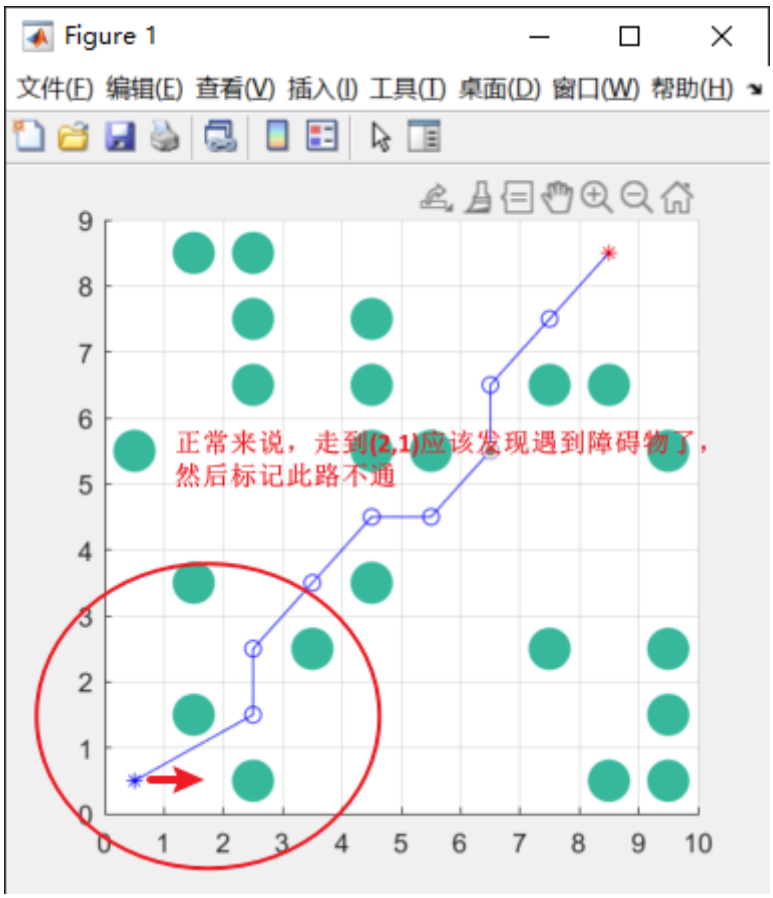
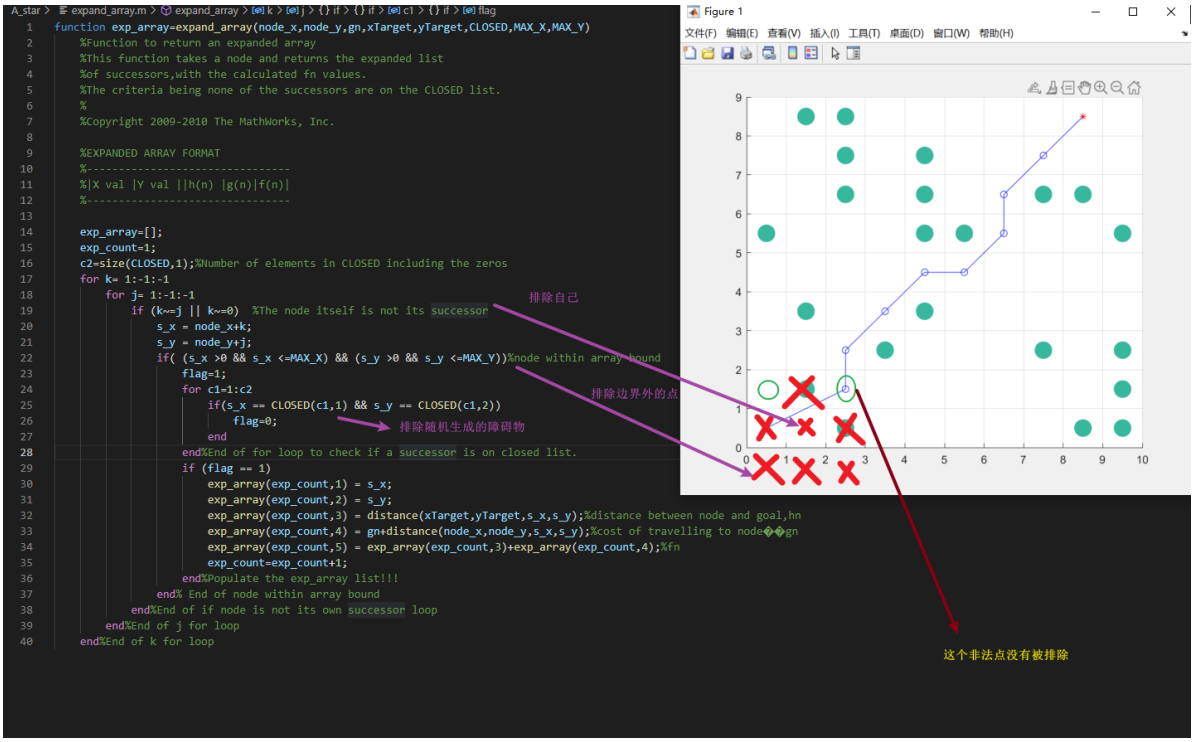
zqchen-第 2 章作业
MATLAB 版本
C++ 版本
在 WSL2 中出现的出现的找不到头文件的问题
DELL-Ubuntu：把新的三个功能包拷贝到新的工作空间
最终效果截图演示
保存障碍物地图到 .pcd 文件中
算法运行效率的比较以及是否加入 Tie Breaker 产生的影响
A* 算法采用 Euclidean 和 Diagonal 的效率对比
实验1：(25, 25, 0) ----> (29, 25, 0)
实验2：(25, 25, 0) ----> (49, 32, 0)
加入 Tie Breaker 对效率的影响
实验3：Euclidean + p=0.001
实验4：Diagonal + p=0.001
实验5：Diagonal + h*(1 + p=0.001/0.005) 或 Diagonal + h + 0.001*dot_product
对算法的理解和编程实现上的心得体会

