日期：2022 年 7 月 9 日

中山大學 SUN YAT-SEN UNIVERSITY

成绩：_____

学院：智能工程学院　　课程：自动驾驶技术基础　　周次：第 21 周

专业：智能科学与技术　　姓名：方桂安　　学号：20354027

1 题一

1.1 题目

学院派的运动规划主要包括：前端 ___ 规划与后端 ___ 规划。

1.2 解答

学院派的运动规划主要包括前端路径规划与后端轨迹规划。路径规划和轨迹规划组成了运动规划。

- 前端规划：（路径搜索）在地图中，搜索出一条避开障碍物的轨迹；
- 后端规划：（轨迹优化）对搜索到的轨迹进行优化，从而符合机器人的运动学和动力学约束。

2 题二

2.1 题目

请举例三种常见基于图搜索的路径规划算法。

2.2 解答

- Dijkstra
- A*
- JPS

3 题三

3.1 题目

请举例三种常见的基于采样的路径规划算法。

3.2 解答

- PRM
- RRT
- RRT*

4 题四

4.1 题目

Dijkstra 算法有什么优点和缺点?

4.2 解答

1. **优点：**具有完备性和最优性，既如果在起始点和目标点间有路径解存在，那么一定可以得到在某个评价指标上是最优的解（评价指标一般为路径的长度）；
2. **缺点：**计算复杂度高：没有全局的信息，求解的过程中，是按照均匀的向外扩散的，类似于穷举搜索，类似于 BFS。

5 题五

5.1 题目

四旋翼无人机是否满足微分平坦性质?

5.2 解答

对一些存在平坦输出的非线性系统，如果可以找到一组系统输出，使得所有状态变量和输入变量都可以由这组输出及其有限阶微分进行表示，那么该系统即为微分平坦系统。

要判断四旋翼无人机是否满足微分平坦性质，可以采用通过确定其平坦输出这一最简单直接的方法进行判定。下面将针对运动学和动力学模型，分别找到输入由输出及其有限微分项的表达式，从而判定系统的平坦属性。

$$\ddot{x} = \theta g; \qquad I_{xx}\ddot{\theta} = U_\theta$$
$$\ddot{y} = -\phi g; \qquad I_{yy}\ddot{\phi} = U_\phi$$
$$m\ddot{z} = U_z - mg; \quad I_z\ddot{\psi} = U_\psi$$

因此, 系统的平坦输出 $F_1 = z, F_2 = x, F_3 = y, F_4 = \psi$ 。除此之外, θ 和 ϕ 表示成平坦输出的函数如下:

$$\theta = \frac{\ddot{F}_2}{g}; \phi = -\frac{\ddot{F}_3}{g}$$

控制输入表示成平坦输出 F 的函数为:

$$U_z = m\left(\ddot{F}_1 + g\right); U_\theta = I_{xx}\frac{F_2^{(4)}}{g}$$
$$U_\phi = -I_{yy}\frac{F_3^{(4)}}{g}; U_\psi = I_{zz}\ddot{F}_4$$

$F_1^*, F_2^*, F_3^*, F_4^*$ 分别是 F_1, F_2, F_3, F_4 的期望表达。微分参数化后, 如下式所示, 一个重要结果是: 如果系统沿着参考轨迹飞行, 那么参考轨迹的控制输入为:

$$U_z^* = m\left(\ddot{F}_1^* + g\right); U_\theta^* = I_{xx}\frac{F_2^{(4)^*}}{\mathrm{g}}; U_\phi^* = -I_{yy}\frac{F_3^{(4)^*}}{\mathrm{g}}; U_\psi^* = I_{zz}\ddot{F}_4^*$$

