程对上式线性方程组的迭代，所以我们对牛顿法作出修正。通过利用已求得的值在小循环内进行多次运算迭代，来减小大循环的迭代次数。

$$\begin{cases} x^{k,0} = x^k \\ x^{k,i} = x^{k,i-1} - [F'(x^k)]^{-1} F(x^{k,i-1}) & i = 1, \dots, m; k = 1, \dots, n \\ x^{k+1} = x^{k,m} \end{cases}$$

1.2 实践一

给定非线性方程组如下所示，利用 Matlab 作出两个方程组的图像如图 1 所示

$$\begin{cases} x_1^2 + 2x_2^2 - 1 = 0 \\ 2x_1 + x_2 - 2 = 0 \end{cases}$$

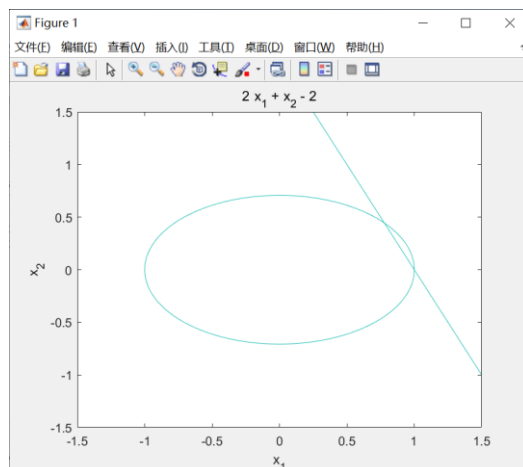
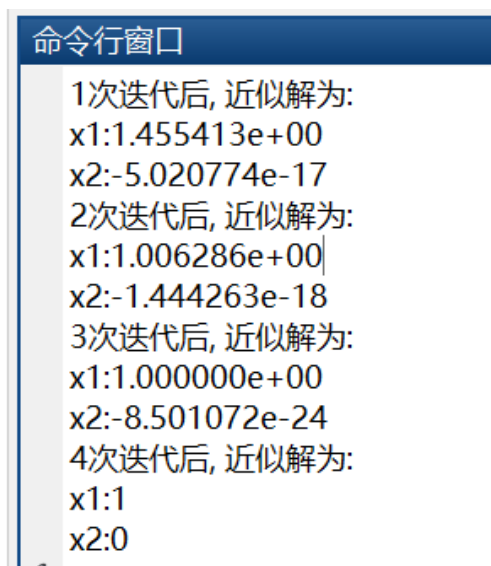


图 1 两方程交点图

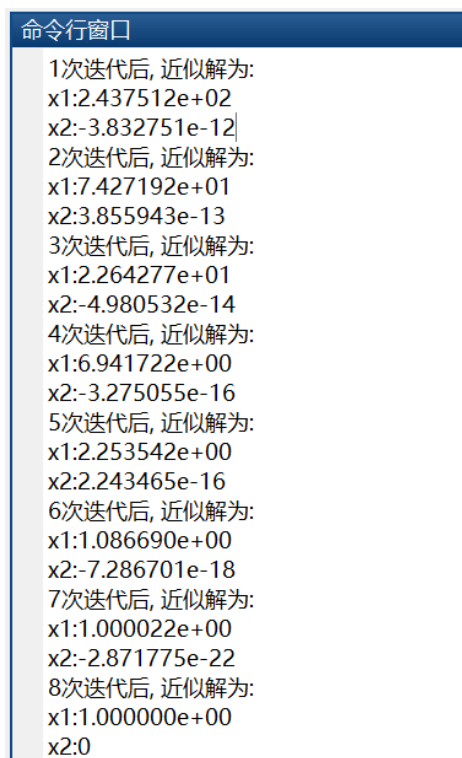
对于此方程组，我们将数据输入已编写好的 Matlab 程序中，另初值设为[4,3]，求解结果如图 1 所示，可以看到程序通过四次迭代完成运算。

若我们将修改输入初值为[800,800]，作为测试。如图 2 所示，通过八次迭代，完成运算。修正牛顿法在减小迭代次数上有所贡献。当然，在实际实验过程中，我们可以通过简要的画图确定初值的大致范围，而避免盲目给定初值增加运算量。



```
命令行窗口
1次迭代后, 近似解为:
x1:1.455413e+00
x2:-5.020774e-17
2次迭代后, 近似解为:
x1:1.006286e+00
x2:-1.444263e-18
3次迭代后, 近似解为:
x1:1.000000e+00
x2:-8.501072e-24
4次迭代后, 近似解为:
x1:1
x2:0
```

图 2 初值[4, 3]的求解



```
命令行窗口
1次迭代后, 近似解为:
x1:2.437512e+02
x2:-3.832751e-12
2次迭代后, 近似解为:
x1:7.427192e+01
x2:3.855943e-13
3次迭代后, 近似解为:
x1:2.264277e+01
x2:-4.980532e-14
4次迭代后, 近似解为:
x1:6.941722e+00
x2:-3.275055e-16
5次迭代后, 近似解为:
x1:2.253542e+00
x2:2.243465e-16
6次迭代后, 近似解为:
x1:1.086690e+00
x2:-7.286701e-18
7次迭代后, 近似解为:
x1:1.000022e+00
x2:-2.871775e-22
8次迭代后, 近似解为:
x1:1.000000e+00
x2:0
```

