Electronics Design Lab Report #4

Introduction

This assignment is about navigation techniques in the autonomous driving, it uses A star planning algorithm. I learnt about various motion planning methods especially A star and RRT. They are both the graph search algorithms that represent a map as a discrete square of particular size each one representing a particular point. This makes a graph out of path where each movement may be calculated by cost function and used to make decision continue the movement or not like in A-star. Thus, a set of actions are discretized which opens a way for implementation of various algorithms.

A-star is a relatively simple algorithm, which knows the exact location of the robot and target coordinates that it is trying to reach beforehand. At each step, surrounding area is analyzed to see if it is suitable for driving - no obstacle is present there. If so, a score for each possible step is calculated, which considers a distance from the current position to next step and distance from next step to the goal. However, such calculation does not take into account any obstructions on the way, it analyses straight line distance. Next step is made by going to the position, which has the lowest score, thus is closest to the goal. Disadvantages of this method is its inefficiency as a robot may go around for long time until the optimal path is found, so it may be slow. However, its simplicity is a plus, also it may be modified in numerous ways like making more complicated cost function that takes into account not only the distances, but the type of the surface it is moving which may be helpful in the real world cases where the car should plan its way in not only the closest distance, but also a good road type instead of sand, grass etc.

RRT is a probabilistic based search algorithm. In some cases, using random choices can be very efficient. This idea was incorporated in probabilistic based search algorithms like Real-Time Randomised Path Planning (RRT) method. It is also using heuristics to move towards the goal, but that happens only every 10th of a step or so. Other times it just simply goes random direction, building up tree-like structure. This type of algorithms are based on the idea, that often there are many obstacles between starting point and the goal and much time can be wasted by trying to go straight towards the target and hitting dead ends. Instead, larger area around is being searched, thus alternative path can be quickly found. Main advantages of RRT are: uniformly distributed search - it covers all area, always moves towards the goal (not necessarily every step) and the simplicity of it provides high execution speed.

In the implementation of this algorithm I learnt more information about how to use ros and especially rviz, how to publish nav/occupancy grid message and read the path topic. Additionally, finding the obstacles from the message information was another challenge for me and I understood how to recognize the obstacle using the sensor information message.

Explanation

TODO1

```
// TODO #1 Fill the code
// 1. set a origin and target in the grid map.
// 2. Add collision using generator.addCollision()
// 3. Visualize a collision points using "collisionCloudMsg"

AStar::Vec2i origin = { 0, 0 };
AStar::Vec2i target = { target_x, target_y };

double shiftx = 0.0;
double shifty = 0.0;
// normalize
if (target.x < 0){
    origin.x -= target.x;
    shiftx = target.x;
    target.x = 0;
     }
    if (target.y < 0){
    origin.y -= target.y;
    shifty = target.y;
    target.y = 0;
 }
```

We simply put the origin in the center which is zero. After that, the target is simply a its target components that was extracted from the message and put in the Vec2i form. However we have somehow deal with the negative value coordinates as A-star algorithm works with only positive values. For this, we add displacement part when some of the value is negative. So whenever one of the component is negative, we put it to the center and modify the origin by adding the same translation from which the target component was ignored. We also remover this displacement as we need to modify the path in this condition. That's why variables shiftx and shifty were created.

```
for (unsigned int width = 0; width < grid_x_size; width++)
{
    for (unsigned int height = 0; height < grid_y_size; height++)
    {
    //float sens = msg.data[width + height * msg.info.resolution];
    int sens = msg.data[width + height * msg.info.width];
    //std::cout << sens << std::endl;
    if (sens > 80){
        std::cout << "obstacle" << std::endl;
        pcl::PointXYZI pointBuf;
        pointBuf.x = width * msg.info.resolution + msg.info.resolution / 2 + msg.info.origin.position.x;
        pointBuf.y = height * msg.info.resolution + msg.info.resolution / 2 + msg.info.origin.position.y;
        pointBuf.z = 0.0;
        pointBuf.intensity = 1;
        collision_point_ptr->push_back(pointBuf);

        AStar::Vec2i temp;
        temp = { width, height };
        generator.addCollision(temp,5);
     }
    }
}
```

To add collisions into our form make the double for loop to go thought each point in the grid map and whenever the point is black (its sens is 100) means it is obstacle and this is collision point, thus there is if statement and when the value of sensor is greater than 80 we count it as a collision. Like last homework, we add this point to the collision pointer and then put is coordinates to our map using addCollision(..).