综上所述，四旋翼无人机满足微分平坦性质。

6 题六

6.1 题目

请说明 acceleration，jerk，snap 之间的关系。

6.2 解答

$r = Position$

$\frac{d\mathbf{r}}{dt} = \dot{\mathbf{r}}$ Velocity

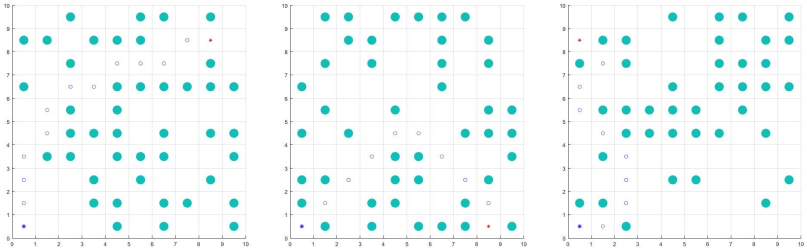
$\frac{d^2\mathbf{r}}{dt^2} = \ddot{\mathbf{r}}$ Acceleration

$\frac{d^3\mathbf{r}}{dt^3} = \dddot{\mathbf{r}}$ Jerk

$\frac{d^4\mathbf{r}}{dt^4} = \ddot{\dddot{\mathbf{r}}}$ Snap

在物理学中，位移是指定一个点、粒子或物体的位置变化的矢量。位置向量从参考点指向当前位置。位置向量关于时间的一阶、二阶导数是常见的速度和加速度 (Acceleration)；而三阶、四阶导数即为题目中的 Jerk，Snap。

7 Matlab 编程题



(a) 目标在右上角 (b) 目标在右下角 (c) 目标在左上角

7.1 分析

这个任务聚焦于二维栅格地图的路径搜索和障碍物躲避。起点默认设置为（1，1），绿色原点为障碍物。

其中的 A* 算法使用欧几里得距离作为启发函数，故具有完备性，搜索结果总是最优的，而且它探索的节点会比 Dijkstra 算法少得多。

选取的三种随机地图调高了障碍物的生成概率，以防结果路径趋于起点与终点的连线。从图中可以看出，无论终点位于何处，算法都搜索到了最优路径（除非无解）。

这份 MATLAB 代码并不高效，可以从以下两个方面改进：

- 改用优先队列作为容器
- 充分利用 MATALAB 的矩阵操作而不是 for 循环

上面搜索获得的最优路径是最短路径，但是实际应用时我们考虑的指标不一定是距离。对于智能体的规划，还需要考虑计算量（规划时间等）。故我认为 A* 算法的优点在于通过调节启发函数我们可以控制算法的速度和精确度。因为在一些情况，我们可能未必需要最短路径，而是希望能够尽快找到一个路径即可。

启发函数对 A* 算法影响：

1. 在极端情况下，当启发函数 h(n) 始终为 0，则将由 g(n) 决定节点的优先级，此时算法就退化成了 Dijkstra 算法。
2. 如果 h(n) 始终小于等于节点 n 到终点的代价，则 A* 算法保证一定能够找到最短路径。但是当 h(n) 的值越小，算法将遍历越多的节点，也就导致算法越慢。

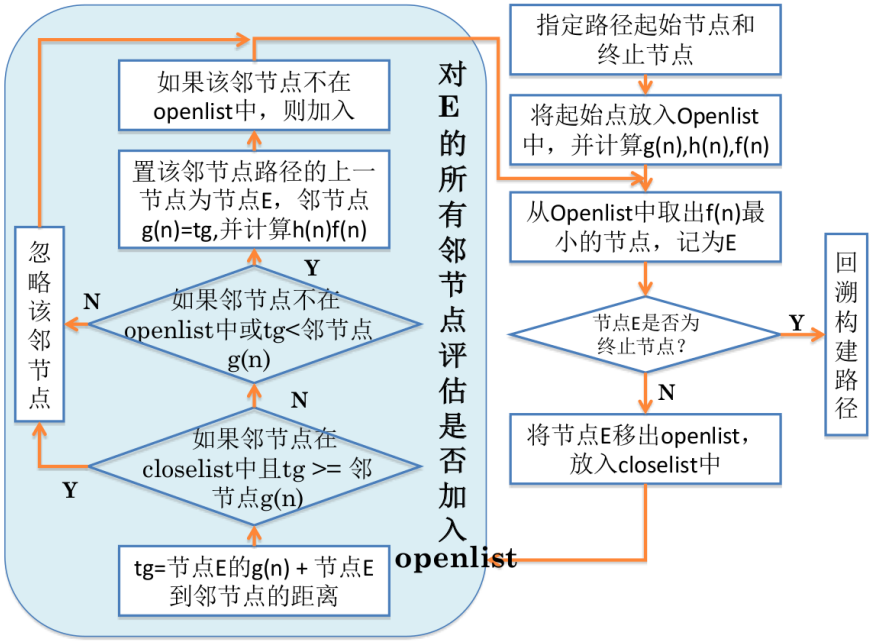
3. 如果 h(n) 完全等于节点 n 到终点的代价，则 A* 算法将找到最佳路径，并且速度很快。可惜的是，并非所有场景下都能做到这一点。因为在没有达到终点之前，我们很难确切算出距离终点还有多远。