图 3 初值[800, 800]的求解

1.3 实践二

已知某物理量 y 与另两个物理量 t_1 和 t_2 的依赖关系为

$$y = \frac{x_1 x_3 t_1}{1 + x_1 t_1 + x_2 t_2}$$

其中 x_1 , x_2 和 x_3 是待定参数, 为了确定这三个参数所测得 t_1 , t_2 和 y 的五组数据如下:

t_1	1.000	2.000	1.000	2.000	0.100
t_2	1.000	1.000	2.000	2.000	0.000
y	0.126	0.219	0.076	0.126	0.186

对于此题, 采用与实践一类似的解法, 将五组参数带入五个方程中, 得到结果如图 4 所示:

命令行窗口

```

1次迭代后, 近似解为:
x1:5.490718e-01
x2:3.797470e+00
x3:1.839053e+00
2次迭代后, 近似解为:
x1:-7.756491e-02
x2:2.126498e+00
x3:3.866220e+00
3次迭代后, 近似解为:
x1:8.070492e-01
x2:2.019794e+00
x3:-2.388304e+03
4次迭代后, 近似解为:
x1:-8.465866e-02
x2:2.308167e+00
x3:-2.207310e+03
5次迭代后, 近似解为:
x1:7.188080e-01
x2:1.799684e+00
x3:1.854462e+06
6次迭代后, 近似解为:
x1:-9.042057e-02
x2:2.242311e+00
x3:1.809864e+06
7次迭代后, 近似解为:
x1:1.872951e-07
x2:2.000847e+00
x3:1.809864e+06
8次迭代后, 近似解为:
x1:1.872951e-07
x2:2.000847e+00
x3:1.809864e+06

f1:0.126*(1+1*x1+1*x2)-1*x1*x3=0.039128
f2:0.219*(1+2*x1+1*x2)-2*x1*x3=0.000847
f3:0.076*(1+1*x1+2*x2)-1*x1*x3=0.041150
f4:0.126*(1+2*x1+2*x2)-2*x1*x3=-0.047744
f5:0.186*(1+0.1*x1+0*x2)-0.1*x1*x3=0.152102
    
```

图 4 实践二运行结果

2. A*路径规划算法

2.1 A*算法概述

A*算法是目前应用最广的路径规划算法，最大的特点是若路径存在，使用A*算法一定能找到。常规A*算法的复杂度为 $O(n^2)$ ，在得到充分优化后，其复杂度还可进行降低，在完备算法中表现极为突出。同时，A*算法鲁棒性较高，对地图中障碍物分布要求较低，表现较为稳定。

A*算法作为启发性算法，核心控制函数为

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

其中 $f(n)$ 是节点 n 的综合优先级，当我们选择下一个要遍历的节点时，我们总会选取综合优先级最高（值最小）的节点。 $g(n)$ 是节点 n 距离起点的代价。而 $h(n)$ 是节点 n 距离终点的预计代价，这也就是A*算法的启发函数，常用的启发式距离有曼哈顿距离、欧几里得距离与对角线距离。

2.2 A*算法实践

利用Matlab编程，我对课程提供的两张地图（如图5、图6所示）进行了实践与检测。两张地图中，迷宫地图的主要特点是道路较多需要逐步的搜索，而第二张实地地图相对而言道路简单，但数据量巨大，在题目不允许降低地图分辨率的情况下，对于电脑的算力是一场挑战。

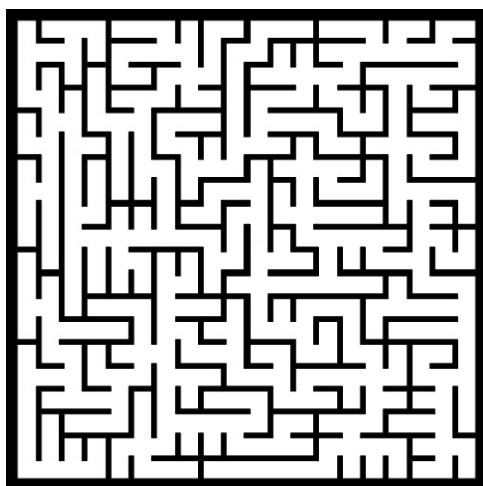


图 5 迷宫地图

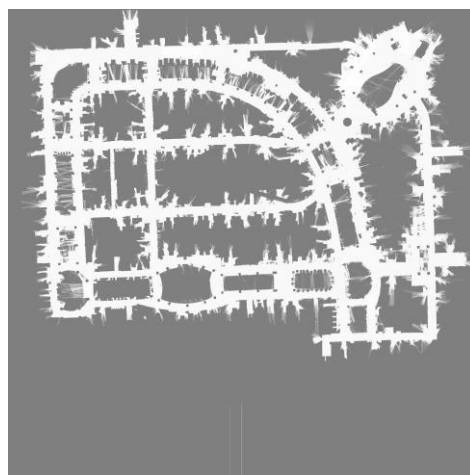


图 6 思岚实地地图

通过不断的优化，最终A*算法得到的运行结果如下图所示。对于迷宫的探索运行结果为0.39s，而对于实地地图的探索运行结果为5.97s。

同时，通过多次测试，我发现A*算法规划得出的路径，具有稳定性。不断输出规划路径长度，可以观察到A*算法规划的迷宫地图路径长度始终为1017，而规划的实地地图的长度始终为5125，有着输出的稳定性。