zqchen-第2章作业

MATLAB版本

最开始2022-03-22周二开始做MATLAB版本的A-star算法，最后花了5个多小时调试成功，但是发现了有一个作业内置函数实现的问题，这个问题我尝试修复了好几个小时仍然没有成功，时间紧张最后就没有继续深究，暂时知道这个问题是存在的，以后有时间再试一下。





这张图片对应的随机障碍物地图我用 `MATLAB` 保存到了 `map_wrong_1.mat` 文件中，想要复现我的问题，只需把作业的 `main.m` 中随机生成障碍物地图的那行注释掉，然后新加一行把 `map` 变量加载进来即可。

```
% Used for Motion Planning for Mobile Robots
% Thanks to HKUST ELEC 5660
close all; clear all; clc;
addpath('A_star')
```

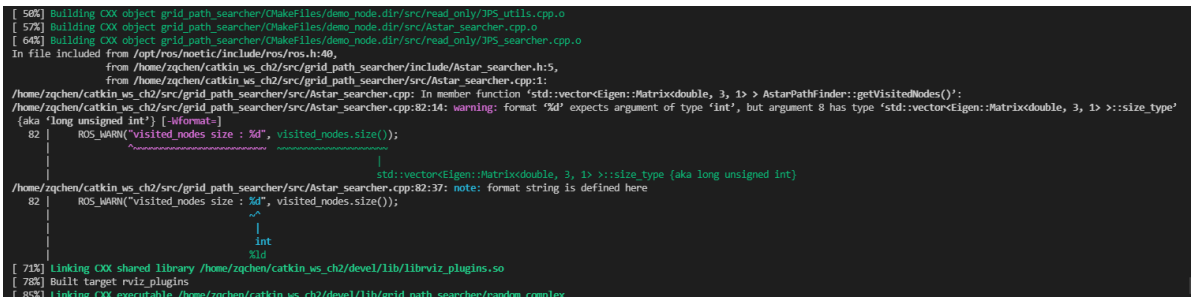
```
% Environment map in 2D space
xStart = 1.0;
yStart = 1.0;
xTarget = 9.0;
yTarget = 9.0;
MAX_X = 10;
MAX_Y = 10;

%如果想要复现我的问题，只需把下面这行注释掉，然后新加一行把map变量加载进来即可
%map = obstacle_map(xStart, yStart, xTarget, yTarget, MAX_X, MAX_Y);
load("C:\Users\zhuoqun.chen\Desktop\2022Spring\深蓝学院-路径规划课程
\hw_2\map_wrong_1.mat", "map")

% Waypoint Generator Using the A*
path = A_star_search(map, MAX_X,MAX_Y);

% visualize the 2D grid map
visualize_map(map, path, []);

% save map
% save('Data/map.mat', 'map', 'MAX_X', 'MAX_Y');
```

C++版本

刚开始本来是想在 Thinkpad-Win10 中的 WSL2 中运行的，但是配置完后发现居然突然打开不了 RVIZ 了，所以这次只能在 DELL-Ubuntu20.04 中测试了。做第 1 章作业的时候起的是 catkin_ws，这次第 2 章新建一个 catkin_ws_ch2。

在WSL2中出现的出现的找不到头文件的问题

首先 catkin_make 编译的时候出现了警告，但是还是编译通过了，不知道对后续有无影响（做完作业后发现并无影响）：



编译完成后发现各个文件的最开始的包含头文件的地方一直有红色波浪线的报错，参考了一些 CSDN 上的文章，发现是可以通过在编译的时候额外添加一个 CMAKE 选项

```
catkin_make -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
```

使得生成一个`compile_commands.json`的文件，把这个文件加入到`VS Code`的`C/C++扩展插件`的配置中就可以不报错了，至于这个是不是一个最优的方案再另说，首先保证解决当下的问题。

`DELL-Ubuntu`：把新的三个功能包拷贝到新的工作空间

在`DELL-Ubuntu20.04`下：

把三个功能包拷贝到`/usr/local/catkin_ws_ch2/src/`路径下（`/home`目录内存不够了，我以前记录过这个问题），进入到`src`目录下`catkin_init_workspace`后返回到`catkin_ws_ch2`根目录`catkin_make`，这时会报错，错误和第`1`章的作业一样的，只需要设置`C++`标准为`14`并且把分布的`topic`由`"/world"`改为`"world"`即可。

