

SearchBased motion planning

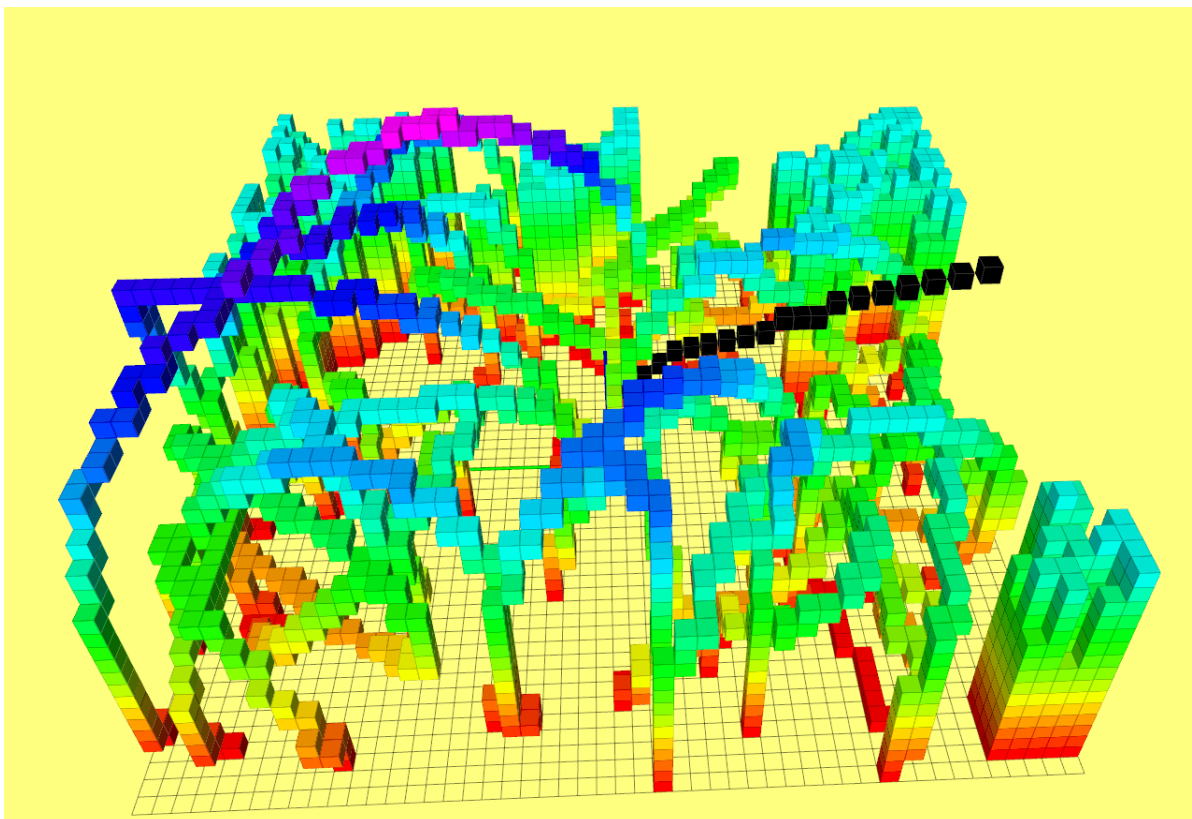
第二章作业

算法流程及运行效果

A星 算法

```
# A*算法流程
# 维持一个最优队列(priority queue)取存储被扩展的节点,代码中采用C++ STL的multimap实现,
# multimap将{key,value}当做元素,允许重复元素。multimap根据key的排序准则自动将元素排序,因此
# 使用时只需考虑插入和删除操作即可。
# 定义一个启发式函数,代码见getHeu()函数
# 将起点放入最优队列中
# 指定起点的 g 值为 0, 其余节点的 g 值为infinite
# 进入循环
#   如果队列为空, return false; break;
#   从最优队列中移除 $f(n) = g(n) + h(n)$ 最小的节点,该节点作为当前节点,并标志为已扩展
#   如果当前节点是终点,则 return true; break;
#   对于当前节点周围未被扩展的邻居节点:
#     如果  $g(m) = \text{infinite}$ 
#       设置  $g(m) = g(n) + C_{nm}$ ;
#       把当前节点加入最优队列
#     如果  $g(m) > g(n) + C_{nm}$ 
#       更新  $g(m) = g(n) + C_{nm}$ ;
#   end
# 结束循环
```