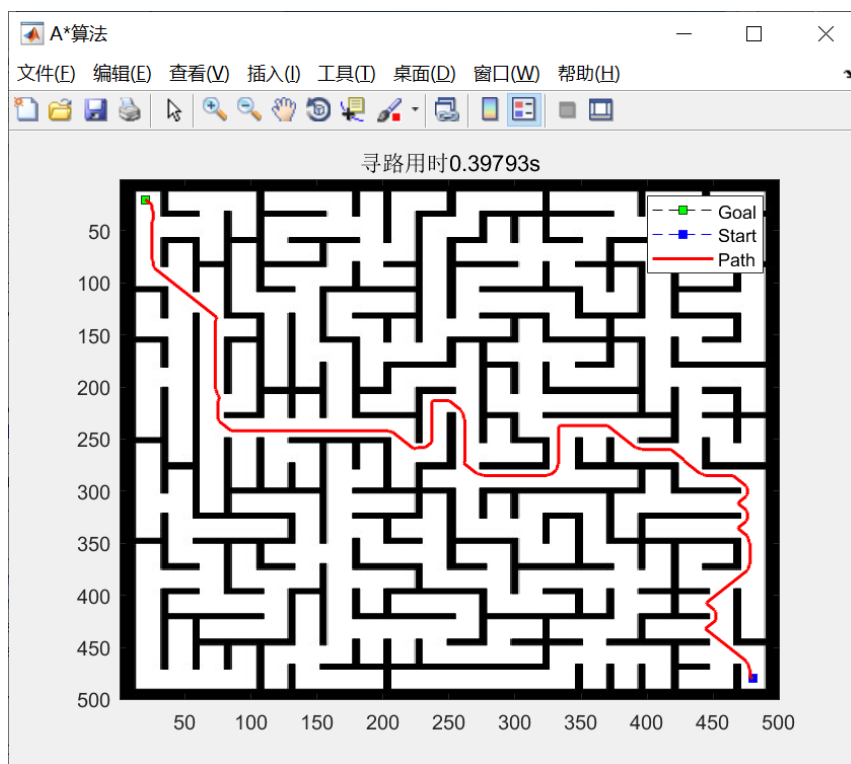


图 7 A*算法探索迷宫

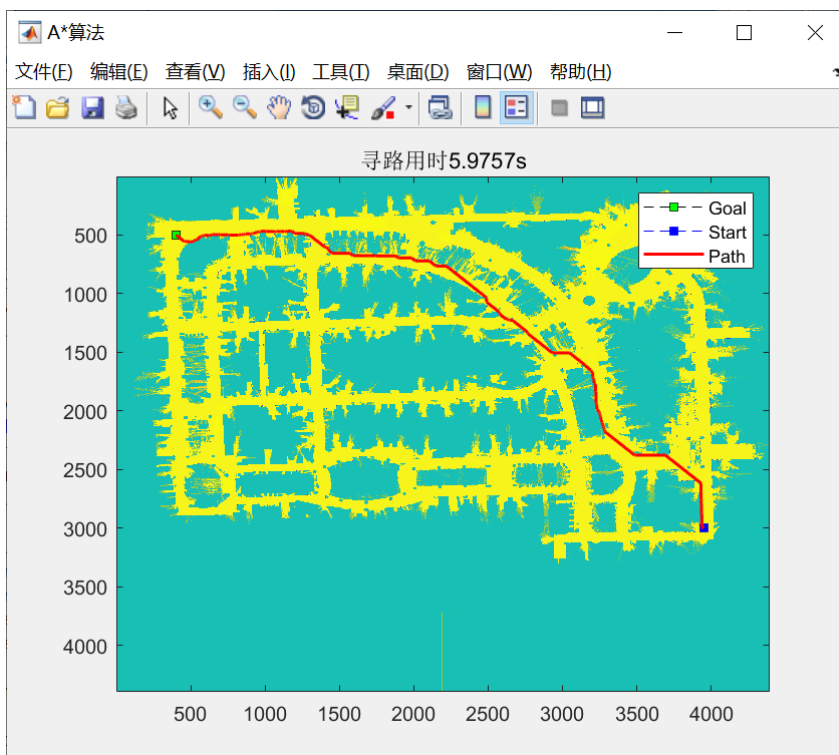


图 8 A*算法探索实地地图

2.3 优化过程

2.3.1 启发式距离的选择

在实践过程中，初始时我采用了欧几里得距离作为启发式距离，欧式距离对于迷宫地图来说运行正常，但对实地地图，运算的结果如图 9 所示。可以看到，运算的时间是巨额且不可接受的。因此，我最终选择了曼哈顿距离作为寻路的启发式距离，运行结果如图 10 所示，这一改进在时间上有了巨大的进步，相对而言较为可接受。