TODO2

```
auto path = generator.findPath(origin, target);
```

Everything is done with a simple function call that is in another file. Let's discuss how it is going.

1. It initializes the open list
2. It Initializes the closed list and puts the starting node on the open list and sets f = 0
3. while the open list is not empty
   a) find the node with the least f on the open list, let's say "q"
   b) pop q off the open list
   c) generate q's successors and set them parents to q in order to know other directions
   d) for each successor
      i) if successor is the goal, stop search
         successor g = q.g + distance between successor and q
         successor h = distance from goal to successor. Next simple equation
         successor.f = successor.g + successor.h
      ii) if a node with the same position as successor is in the open list which has a lower f than successor, skip this successor
      iii) if a node with the same position as successor is in the closed list which has a lower f than successor, skip this successor

otherwise, add the node to the open list

end (for loop)

e) push q on the closed list

end (while loop)

Note that if the path encounters and obstacle it deletes the current path from the list and starts again searching for another path. That's how it moves around the obstacles.

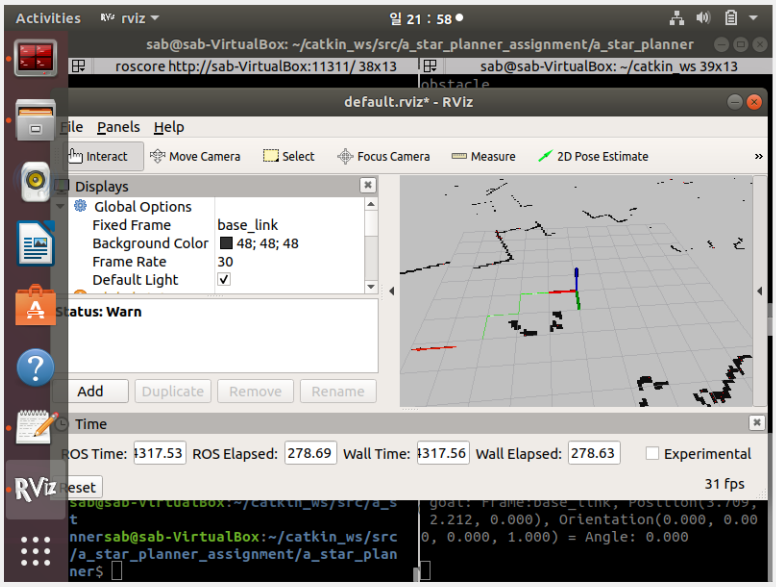
TODO3

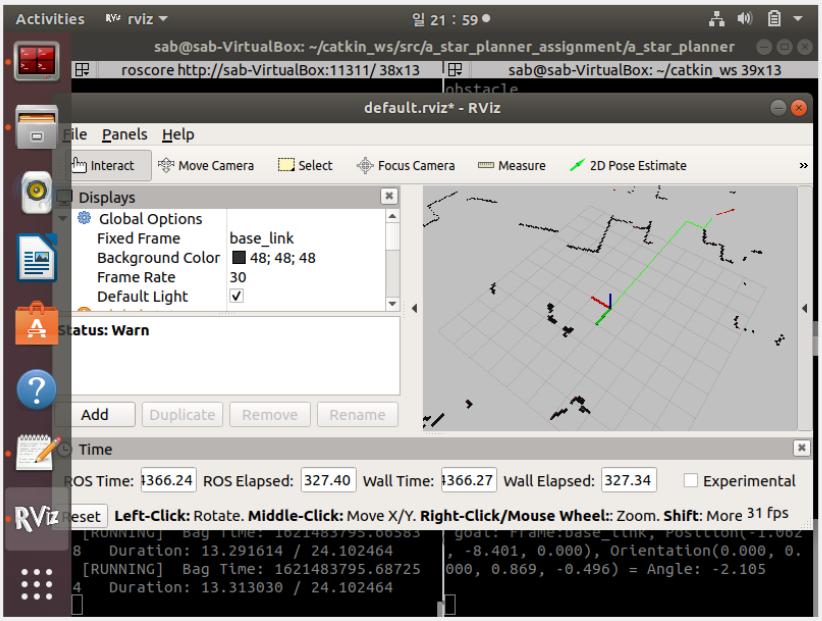
```
// TODO #3: Fill the code
// 1. Publish the result using nav_msgs/Path.msg
geometry_msgs::PoseStamped pose_stamped;
pose_stamped.header.seq = 0;

for (auto& point : path)
{
    pose_stamped.header.stamp = ros::Time::now();
    pose_stamped.header.frame_id = "base_link";

    pose_stamped.pose.position.x = point.x + shiftx;
    pose_stamped.pose.position.y = point.y + shifty;
    pose_stamped.pose.position.z = 0;

    AStartPathMsg.poses.push_back(pose_stamped);
    pose_stamped.header.seq++;
}

pubAstarPath.publish(AStartPathMsg);
```