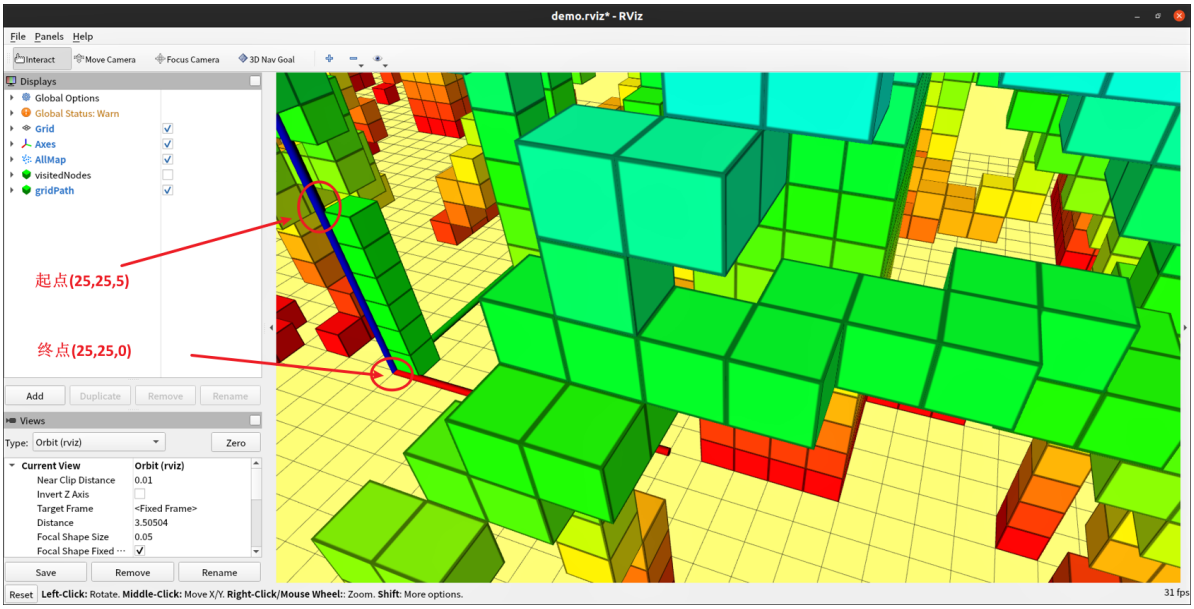
编译成功后先不要急着运行，因为肯定是无法运行的，此时我还没有开始实现`A-star`算法呢。但是此时`roscore`后`RVIZ`是能正常进去的。

使用上面相同的步骤，不知道为什么在`Horizon-Thinkpad`上出现`RVIZ`不能正常启动的情况。

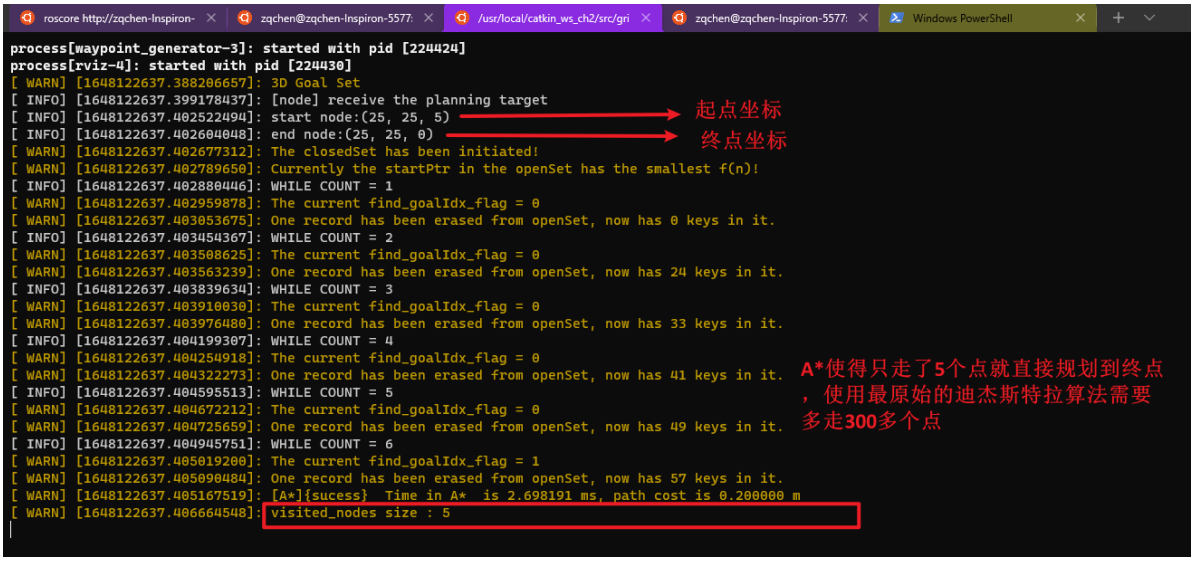
最终效果截图演示

`2022-03-24周四晚`ROS上的`C++`版本的代码终于调试成功了！超级鸡冻！

在`RVIZ`中点击两次鼠标左键，生成目标节点的位置：



运行`A-star`算法成功的命令行：



如果保持`getHeu()`函数内部实现为空，即使得`h值`一直为`0`，那么实现的就是`Dijkstra`算法，没有欧几里得距离对应的`h值`作为贪心策略的指引，效率比`A*`要慢很多，经过测试，即使是最简单的点也需要多走好几百个点才能找到最优路径。