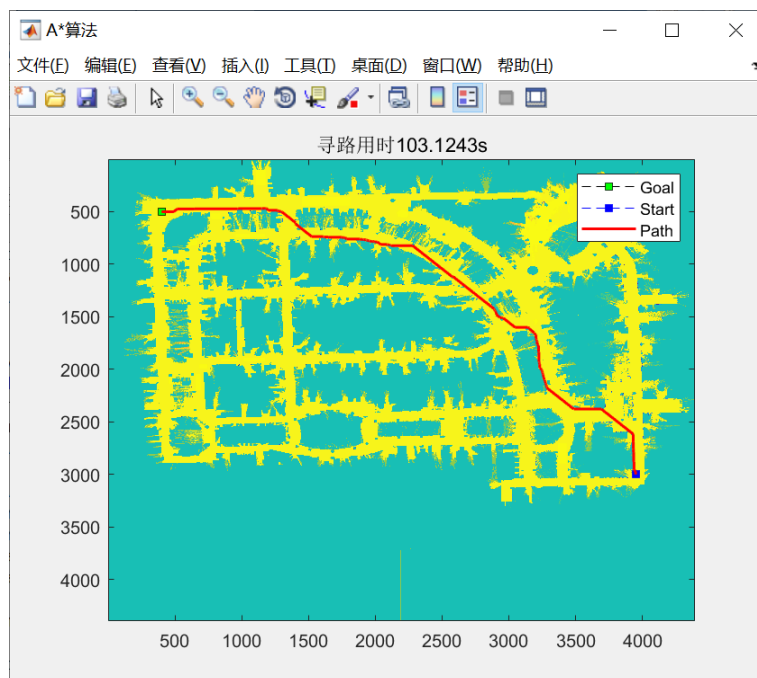


图 9 欧式距离规划

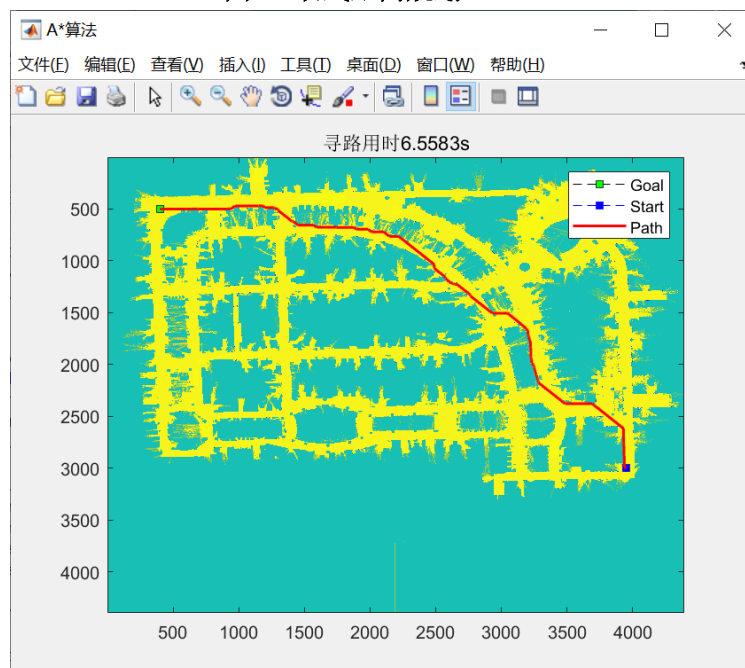


图 10 曼哈顿距离规划路径

2.3.2 高斯滤波处理

对A*算法来说，由于时间复杂度为 $O(n^2)$ ，因此，我们可以通过缩减搜索空间的规模，来提高A*算法的效率。所以，我尝试在进行路径搜索之前对地图进行高斯滤波预处理，来减小规划代价。这一处理的对于迷宫地图而言，减少了可行走的栅格数，有着一定的改进。具体改进成果如图11与图12所示。