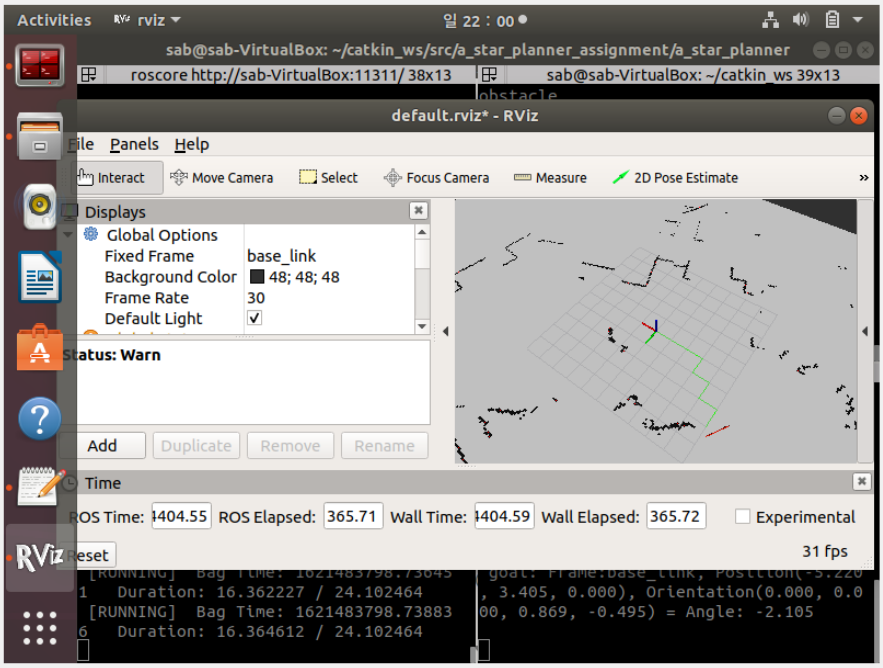
We extract the information from the path and add it to the occupancy grid message. With the help of documentation, we find that occupancy grid possesses the same components. For loop as for each point in the path we extract the position coordinated and put it with some index which shows the particular point order. We also add the shifts that were created in TODO1, if the values were positive, shifts are 0 and nothing changes. In the case of negative values, we get the translation shift that was created due to the shift of origin and target and we normalize it my adding (subtracting as its value is negative) the same shift in the resultant message. So, after that we publish the message.