保存障碍物地图到`.pcd`文件中

为了更好的比较几种基于图搜索的算法的优劣，有必要控制变量进行比较：控制相同的起点并且使用同一个随机生成的地图。

首先打开`RVIZ`，对`rcvPointCloudCallBack()`函数进行一下修改，使得把`ROS`传来的点云`sensor_msgs::PointCloud2`先转化为`pcl`库的点云格式`pcl::PointCloud<pcl::PointXYZ>`后存储到`/usr/local/catkin_ws_ch2/test_temp_pcd.pcd`文件中。

注意不要直接写`test_temp_pcd.pcd`，参考Using savePCDFileASCII, PCD file isn't created (no errors though) - ROS Answers: Open Source Q&A Forum，如果直接写会在当前目录中找不到这个生成的文件的，因为默认会放在`/.ros`目录下。之后执行`catkin_make`并运行`roslaunch grid_path_searcher demo.launch`。

记得在`demo_node.cpp`的开头额外添加一个头文件（参考：pcl-ros-tutorial/PCL Reference with ROS.md at master · methylDragon/pcl-ros-tutorial）：

```
#include <pcl/io/pcd_io.h>
```

```
void rcvPointCloudCallBack(const sensor_msgs::PointCloud2 & pointcloud_map)
{
    if(_has_map ) return;

    pcl::PointCloud<pcl::PointXYZ> cloud;
    pcl::PointCloud<pcl::PointXYZ> cloud_vis;
    sensor_msgs::PointCloud2 map_vis;

    pcl::fromROSMsg(pointcloud_map, cloud);

    if( (int)cloud.points.size() == 0 ) return;
```

```
    pcl::PointXYZ pt;
    for (int idx = 0; idx < (int)cloud.points.size(); idx++)
    {
        pt = cloud.points[idx];

        // set obstalces into grid map for path planning
        // 将障碍物信息设置进入栅格化地图，为后续路径规划做准备
        _astar_path_finder->setObs(pt.x, pt.y, pt.z);
        _jps_path_finder->setObs(pt.x, pt.y, pt.z);

        // for visualize only
        // 可视化地图部分
        Vector3d cor_round = _astar_path_finder-
>coordRounding(Vector3d(pt.x, pt.y, pt.z));
        pt.x = cor_round(0);
        pt.y = cor_round(1);
        pt.z = cor_round(2);
        cloud_vis.points.push_back(pt);
    }

    cloud_vis.width    = cloud_vis.points.size();
    cloud_vis.height   = 1;
    cloud_vis.is_dense = true;

    /*****************************************************************/
    //added by zqchen to save the point_cloud to a .pcd file
    //to get a fixed map to compare the performance among different
algorithms.
    pcl::io::savePCDFileASCII ("/usr/local/catkin_ws_ch2/test_temp_pcd.pcd",
cloud_vis);
    ROS_WARN("Generate .pcd Success!");
    /*****************************************************************/

    pcl::toROSMsg(cloud_vis, map_vis);

    map_vis.header.frame_id = "world";
    _grid_map_vis_pub.publish(map_vis);

    _has_map = true;
}
```

这样点云就保存在该文件中了。

之后我们可以一直使用这个地图，只需加载进来即可，还需修改`rcvPointCloudCallBack()`函数，两段`***`注释行中间的是新增的和注释掉的部分：

```
void rcvPointCloudCallBack(const sensor_msgs::PointCloud2 & pointcloud_map)
{
    if(_has_map ) return;
```

```
    pcl::PointCloud<pcl::PointXYZ> cloud;
    pcl::PointCloud<pcl::PointXYZ> cloud_vis;
    sensor_msgs::PointCloud2 map_vis;

    //把这行注释掉，可以下次不使用新的随机生成的地图，而是使用我之前保存的.pcd文
件载入
    // pcl::fromROSMsg(pointcloud_map, cloud);

    /*****************************************************************/
    if (pcl::io::loadPCDFile<pcl::PointXYZ>
("/usr/local/catkin_ws_ch2/test_temp_pcd.pcd", cloud) == -1)
    {
        ROS_WARN("Couldn't read file test_temp_pcd.pcd!");
        return;
    }
    ROS_WARN("Load test_temp_pcd.pcd Success!");//点数应该为63059
   /*****************************************************************/

    if( (int)cloud.points.size() == 0 ) return;

    pcl::PointXYZ pt;
    for (int idx = 0; idx < (int)cloud.points.size(); idx++)
    {
        pt = cloud.points[idx];

        // set obstalces into grid map for path planning
        // 将障碍物信息设置进入栅格化地图，为后续路径规划做准备
        _astar_path_finder->setObs(pt.x, pt.y, pt.z);
        _jps_path_finder->setObs(pt.x, pt.y, pt.z);

        // for visualize only
        // 可视化地图部分
        Vector3d cor_round = _astar_path_finder-
>coordRounding(Vector3d(pt.x, pt.y, pt.z));
        pt.x = cor_round(0);
        pt.y = cor_round(1);
        pt.z = cor_round(2);
        cloud_vis.points.push_back(pt);
    }

    cloud_vis.width    = cloud_vis.points.size();
    cloud_vis.height   = 1;
    cloud_vis.is_dense = true;


   /*****************************************************************/
    //added by zqchen to save the point_cloud to a .pcd file
    //to get a fixed map to compare the performance among different
algorithms.
```

```
    // pcl::io::savePCDFileASCII
("/usr/local/catkin_ws_ch2/test_temp_pcd.pcd", cloud_vis);
    // ROS_WARN("Generate .pcd Success!");
    /**********************************************************/

    pcl::toROSMsg(cloud_vis, map_vis);

    map_vis.header.frame_id = "world";
    _grid_map_vis_pub.publish(map_vis);

    _has_map = true;
}
```