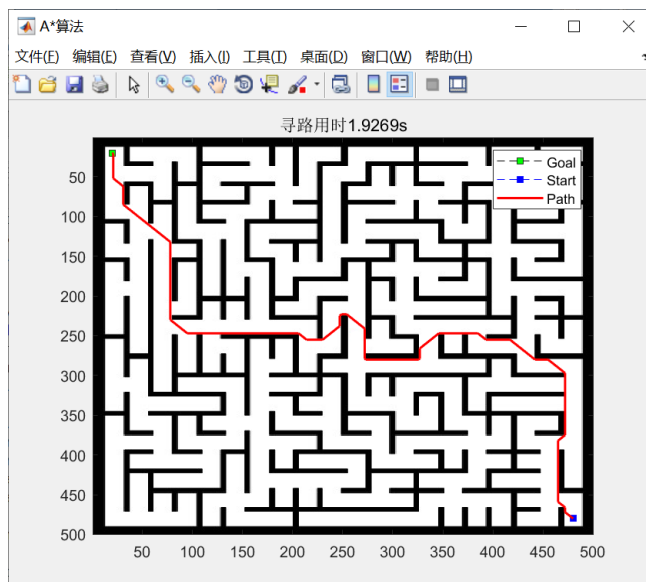


图 11 未经高斯处理

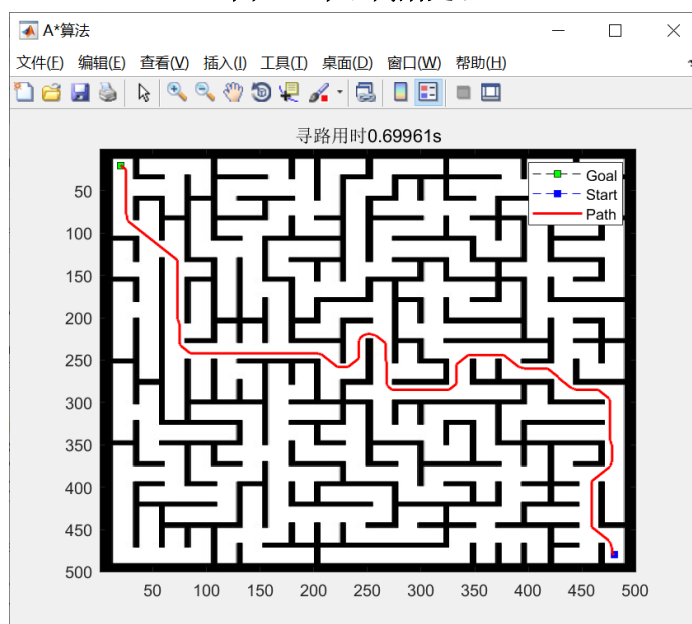


图 12 高斯处理后

可以发现通过使用高斯滤波预处理后，寻路用时有了较大的改善，同时规划出的路径与原本相比保持了与障碍物的安全距离。因此在实际机器人路径规划的解决中，也可以利用地图预处理实现安全距离。

在这里，我使用的滤波参数是默认的固定值。我认为可以通过对地图进行评估，参数化地图的区域障碍密集度，在不同的区域采用不同的滤波参数优化，以达到更好的效果。由于个人精力问题，尚未在这方面展开更多的探索。

2.3.3 方向因子

在原有的路径规划启发式算法中，仅考虑了距离这一启发因子。联想到车辆控制方面的Stanley算法，我尝试着加入了方向因子，在路径规划中加入朝着终点前进的趋势。通过测试发现，在两张地图的规划时间上都有了一定的改善。具体的测试运行结果如下图13、图14所示。

