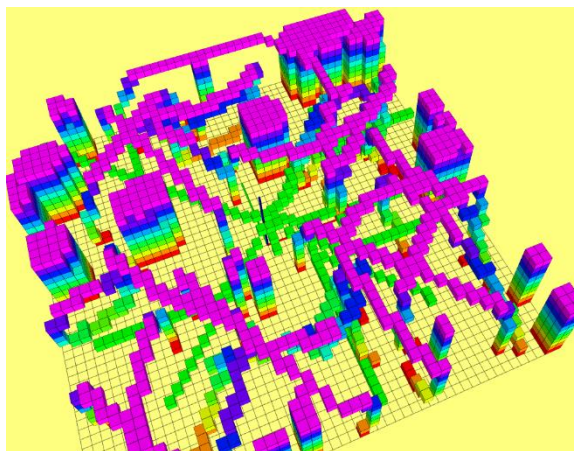


第二章作业（ROS 版本）

1. 算法流程

- 1.若现有点不为终点，则将该点从 openset 里删除，并加入 closedset
- 2.对该点周围 26 个点进行扩展，若未扩展过，则加入 openset；若扩展过了，则判断若现有点的 gn 小于之前的 gn，则覆盖原有信息，重新构建 nodeMapIt 的映射关系。
- 3.查找到终点后，利用 retrieve 函数通过 cameFrom 父节点指针信息依次将点加入到 gridPath 中
- 4.getPath 函数将 gridPath 中的各指针对应的点的 coord 坐标录入到 grid 数组内，并返回给 demo_node.cpp 中去，由此获得路径。

2. 运行结果：



```
/home/ruoyao/catkin_ws/src/grid_path_searcher/launch/demo.launch http://localhost:11311
* /random_complex/map/z_size: 2.0
* /random_complex/sensing/rate: 0.5
* /roslistro: kinetic
* /rosversion: 1.12.14
* /waypoint_generator/waypoint_type: manual-lonely-way...

NODES
  demo_node (grid_path_searcher/demo_node)
  random_complex (grid_path_searcher/random_complex)
  waypoint_generator (waypoint_generator/waypoint_generator)

ROS_MASTER_URI=http://localhost:11311

process[demo_node-1]: started with pid [8839]
process[random_complex-2]: started with pid [8840]
process[waypoint_generator-3]: started with pid [8851]
[INFO] [1571973787.937299117]: [node] receive the planning target
[WARN] [1571973787.937421404]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.040730 ms, path cost lf 4.507034 m
already get here
[WARN] [1571973787.937722861]: visited_nodes size : 19
[INFO] [1571973791.527302846]: [node] receive the planning target
[WARN] [1571973791.527563678]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.108663 ms, path cost lf 4.721247 m
already get here
[WARN] [1571973791.528095140]: visited_nodes size : 18
[INFO] [1571973801.327354167]: [node] receive the planning target
[WARN] [1571973801.327628944]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.176481 ms, path cost lf 4.907034 m
already get here
[WARN] [1571973801.328105347]: visited_nodes size : 21
[INFO] [1571973803.828398989]: [node] receive the planning target
[WARN] [1571973803.828129866]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.423115 ms, path cost lf 6.349675 m
already get here
[WARN] [1571973803.828797498]: visited_nodes size : 22
[INFO] [1571973807.767343174]: [node] receive the planning target
[WARN] [1571973807.767480222]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.208628 ms, path cost lf 4.440382 m
already get here
[WARN] [1571973807.768111810]: visited_nodes size : 16
[INFO] [1571973810.537292846]: [node] receive the planning target
[WARN] [1571973810.537462027]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.006740 ms, path cost lf 0.848528 m
already get here
[WARN] [1571973810.538100941]: visited_nodes size : 3
[INFO] [1571973813.547363300]: [node] receive the planning target
[WARN] [1571973813.547462290]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.103006 ms, path cost lf 3.960624 m
already get here
[WARN] [1571973813.548138007]: visited_nodes size : 29
[INFO] [1571973816.977329015]: [node] receive the planning target
[WARN] [1571973816.977689512]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.264302 ms, path cost lf 5.268483 m
already get here
[WARN] [1571973816.978295290]: visited_nodes size : 23
[INFO] [1571973822.277318611]: [node] receive the planning target
[WARN] [1571973822.277398472]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.192401 ms, path cost lf 4.477635 m
already get here
[WARN] [1571973822.278146740]: visited_nodes size : 15
[INFO] [1571973828.637295995]: [node] receive the planning target
[WARN] [1571973828.637603399]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.209447 ms, path cost lf 4.696910 m
already get here
[WARN] [1571973828.637997333]: visited_nodes size : 16
[INFO] [1571973836.377278588]: [node] receive the planning target
[WARN] [1571973836.377038053]: [A*](success), This is the test of the heuristic Diagonal, Time in A* is 0.458019 ms, path cost lf 5.736287 m
already get here
[WARN] [1571973836.378145555]: visited_nodes size : 17
```

3. 不同启发式函数对 A*运行效率的影响（均带有 tie_breaker）：

- Diagonal: 平均用时: 0.1117ms, 平均 path_cost: 3.685m 速度: 32.99m/ms
- Manhattan:平均用时: 0.2271ms, 平均 path_cost: 4.827m 速度: 21.25m/ms
- Euclidean: 平均用时: 0.3435ms, 平均 path_cost: 5.195m 速度: 15.12m/ms

由此可见该程序效率: diagonal>manhattan>Euclidean

4. 对比是否加入 tie_breaker 对 A*效率的影响

diagonal 未加入 tie_breaker 时:

平均用时: 0.2068ms , 平均 path_cost: 5.54m 速度: 26.79m/ms

与问题 3 中的结果对比, 可发现加入了 tie_breaker 的 diagonal 比未加入 tie_breaker 的效率更高

4.遇到的问题:

1.作为一个小白, c++有很多东西没有接触过, 例如 Eigen 库、make_pair、vector、push_back、*it->first 之类的用法都不明白, 因此花了很多时间来看懂给的代码。

2.ros 上运行程序出现死循环时, ubuntu 卡死了, 难以进行检查。同时修改代码后重新运行所花费的时间很长。

3.不过同时, 通过利用 multimap 自动排序和 vector 的函数, 能够使得代码的编写易于 matlab 版本的代码编写。