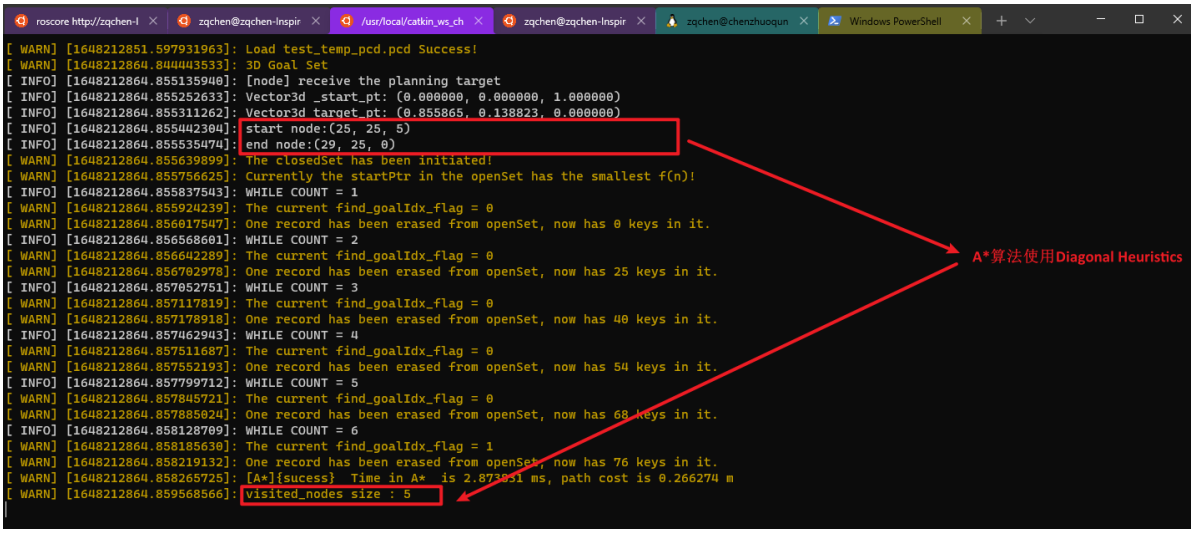
再次编译运行后可以看到地图还是原来生成的那个地图。之后我们就可以在同一个点云障碍物地图中做测试了。