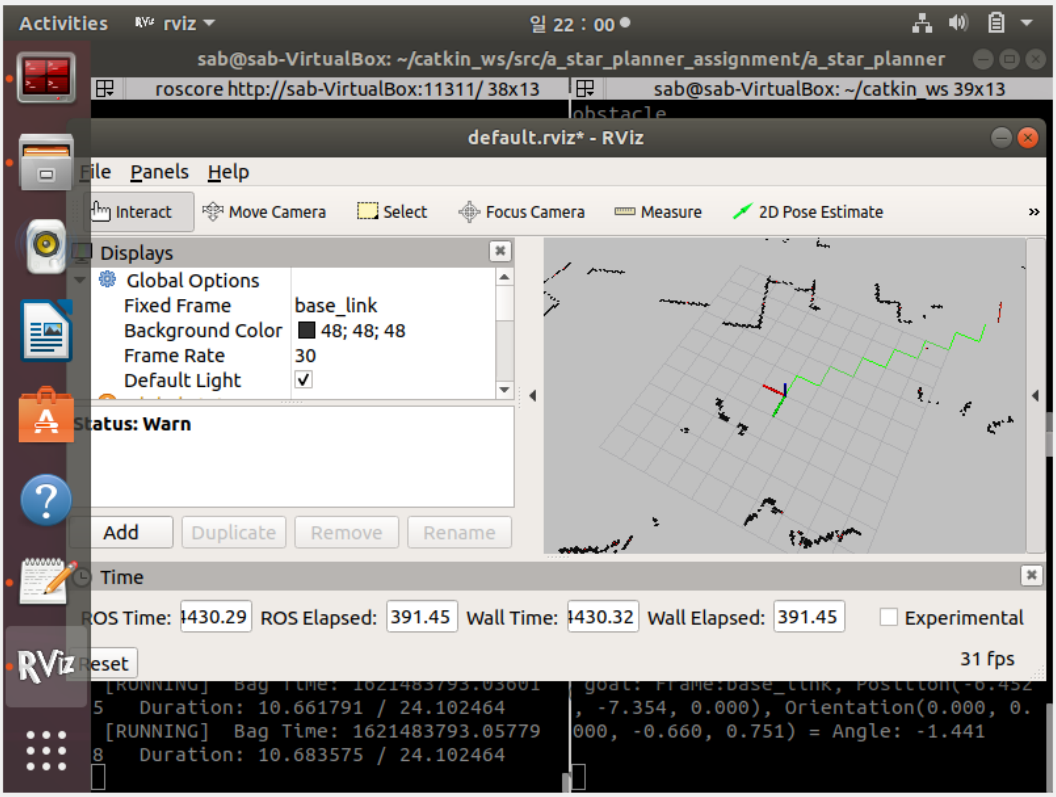
Discussion

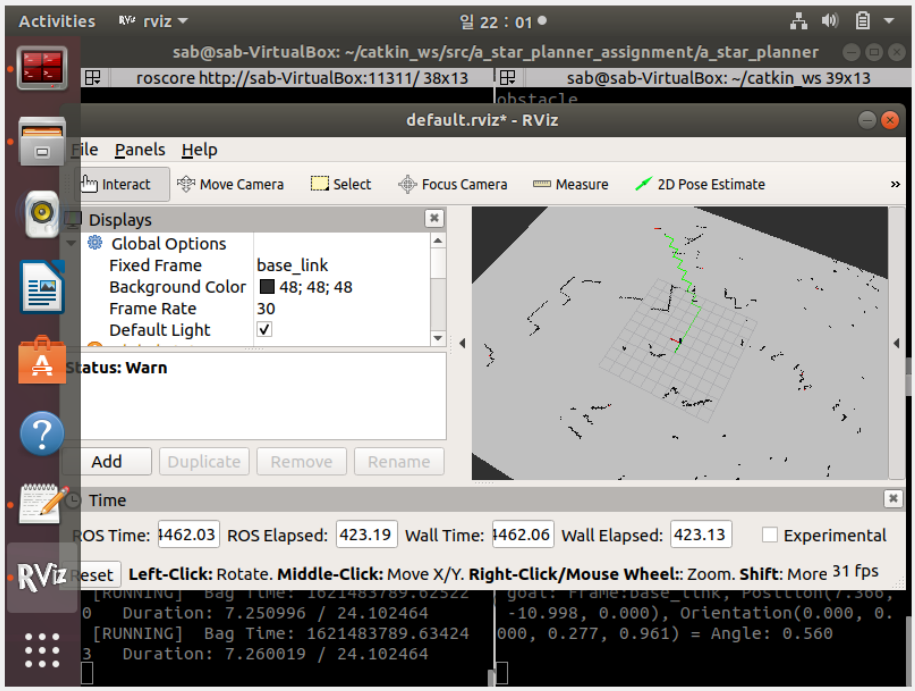
Here is the result of the various input. The challenge in my code was that it was too slow for now reason and I needed to wait for a long time to see the result of the topic.


Activities
rviz
일 21 : 58
sab@sab-VirtualBox: ~/catkin_ws/src/a_star_planner_assignment/a_star_planner
roscore http://sab-VirtualBox:11311/ 38x13
sab@sab-VirtualBox: ~/catkin_ws 39x13
default.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
Displays
Global Options
Fixed Frame
base_link
Background Color
48; 48; 48
Frame Rate
30
Default Light
Status: Warn
Add
Duplicate
Remove
Rename
Time
ROS Time:
ROS Elapsed:
278.69
Wall Time:
Wall Elapsed:
278.63
Experimental
Reset
31 fps
nnersab@sab-VirtualBox:~/catkin_ws/src
/a_star_planner_assignment/a_star_plan
ner$
2.212, 0.000), Orientation(0.000, 0.00
0, 0.000, 1.000) = Angle: 0.000