4. 如果 h(n) 的值比节点 n 到终点的代价要大，则 A* 算法不能保证找到最短路径，不过此时会很快。

5. 在另外一个极端情况下，如果 h()n 相较于 g(n) 大很多，则此时只有 h(n) 产生效果，这也就变成了最佳优先搜索。

8 ROS 编程题

8.1 算法流程与运行结果



1. 实现 step 1 中的要求，即完成 getHeu 函数，在这个函数里接受两个指针参数 GridNodePtr，分别对他们的坐标求欧式距离或者曼哈顿距离；

2. 实现 step 2 中的要求，把在初始化函数中已经初始化好的 GridNodeMap 中每一个 node 的 fscore 设置好；并把 GridNodeMap 中代表路径起点的 node 的 id 设置为 1；

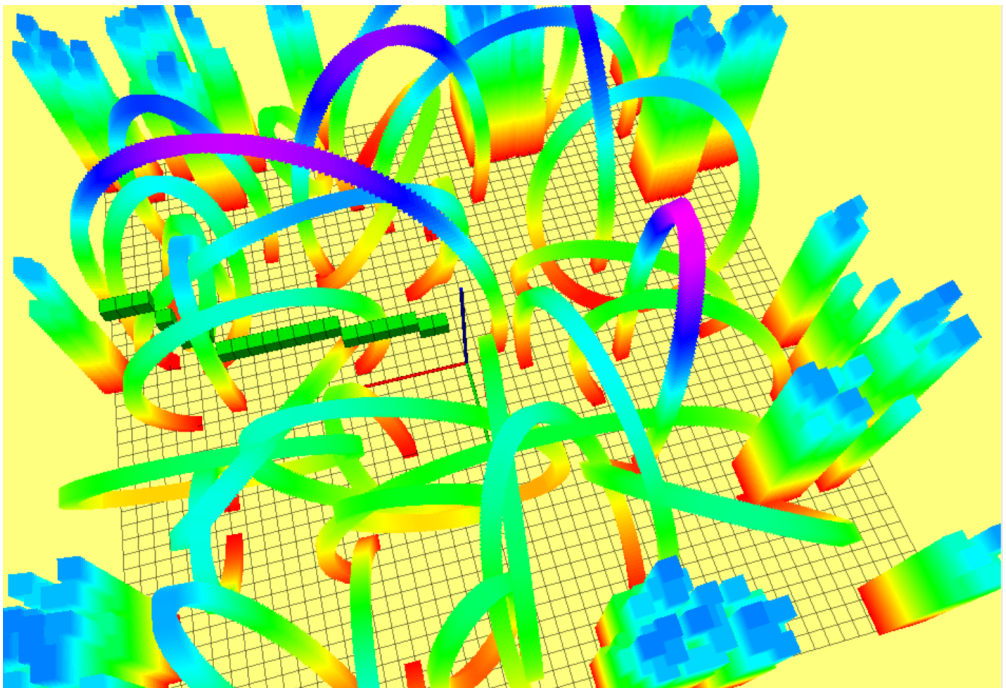
3. 实现 step 3 中的要求，将 currentPtr 设置为 openset 中的第一个，并把第一个从 openset 中删除；