Rviz可视化, 路径点颜色为黑色



性能对比

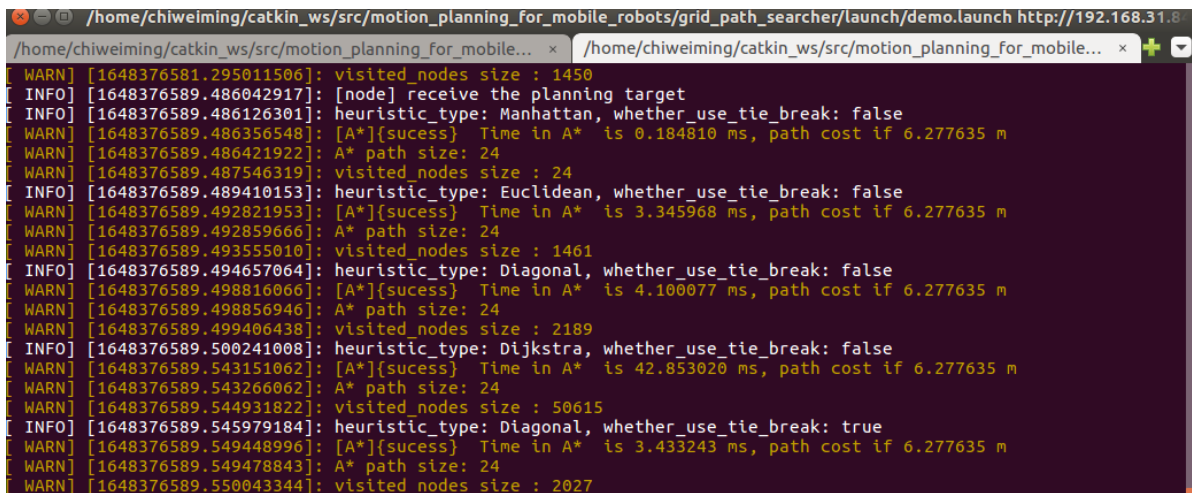
A星启发式函数及是否使用tie breaker对比

// 在AstarPathFinder中设置heuristic_type和use_Tie_breaker_变量，分别表示采用的启发式函数以及是否使用Tie breaker

// 在程序中我们同时对比不同启发式函数及其是否使用tie breaker，终端显示如下

在demo_node.cpp中实例化如下对象：

```
AstarPathFinder * _astar_path_finder = new AstarPathFinder(); // 使用
Manhattan作为启发式函数, 无Tie breaker
AstarPathFinder * _astar_path_finder_with_Euclidean = new
AstarPathFinder(Euclidean, false); // 使用Euclidean作为启发式函数, 无Tie breaker
AstarPathFinder * _astar_path_finder_with_Diagonal = new
AstarPathFinder(Diagonal, false); // 使用Diagonal作为启发式函数, 无Tie breaker
AstarPathFinder * _astar_path_finder_with_Dijkstra = new
AstarPathFinder(Dijkstra, false); // 使用Diagonal作为启发式函数, 无Tie breaker
AstarPathFinder * _astar_path_finder_with_Euclidean_TB = new
AstarPathFinder(Euclidean, true); // 使用Euclidean作为启发式函数, 启动tie breaker
```



The terminal output shows the results of A* searches for different heuristics and tie breaker settings. The output is as follows:

```
/home/chiweiming/catkin_ws/src/motion_planning_for_mobile_robots/grid_path_searcher/launch/demo.launch http://192.168.31.8:
/home/chiweiming/catkin_ws/src/motion_planning_for_mobile... x /home/chiweiming/catkin_ws/src/motion_planning_for_mobile... x
WARN [1648376581.295011506]: visited_nodes size : 1450
INFO [1648376589.486042917]: [node] receive the planning target
INFO [1648376589.486126301]: heuristic_type: Manhattan, whether_use_tie_break: false
WARN [1648376589.486356548]: [A*]{success} Time in A* is 0.184810 ms, path cost if 6.277635 m
WARN [1648376589.486421922]: A* path size: 24
WARN [1648376589.487546319]: visited_nodes size : 24
INFO [1648376589.489410153]: heuristic_type: Euclidean, whether_use_tie_break: false
WARN [1648376589.492821953]: [A*]{success} Time in A* is 3.345968 ms, path cost if 6.277635 m
WARN [1648376589.492859666]: A* path size: 24
WARN [1648376589.493555010]: visited_nodes size : 1461
INFO [1648376589.494657064]: heuristic_type: Diagonal, whether_use_tie_break: false
WARN [1648376589.498816066]: [A*]{success} Time in A* is 4.100077 ms, path cost if 6.277635 m
WARN [1648376589.498856946]: A* path size: 24
WARN [1648376589.499406438]: visited_nodes size : 2189
INFO [1648376589.500241008]: heuristic_type: Dijkstra, whether_use_tie_break: false
WARN [1648376589.543151062]: [A*]{success} Time in A* is 42.853020 ms, path cost if 6.277635 m
WARN [1648376589.543266062]: A* path size: 24
WARN [1648376589.544931822]: visited_nodes size : 50615
INFO [1648376589.545979184]: heuristic_type: Diagonal, whether_use_tie_break: true
WARN [1648376589.549448996]: [A*]{success} Time in A* is 3.433243 ms, path cost if 6.277635 m
WARN [1648376589.549478843]: A* path size: 24
WARN [1648376589.550043344]: visited_nodes size : 2027
```

Modality	Round	Run time(ms)	Path size	Visited size
Manhattan	1	2.61	32	1336
	2	4.65	29	2026
	3	0.24	25	35
	Mean	2.50	28.67	1132.30
Euclidean	1	6.33	26	4514
	2	7.94	28	6658
	3	2.49	25	1373
	Mean	5.59	26.33	1481.67
Diagonal	1	2.53	29	1992
	2	3.04	28	2664
	3	0.24	25	51
	Mean	1.94	27.33	1569
Dijkstra	1	41.94	26	55801
	2	41.71	28	55878
	3	35.43	25	44878
	Mean	39.69	26.33	52185
Euclidean+Tie breaker	1	3.59	28	2326
	2	3.66	29	2661
	3	0.21	25	30
	Mean	2.49	27.33	1672.33