A* Planning

Learning objective:

1. Implement A*!

2. Understand how the tuning of cost map will affect the behavior of path planner

Equipment:

1. A personal computer

Mission statement

Given an occupancy grid with cost map, a goal position, can you plan a path between your current position to goal position?

Writing and submission:

1. Submit a working package on Canvas(Alternatively, you may set up your own Github or bitbucket page. Put the page link in your lab report, see below)

2. Write a lab report and submit it on Canvas in IEEE conference proceeding format or ASME conference proceeding format: (2-8 pages).

Grading

Lab report and programming should be done in a pair of 2 or individually. Between groups you are allowed to collectively interpreting material and understanding the course package; sharing code or lab report will be considered violation of course policy.

1. 50%: functional code with adequate annotation

2. 15%: well-organized abstract, introduction with sufficient extra-curriculum research, and method description

3. 35%: Experiment result report, analysis, and discussion.

- You must describe how you get the heuristics
- You must test at least 3 μ values for combining the heuristic and the known cost. Discuss the differences in the result, in the aspect of calculation time (i.e. how many positions you explored before reaching the goal) (important!), path length, path "quality", etc. The test should be done on at least 2 different maps.
- You may additionally vary other parameters, such as the way to generate the cost map, the way to get the heuristics, to check the path generation.

4. 10% bonus max at discretion of the instructor: do something cool?

Part 1: Download new package updates

1. There are minor updates.

You can download the new package here (github). I do not update the previous bitbucket or local file anymore.

The following updates are made:

- line 10 in E160_graphics.py:

Uncomment line 10

```
from path_planner import *
```

- line 66-67 in E160_graphics.py:

Uncomment line 66-67

```
self.path = path_planner(self)
self.canvas.bind("<Button-3>", self.callback_right)
```

- line 25-34 in Path.py:

  A clear path method and save_path method is added:

```
def save_path(self,file_name="Log\path.txt"):
        f = open(file_name, 'a+')
        for v in self.poses:
        # edit this line to have data logging of the data you care
about
                data = [str(v.map_i), str(v.map_j), str(v.theta)]
                f.write('\t'.join(data) + '\n')
        f.close()

def clear_path(self):
        self.poses.clear()
```

- line 102 in path_planner.py

  You should always clear the path before starting path planner. Add the following command:

```
self.path.clear_path()
```

- line 110 in path_planner.py

  You now can save the path as a csv file:

```
self.path.save_path(file_name="Log\path.csv")
```

2. Install missing dependencies using Pip

Part 2: Browse "Path.py"

1. This is the file that defines a path. A Path is nothing but a list of Pose. A Pose is nothing but includes three values: `map_i`, `map_j`, `theta`. Here, `map_i` and `map_j` represents a coordinate on image frame (not world frame). We use image frame simply to make our code be compatible with cost_map

Part 3: Browse "path_planning.py" and do A*!

1. Like `cost_map.py`, this is a class that you will work on. You can change the map scale in E160_graphics.py for debugging purposes. For submission, the map should be 500 x 500 (`self.scale=250` in E160_graphics.py)

2. Use `self.set_goal(x, y, theta)` (line 28 of path_planning.py) to set your goal. The goal is in world frame. The `set_goal` method will automatically set the goal to the map frame (you don't need to anything). **Your planning are all on the map frame.** Currently, your start position is always at center of your image (0,0,0) at your frame. "theta" does not matter at this moment.

3. Your efforts should be on `plan_path(self)` function.

- To add a pose to the path, simply:

```
self.path.add_pose(Pose(map_i, map_j, theta))
```

4. At start, you should be able to see a red path as in Fig. 1. This is simply wrong, of course. It merely show you how we add pose to `self.path`

5. To get the potential map (aka costmap) you acquired in your cost_map.py, you can simply access the variable:

```
self.costmap.costmap
```

If your cost_map.py can generate the cost map correctly, you will get your 500 x 500 cost map at `self.costmap.costmap`

6. Line 104-107 in `plan_path(self)` method demonstrate how to add a position into the path.

If you decide a position on the map should be on your path, call method `self.path.add_pose(p:Pose)`. For example, if you want to add (p[0], p[1], 0) into your path:

```
self.path.add_pose(Pose(map_i=p[0],map_j=p[1],theta=0))
```

The path should be in the map frame.

7. The `bresenham` method is for demonstration purpose. You don't need it at all in this task.

8. You can save the planned path in a .csv file, simply call (e.g. in line 109):

```
self.path.save_path(file_name="Log\path.csv")
```

You can change the file name the file directory.

The file is saved in .csv format with tab as the separator. If you want to view it in Excel, please read this article.

Warning: if you do not change the file name, the newly generated path will be simply appended after the old path. In order to save each path correctly, you need to update the file name each time.

Fig 1. Red path reflect the path we planned. In this case, it is wrong.