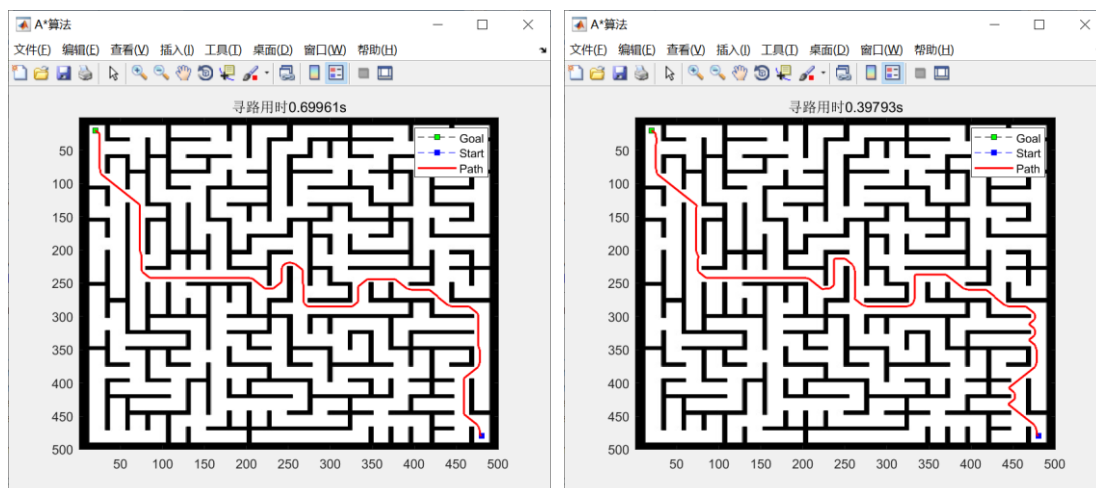


图 13 加入方向因子前后的迷宫地图

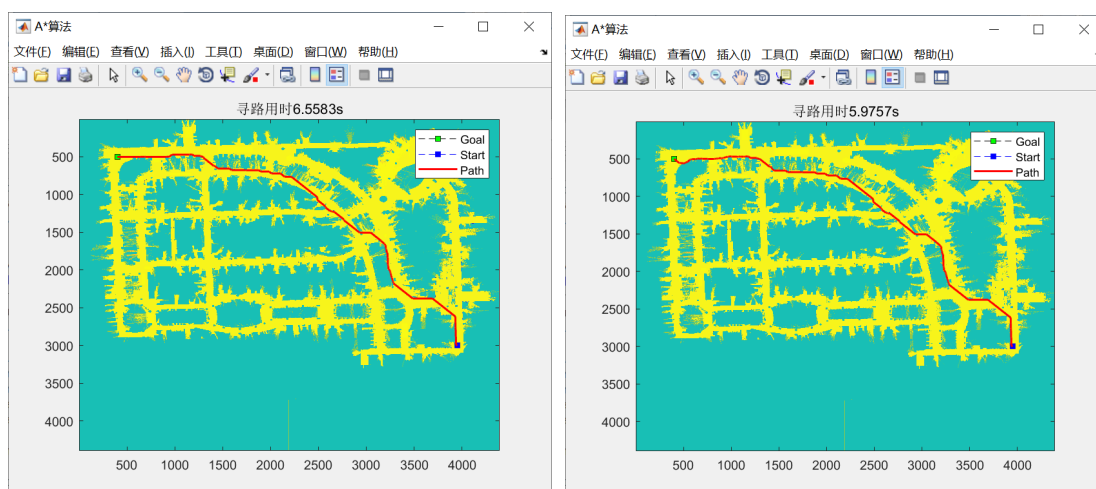


图 14 加入方向因子前后的迷宫地图

2.4 总结

A*算法作为目前应用最广的路径规划算法，有着存在路径必有解、路径最优等优点，其时间与空间复杂度在被优化后，也可以被接受。在实践过程中，由于并未对算法进行堆排序等方面的优化，我使用A*算法对大地图的运行耗费时间相对较长。同时，由于时间与精力的限制，我并未对算法的鲁棒性与普适性做出测试，后续有待改进。

3. RRT 路径规划算法

3.1 RRT 算法概述

RRT算法也是目前较为流行的一种路径规划算法，他的最大特点是利用随机生成树，执行速度极快，但成也随机败也随机，算法在一些特殊地图中可能会无法找到相应的路线，且路径可能并非最优。

RRT是一种多维空间中有效率的规划方法。它以一个初始点作为根节点，通过随机采样增加叶子节点的方式，生成一个随机扩展树，当随机树中的叶子节点包含了目标点或进入了目标区域，便可以在随机树中找到一条由从初始点到目标点的路径。

3.2 RRT 算法实践

利用Matlab编程，我对课程提供的两张地图（如图5、图6所示）进行了RRT算法的编写与实践。通过测试发现，RRT算法在迷宫这类包含大量障碍物或狭窄通道约束的地图中，算法的收敛速度慢，效率会大幅下降，甚至完全无法找到对应的路径，但在思岚实地地图中表现较好。

考虑RRT算法的不确定性，我对程序进行了十次随机测试，并进行记录，具体的测试数据如下表所示。

测试次数	耗费时间/s	路径长度
1	0.922	6823
2	0.854	5948
3	2.408	6968
4	0.915	6998
5	1.794	6423
6	0.604	6167
7	0.578	6224
8	2.034	10193
9	0.288	5835
10	0.522	7643

表格 1 RRT 算法测试表

可以十次测试中，耗费平均时间为1.09s，耗费时间总体方差为0.47，平均路径长度为6922。与A*算法相比，在时间上有了极大的进步，但规划出的路径长度与A*算法得到的5125相比，耗费较为巨大。

通过观察，我们可以发现RRT算法作为一种通用的方法，其稳定性并不良好，波动较大。在实践中，可以发现运行结果存在图15所示的耗时0.288s、路径长度5835的较优路径，也存在图16所示的耗时2.03s、路径长度10193的爆炸路径，以图16为代表的路径甚至存在折回、绕路问题。