Activities
rviz
일 21 : 59
sab@sab-VirtualBox: ~/catkin_ws/src/a_star_planner_assignment/a_star_planner
roscore http://sab-VirtualBox:11311/ 38x13
sab@sab-VirtualBox: ~/catkin_ws 39x13
default.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
Displays
Global Options
Fixed Frame
base_link
Background Color
48; 48; 48
Frame Rate
30
Default Light
Status: Warn
Add
Duplicate
Remove
Rename
Time
ROS Time:
366.24
ROS Elapsed:
327.40
Wall Time:
366.27
Wall Elapsed:
327.34
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More
31 fps
Duration: 13.291614 / 24.102464
[RUNNING] Bag Time: 1621483795.68725
Duration: 13.313030 / 24.102464
, -8.401, 0.000), Orientation(0.000, 0.
000, 0.869, -0.496) = Angle: -2.105


Activities
rviz
일 22 : 00
sab@sab-VirtualBox: ~/catkin_ws/src/a_star_planner_assignment/a_star_planner
roscore http://sab-VirtualBox:11311/ 38x13
sab@sab-VirtualBox: ~/catkin_ws 39x13
default.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
Displays
Global Options
Fixed Frame
base_link
Background Color
48; 48; 48
Frame Rate
30
Default Light
Status: Warn
Add
Duplicate
Remove
Rename
Time
ROS Time:
ROS Elapsed:
365.71
Wall Time:
Wall Elapsed:
365.72
Experimental
31 fps
Duration: 16.362227 / 24.102464
[RUNNING] Bag Time: 1621483798.73883
Duration: 16.364612 / 24.102464
, 3.405, 0.000), Orientation(0.000, 0.0
00, 0.869, -0.495) = Angle: -2.105


Activities
rviz
sab@sab-VirtualBox: ~/catkin_ws/src/a_star_planner_assignment/a_star_planner
roscore http://sab-VirtualBox:11311/ 38x13
sab@sab-VirtualBox: ~/catkin_ws 39x13
default.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
Displays
Global Options
Fixed Frame
base_link
Background Color
48; 48; 48
Frame Rate
30
Default Light
Status: Warn
Add
Duplicate
Remove
Rename
Time
ROS Time:
ROS Elapsed:
391.45
Wall Time:
Wall Elapsed:
391.45
Experimental
Reset
31 fps
Duration: 10.661791 / 24.102464
[RUNNING] Bag Time: 1621483793.05779
Duration: 10.683575 / 24.102464
, -7.354, 0.000), Orientation(0.000, 0.
000, -0.660, 0.751) = Angle: -1.441



From various inputs you see that algorithm working in all directions and always reaches the goal points. It also avoids obstacles on its way which is recognized in the last image (it may seem that it goes through it, however there is a gap and it looks at it). So, we may conclude that implementation is well done. The disadvantages as I mentioned earlier is its low speed which can also be the result of slow virtual machine. Another bad thing is that it is not very flexible to the change of the environment as was shown in the lecture video and which again may be encountered by machine's power. Advantage is that its simplicity and workflow that is actually gives a good result at the end.

This algorithm can be well used in the final project as it finds a path from the initial point to the goal by avoiding the obstacles. However, we discussed in the team and get to a conclusion that it is not efficient at all on out case. First of all it changes its direction numerous times during the motion which cannot be easily done by the car at it is not a point in space but rather a big object. Second, the track is ellipse shaped so we cannot just put an initial and final point as thy will be equal and algorithm will just not do anything. Instead we need to run it several times during the motion by setting multiple goal point not that far from the car's position. But, again this needs a complicated model and many decisions where and how often to set this position and so on. Also, its path is very curved which is not that good for the car as it prefers to move in the straight line. Curved path is because algorithm wants to find the shortest distance, but for our car it is better to have large distance but higher speed. However, if the good approach will be made the algorithm can be changed by modifying its cost function in such way that it will prefer the straight-line path and consider the size of the car. Or simply first generate the path and then modify it for our car. Cost function can be also made is such way that the car will decide what way to choose based on its cost that has the speed value, distance, some safety score etc. But this is very

complicated way that wants a lot of unnecessary work. As a result, even if A-star is powerful, it will not be used in our final project.

References

https://www.geeksforgeeks.org/a-search-algorithm/

https://en.wikipedia.org/wiki/A*_search_algorithm

http://docs.ros.org/en/melodic/api/nav_msgs/html/msg/OccupancyGrid.html