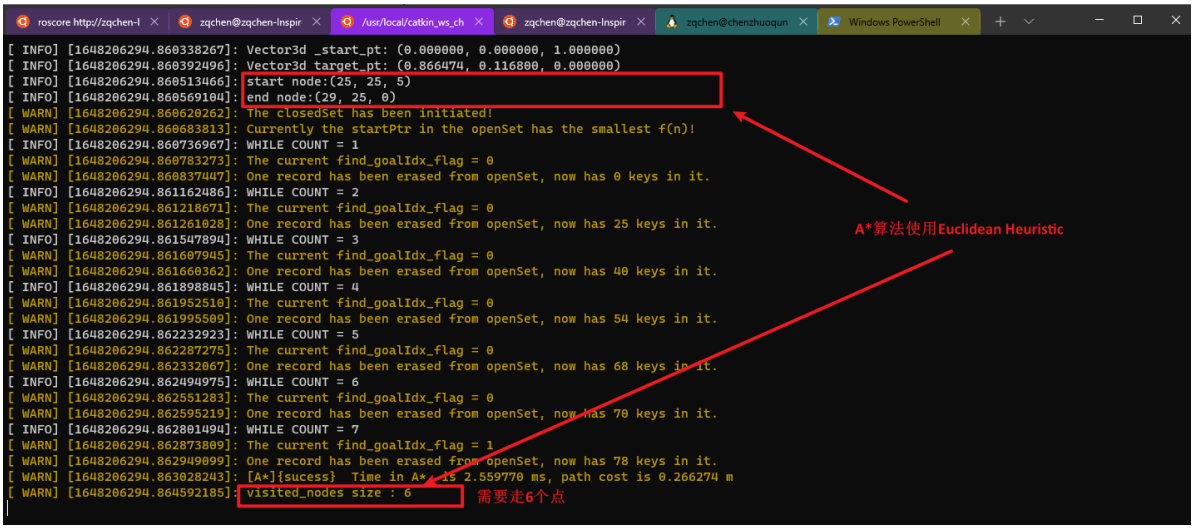
算法运行效率的比较以及是否加入`Tie Breaker`产生的影响

`A*`算法采用`Euclidean`和`Diagonal`的效率对比

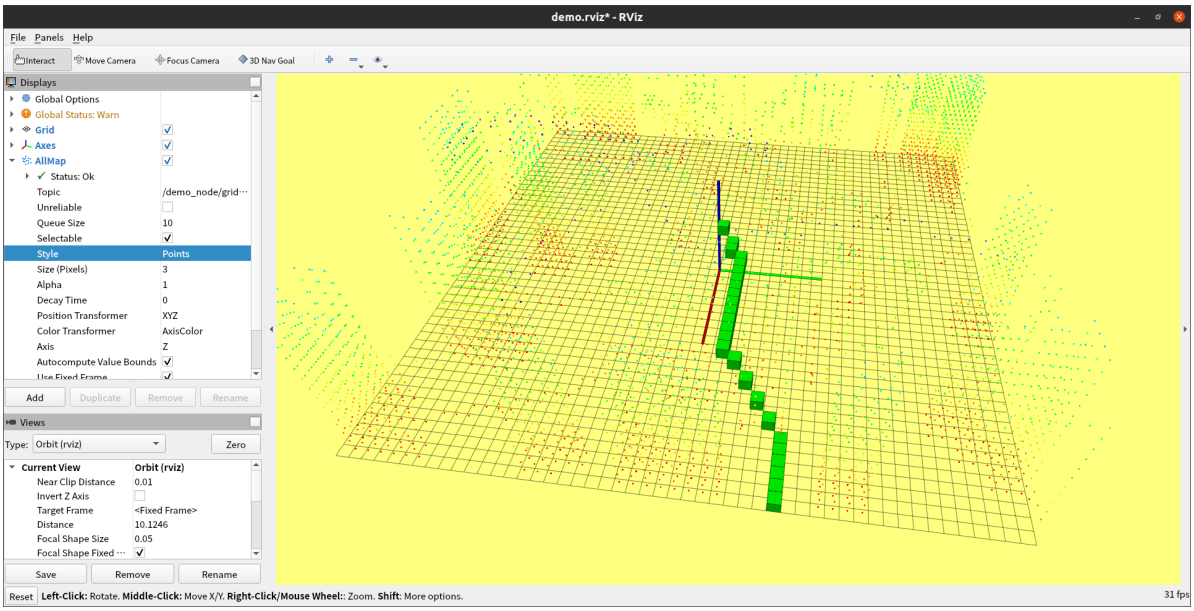
在做实验的过程中，对`A*`算法分别采用了两种启发式函数：`Euclidean`和`Diagonal`。实验结果证明：二者产生的路径都是最优的，但是在三维空间规划时`Diagonal`的效率更高，因为其约束更紧凑。课程中只讲解了二维的`Diagonal`，三维的情况类似，参考python - Calculating 'Diagonal Distance' in 3 dimensions for A* path-finding heuristic - Stack Overflow可以获得三维中`h`的表达式。