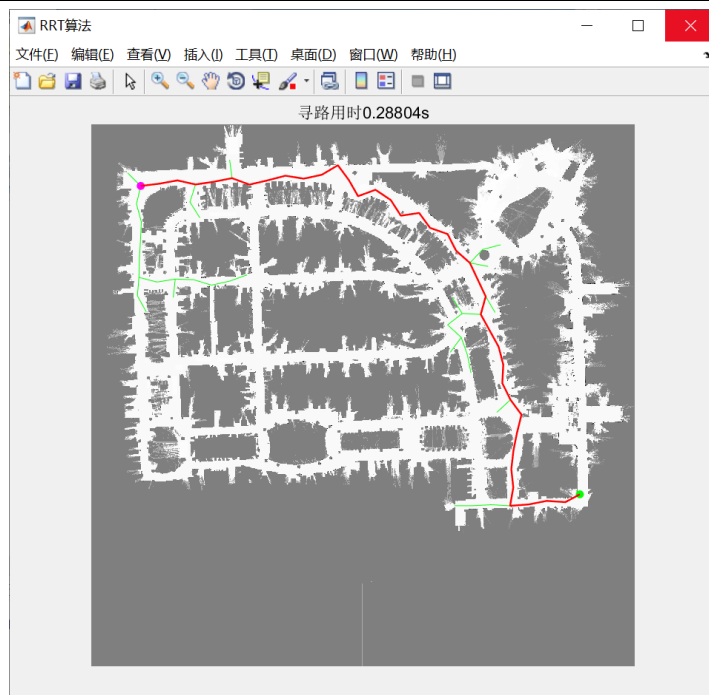


图 15 RRT 算法示例 1

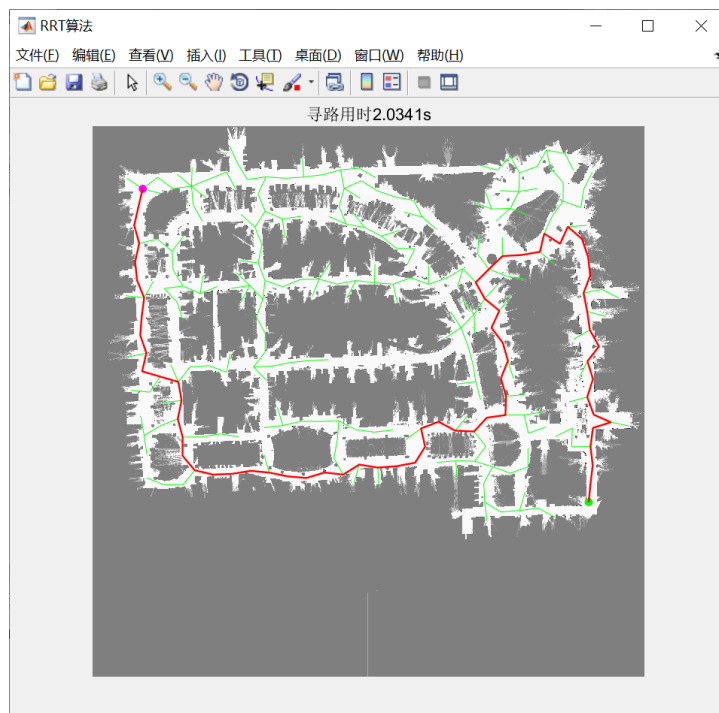


图 16 RRT 算法示例 2

3.3 总结

总体而言，RRT 算法相对 A*算法在时间上有着较大的改进，但其不稳定性造成可能存在的爆炸路径，对于实际应用有着不小的干扰。在下一部分，我将使用双向 RRT 的方法对 RRT 算法进行优化。

4. 双向 RRT 路径规划算法

4.1 双向 RRT 算法概述

基本的 RRT 每次搜索都只有从初始状态点生长的快速扩展随机树来搜索整个状态空间，如果从初始状态点和目标状态点同时生长两棵快速扩展随机树来搜索状态空间，效率会更高，这也是双向 RRT 算法的基本原理。

这种双向的 RRT 与原始 RRT 算法相比，搜索速度、搜索效率有了显著提高。首先，Connect 算法较之前的算法在扩展的步长上更长，使得树的生长更快；其次，两棵树不断朝向对方交替扩展，而不是采用随机扩展的方式，特别当起始位姿和目标位姿处于约束区域时，两棵树可以通过朝向对方快速扩展而逃离各自的约束区域。这种带有启发性的扩展使得树的扩展更加贪婪和明确，使得双树 RRT 算法较之单树 RRT 算法更加有效。

4.2 双向 RRT 算法实践

利用 Matlab 编程，我对课程提供的思岚实地地图（图6）进行了双向 RRT 算法的编写与实践。

考虑 RRT 算法的不确定性，我仍对程序进行了十次随机测试，并进行记录，具体的测试数据如下表所示。

测试次数	耗费时间/s	路径长度
1	0.242	6802
2	0.168	6950
3	0.378	6069
4	0.125	7413
5	0.195	6046
6	0.238	6529
7	0.295	6386
8	0.411	6241
9	0.074	6205
10	0.266	5906

表格 2 双向 RRT 算法测试表

可以十次测试中，耗费平均时间为 0.239s，耗费时间方差为 0.01，平均路径长度为 6454。与上一部分中 RRT 算法相比的 1.09s 与 6922 相比，在时间上有着极大的进步，且在路径长度上有所改善。

