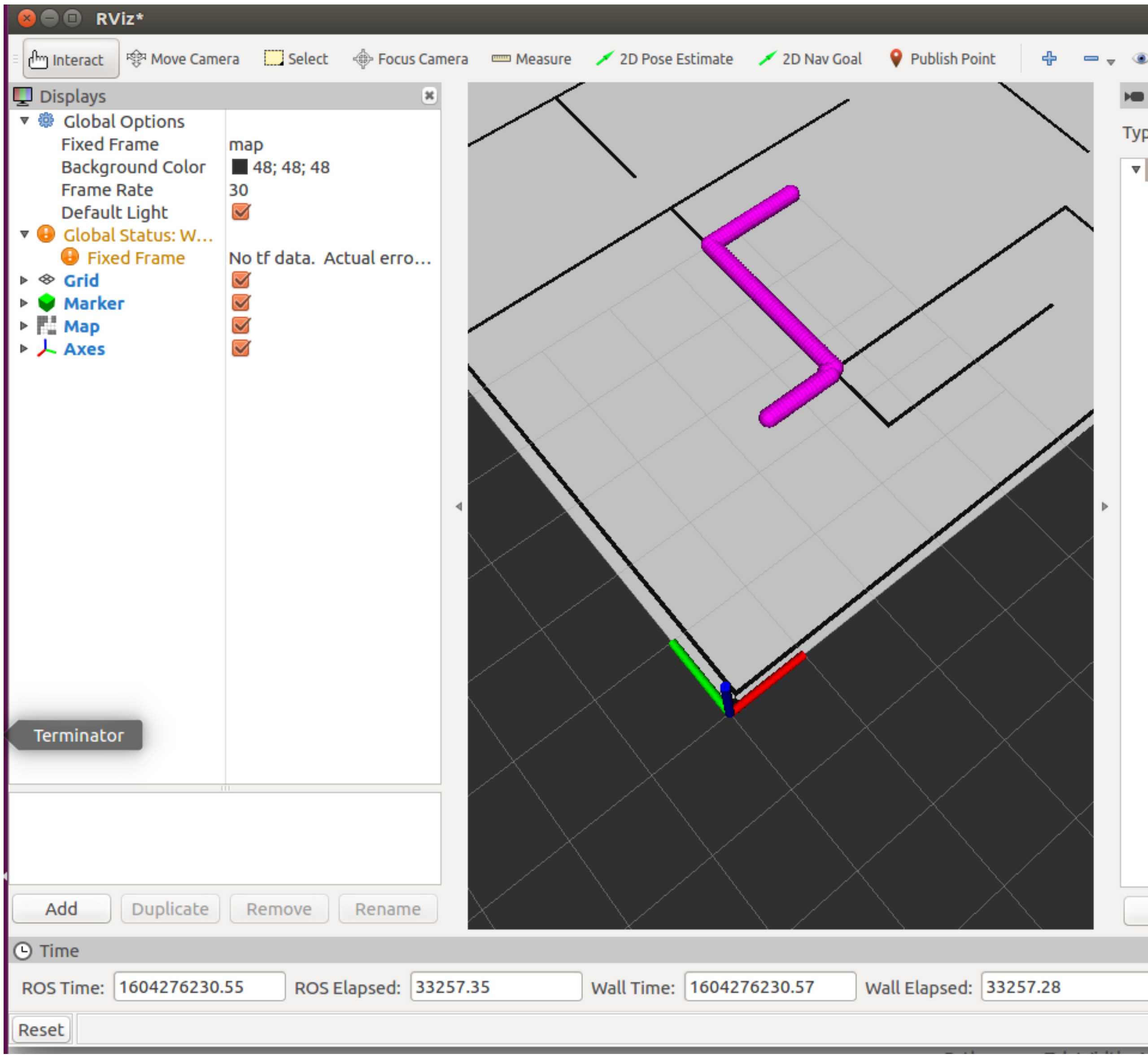
PA 3, CS 81

Garret Andreine

November 2020

1 Description

- Approaching this homework I was initially overwhelmed by the choices of algorithms we discussed in class and the potential to implement them. After some reflecting I decided using A* with a heuristic would be the best choice.
- First thing I did was create two lists, one of the nodes available to me and one of the nodes I have been to, thus allowing me to avoid repetition. I then added the current node the the list of available nodes.
- Going along this list I would sort it in order of distance traveled and distance left to go, going for the optimal path. I would then make sure verify the node hasn't been visited and is not the goal.
- I then collected that node's neighbors in a systematic approach (to limit the amount of turns by favoring directions). As I would get the neighbor nodes, I would add them to a dictionary where their parent node would be the value and the current node would be the key. Allowing me to look up the parent as I went.
- I then continued this process until I found the goal node. Upon finding the goal node I would make a list of the nodes previous nodes from the dictionary until I ended up back at the origin.
- After getting this path from the origin to the goal I would send it to a publishing function where I would publish it in RViz.



Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status: W...
Fixed Frame No tf data. Actual erro...
Grid ✓
Marker ✓
Map ✓
Axes ✓

ROS Time: 1604276230.55 ROS Elapsed: 33257.35 Wall Time: 1604276230.57 Wall Elapsed: 33257.28

2 Evaluation

- Overall I am happy with the implementation.
- I think I am opening myself up to possible errors with:

- Calculating Goal Distance

- Favoring Certain Neighbors First
- Methods of Storing Data

- For Calculating Goal Distance I use the Euclid's Pythagorean theorem to get the most direct path made by the two points. While this is the more direct path it causes the robot to seek to move diagonally, which is not feasible.
- While favoring certain neighbors limits the amount of turns (since I always favor the same directions) I worry it sets me up for a less optimal path than possible since it's a fixed movement path, which is not adaptive to the map.
- In terms of storing data from the program, I don't think I did it the most optimal way. Maybe using a heap data structure would have been better in terms of memory usage and efficiency rather than a list.