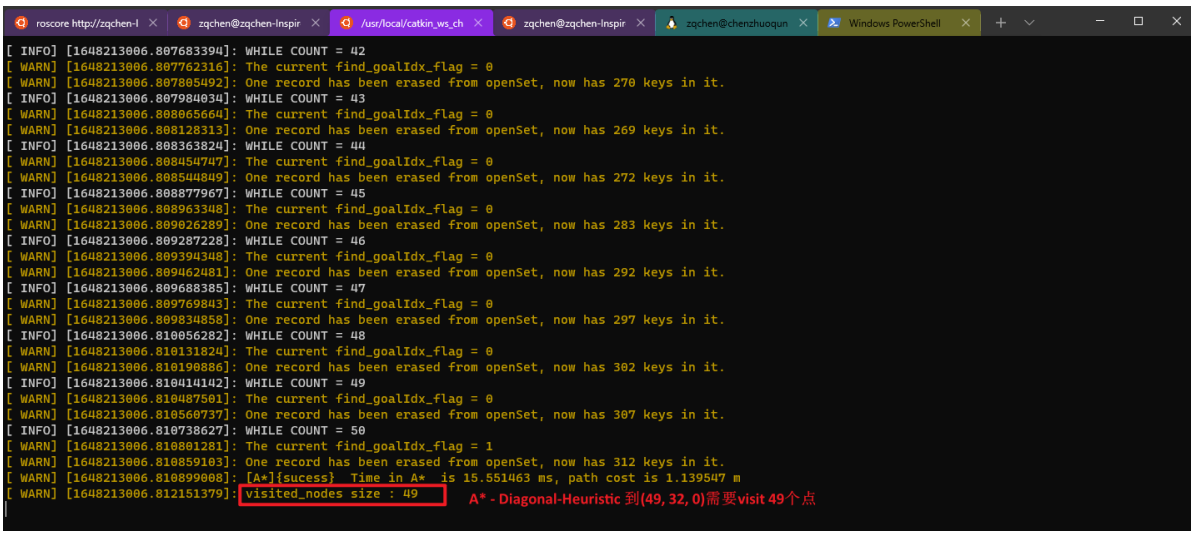
实验1：`(25, 25, 0)` ----> `(29, 25, 0)`

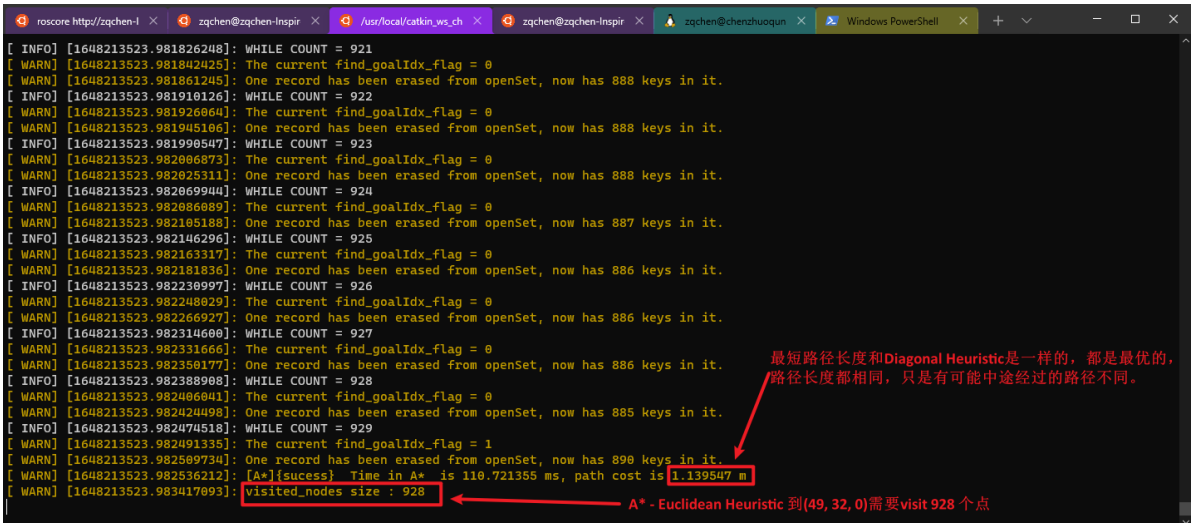




实验2: (25, 25, 0) ----> (49, 32, 0)