观察图17与图18的 RRT 算法，我们可以发现，双向 RRT 与普通 RRT 相比搜索路径减少了很多，在某些情况下，还可能出现图18所示的基本直达路径搜索。

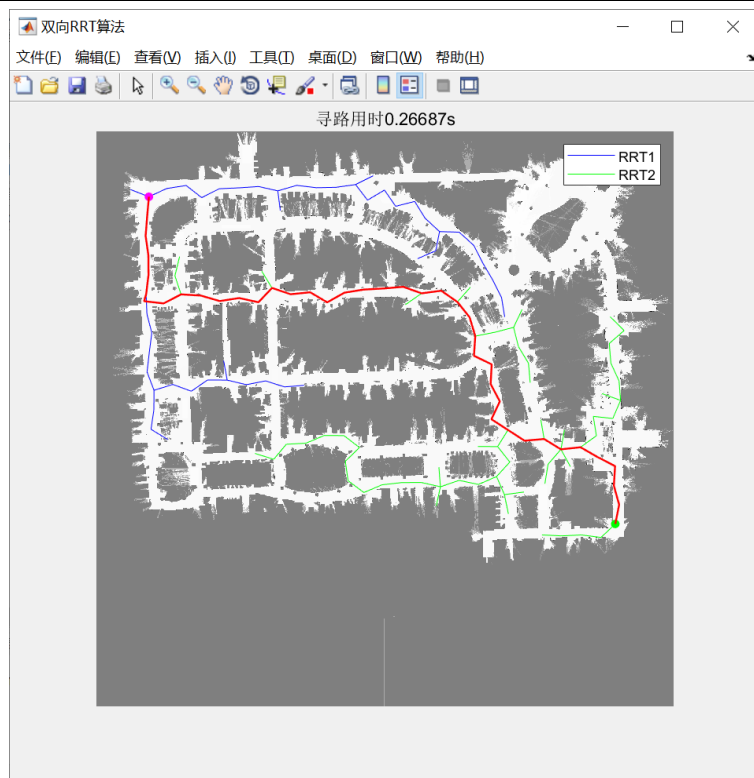


图 17 双向 RRT 算法示例 1

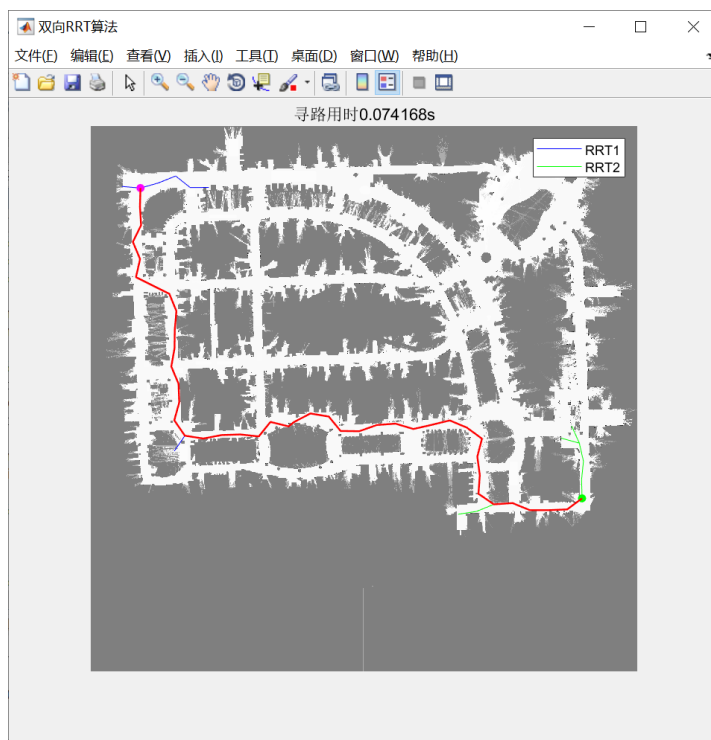


图 18 双向 RRT 算法示例 2

4.3 总结

双向RRT算法作为RRT算法的改进方法，在原有基础上运算性能及效率有了极大的提高，令算法在运行时间与搜寻路径的长度更加可接受。最关键的是双向RRT算法极大提高了普通RRT算法的稳定性，让RRT算法有了极大的应用可能。

5. 总结与展望

本文作为《移动机器人自主定位与导航算法实践》课程的实验报告，是对本门课程上课所学知识的熟悉与实践。文中主要探索牛顿法求解非线性方程与A*算法、RRT算法及双向RRT算法进行路径规划，由于并未接触过C++的MFC设计，文中算法均采用了Matlab进行了实验完成。但Matlab的效率与直接编写的C语言相比，尚有着一些不足，在本次实验中主要表现在使用A*算法对思岚实际地图的路径规划耗时过长。

在实验一的非线性方程组求解问题中，本文主要尝试采用牛顿法进行探究，并对常规牛顿法作出简要的修正来减少迭代次数。

在实验二中的路径规划题目中，本文最初尝试了A*算法，虽然算法稳定性较好且寻出路径近乎最优，但在思岚实际地图中耗时较长。相比之下，后续实现的RRT算法在耗费时间上有了极大的提升，但稳定性相应的降低。最终的双向RRT算法相对较完美地完成了对思岚地图的实践，在算法稳定性、耗费时间上都有了极大的改进。

当然，由于作者知识的局限性及临考时间精力的限制，这篇实验报告或多或少也存在着一些不足之处，对这几种算法的研究并不全面，在程序设计与编写过程中也或多或少有着一些缺陷，需要有更进一步的深入研究。

在考试周结束后，我会花费一些时间阅读一些文献了解更多的课程内容，并在路径规划算法上尝试测试蚁群算法等其余知名算法。我也会对本实验报告作出修正，并将第二版实验报告发送至庞老师的邮箱中。

谢辞

首先要感谢交大为我们提供了这一良好的平台，让我们在低年级便可以接触到实际的实践实验活动。

首先要感谢思岚科技诸位老师为本门课程的精心准备与悉心讲解，林老师、庞老师、黄老师等几位老师毫无保留、深入浅出的讲解为我们讲解了移动机器人的诸多知识，同时大量引用公司在实际的工程的示例，帮助我们拓宽视野，增长见识。作为工科学子，能够在大二接触这课程，无疑对我日后选择方向与从事实践有着巨大的好处。

其次，要感谢学生创新中心对本次课程的支持，感谢董老师与助教老师的兢兢业业，为本课程的默默付出。

此外，还要感谢课程班上的各位同学，在课堂上我们互相交流，共同进步。

最后，再次对为本课程付出过的林老师、庞老师、黄老师、董老师等老师的悉心指导，虽由于疫情原因，未曾见面有所遗憾，但真心的感谢你们的付出。