4. 实现 step 4 中的要求，即实现 AstarGetSucc 函数，将 currentPtr 的 index 提取出来，找到 GridNodeMap 中此 index 在 xyz 轴前后移动一个单位的 node。用 isfree 判断是否非障碍和在地图范围内，判断为 true 则更新 node 的属性后加入到 neighborPtrSets 中。在我的实现中没用到 edgeCostSets。

5. 实现 step 5 和 6 中的要求，当 id=0 时创建一个新的 GridNodePtr 并加入到 openset 中；

6. 实现 step 7 中的要求，当 id=1 时，遍历 openset，将 openset 中与 neighborPtr 具有相同 index 的元素删除，重新创建一个新的 GridNodePtr，其属性与 neighborPtr 相同并加入到 openset 中；

7. 实现 step 8 中的要求，实现回溯，从 terminatePtr 的 camefrom 中一步步回到起点。



上图为 ROS 下的 3D 路径搜索结果，可以看出修改过后，路径不再是两点之间的直线，且在绕过障碍物的同时尽可能保证了是最短的。

8.2 不同启发式函数对运行效率的影响

实验一	Euclidean	Manhattan	Diagonal	Tie_Breaker
时间 ms	3.413633	1.632898	1.815696	2.943285
距离 m	1.360653	1.384084	1.3606353	1.3606353
visited_nodes	266	27	48	252

首先在这里可以分析 Euclidean 以及 Diagonal Heuristic 是满足 admissible 的, 而因为路径是可以斜向上前进的, 所以 Manhattan 是不满足 admissible 的, 因此使用 Manhattan 可以保证较快地搜索到可行路径, 但是不能满足 optimality; 使用 Euclidean 以及 Diagonal Heuristic 可以保证实现 optimality, 当然作为 trade-off, 搜索速度会下降一点。

可以发现 Manhattan 因为不是 admissible 的, 牺牲了一定的最优性换取了速度; 而 Diagonal 作为最 tight 的 heuristic, 实现最优性以及较快的速度。

8.3 是否加入 Tie Breaker 对效率的影响

实验二	Euclidean	Manhattan	Diagonal	Tie_Breaker
时间 ms	1.065659	0.781013	0.693911	0.891030
距离 m	0.895391	0.918823	0.895391	0.895391
visited_nodes	144	21	51	139

加入 Tie Breaker 的作用主要是打破 f 相等的平衡, 这里采取 h=h∗(1+p) , 这样子 h 大一点的点代表可能距离目标点比较远, 因此这样子调整后可以优先扩展距离目标点近的节点, 可以加快扩展速度, 同时因为只是稍微加大了一点, 不会不满足 admissible 的条件, 还可以实现更快的扩展。

Tie Breaker 是在 Euclidean 的基础上进行实验, 要与 Euclidean 进行对比。Tie Breaker 这里可以实现一定速度的提升, 相信如果选择更加合适的 Tie Breaker 可以换取更好的性能。

Tie Breaker 更加倾向于走直线，而不是走折线，当然这样也是有缺点的，打破了路径对称性后，比如正对着障碍时，我们期望的轨迹是绕过障碍物，而 Tie Breaker 则是走向障碍物再沿着障碍物绕过，出现了一定的浪费。所以 Tie Breaker 也给我们带来了一些麻烦，这时候有一个高阶的图搜索算法 JPS 跳点搜索可以解决这个问题。

8.4 遇到的问题以及解决方法

8.4.1 “STRP -1”

本次 ROS 部分的作业最大的困难在于对 linux 系统的不熟悉。完成作业需要用到 linux 系统，但平常只接触过 linux 的终端，没有使用过可视化界面。故开始完成作业前就不得不花费大量时间探索前置工作。

我相继采取了 windows 安装 ros，wsl 结合图形界面，linux 云服务器远程桌面等方法，最终通过 Windows Subsystem for Linux 完成了本次项目。

8.4.2 “STEP 0”

根据作业要求，共有 8 个 STEP 需要完成，在完成 Astar_searcher 相关的代码之后我发现无论怎样修改结果都是两点之间的连线，完全没有搜索出正确的路径。

耗费许久没有取得进展，最终才在 demo_node.cpp 发现了 “STEP 0”，不修改这部分就没办法应用搜索得到的路径。