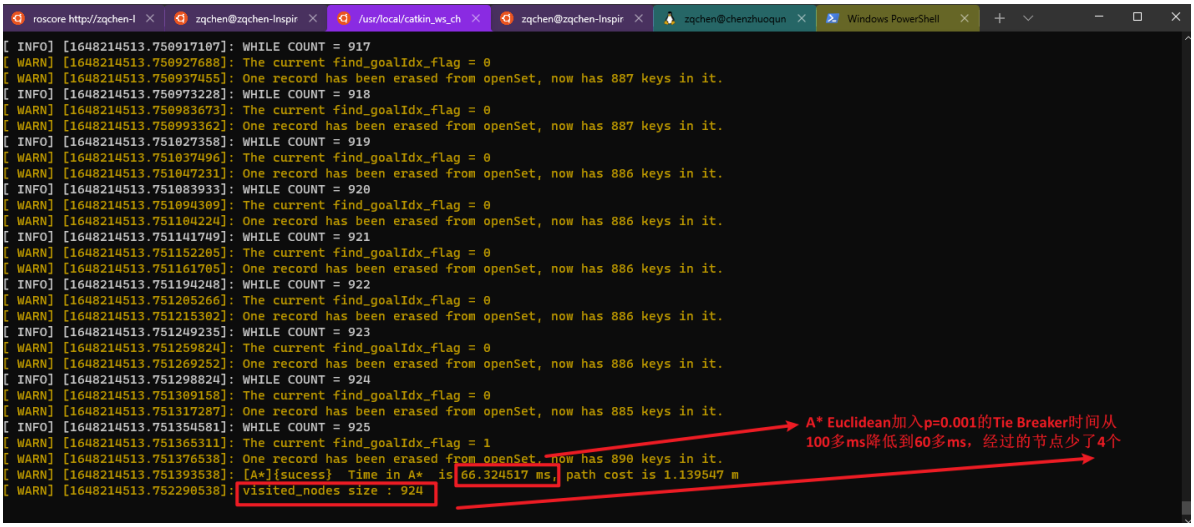




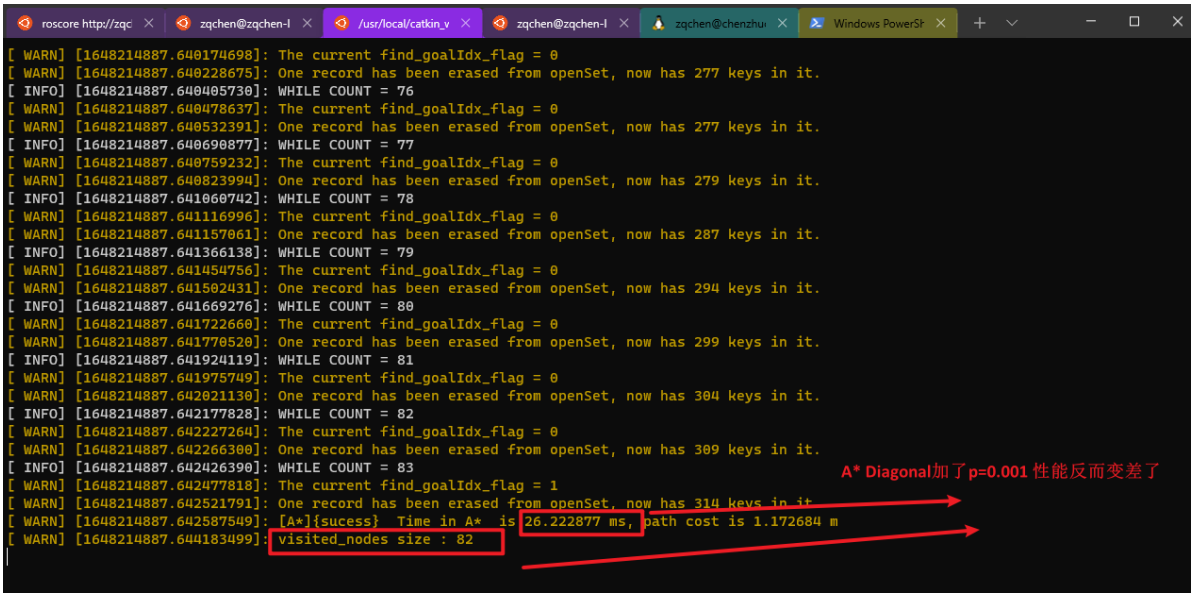


加入`Tie Breaker`对效率的影响

实验3：`Euclidean` + `p=0.001`



实验4：`Diagonal` + `p=0.001`



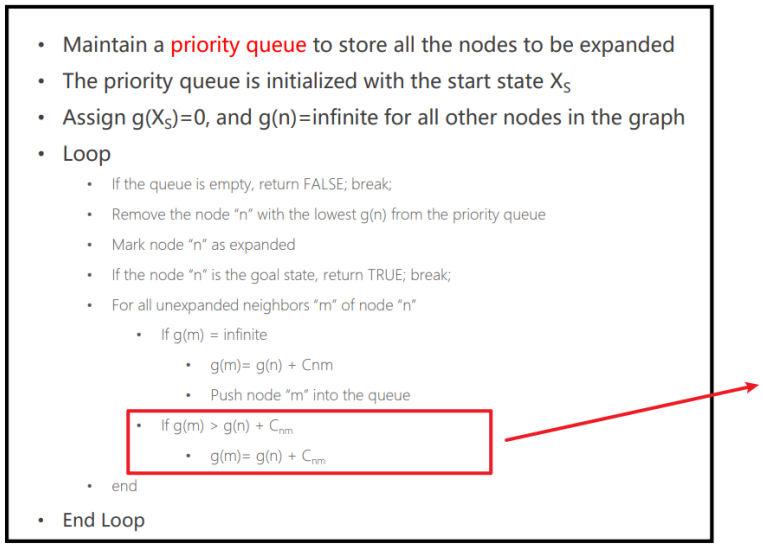
实验5： `Diagonal` + `h*(1 + p=0.001/0.005)` **或** `Diagonal` + `h + 0.001*dot_product`

下表通过固定地图，详细分析了从起点 `(25, 25, 0)` ----> 终点 `(49, 7, 0)` 使用基于 `A*-Diagonal Heuristic` 的算法是否加入 `Tie Breaker` 以及加的 `Tie Breaker` 种类对算法效率的影响。

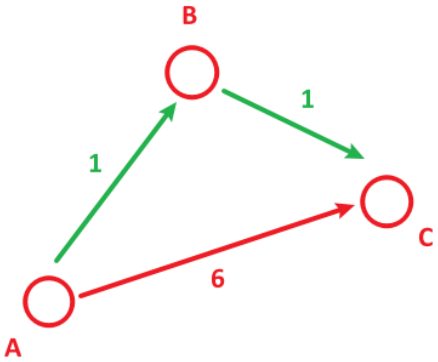
是否加 `Tie Breaker` 以及加的 `Tie Breaker` 种类	`visited nodes`	`path cost`	`time`	**备注**
`h`	`157`	`1.347228m`	`37ms`	无
`h - 0.001*dot_product`	`172`	`1.345233m`	`43ms`	按理来说这个应该是更快的呀
`h + 0.001*dot_product`	`118`	`1.351083m`	`13ms`	不知为何符号反过来更优（？）
`h * (1 + p=0.001)`	`124`	`1.354938m`	`13ms`	无
`h * (1 + p=0.005)`	`113`	`1.354938m`	`12.99ms`	无

对算法的理解和编程实现上的心得体会

Dijkstra’s Algorithm



我觉得，在这个地方在把 `expanded nodes` 压入 `priority quene` 的时候，不光要更新这个数值本身，也需要更新更小的 `g(m)` 的 `parent` 变成了谁（即当前更小的是经过了哪个新节点才到这个节点的），这样才可以在最终找到目标终点后往前回溯时找到正确的父节点，不然路径就是错乱的。



举个例子，比如说从 `A` 节点出发，第一次会把 `B` 和 `C` 都压入到优先级队列中，由于 `B` 和 `C` 都是第一次被探索到，所以 `g` 被更新为 `1` 和 `6`，然后下一次 `B` 节点被弹出，然后发现了在 `open_list` 中的 `C`，这时 `1+1` 显然比 `6` 要更优，那么不光要更新 `C` 节点的 `g` 值，还要记录下来：`C` 现在不应该 `cameFrom` 节点 `A`，而是更新为 `cameFrom` 节点 `B`，这样才是更优的。这样做，才能使得最终路径回溯的时候找到最优路径。

后来，我发现有个结论，参考Heuristics这篇文章：`GridMap` 上根据移动方向局限程度的不同，使用何种 `Heuristic` 函数是最优的，有一些通用的标准结论的。在我们的实验中所有方向都是 `8` 自由度的，因此不难理解为何 `Diagonal` 会比 `Euclidean` 收敛速度更快。

Heuristics for grid maps

On a grid, there are well-known heuristic functions to use.

Use the distance heuristic that matches the allowed movement:

- On a square grid that allows **4 directions** of movement, use Manhattan distance (L_1).
- On a square grid that allows **8 directions** of movement, use Diagonal distance (L_∞).
- On a square grid that allows **any direction** of movement, you might or might not want Euclidean distance (L_2). If A* is finding paths on the grid but you are allowing movement not on the grid, you may want to consider other representations of the map.
- On a hexagon grid that allows **6 directions** of movement, use Manhattan distance adapted to hexagonal grids.