

## Lab 4: Safe wandering using path planning algorithms

DATE: Feb 2nd. 2023

Lecturer: *Queenie Qiu, John Raiti*. Student: *Shucheng Guo*

### 1 Learning Objectives

1. Review and modify code to execute different path planning algorithms on a simulated turtlebot3.
2. Compare and contrast the performance of different algorithms (Dijkstra and A\*) as global planners in the ROS navigation stack.
3. Explore challenges and limitations of different planning algorithms to learn the trade-offs of choosing one over another.

### 2 Install and test the global\_planner package

This global\_planner package is one of the packages from the ros navigation stack meta-package. It provides a wrapper on the navigation class that allows to select the type of planning algorithm used for the global plan, using the information from a map. For this lab, we will assume the map has already been created and saved (.pgm and .yaml files). If you wish to try the global planner on a new environment, you can either launch the gmapping demo and save the new map, or add a gmapping node for SLAM in your launch file. Information about the navigation stack and the global planner can be found in the ROS wiki.

1. Download the zip file with the global\_planner package from the course materials on canvas.
2. Extract the zip file and place the global\_planner folder in your src folder in your workspace ( /catkin/src/). You will need to build your package from the /catkin\_ws/ directory. Make sure that all dependencies are installed before building the package:
3. Once the build has successfully finished, source your new workspace (source devel/setup.bash), launch a turtlebot3 gazebo instance and run the test\_planner launch file:

```
$ rosdep install --from-paths src --ignore-src -r -y
$ catkin_make global_planner
```

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
$ roslaunch global_planner test_planner.launch
```

You should be able to see the following topics as active after rostopic list:  
/move\_base/GlobalPlanner/plan

/move\_base/GlobalPlanner/potential

4. Once you find these topics in your list, your package is successfully installed. If you find any trouble with this, contact the instructors or TA.

## 3 Path planning algorithms

### 3.1 Adding the global planner to the turtlebot3 navigation stack.

In previous labs, turtlebot3\_navigation.launch is launched before we used the 2D Nav Goal. If you look into the launch file, you will notice it launches move\_base.launch – it is where to configure the planners in turtlebot's navigation stack. We want to create a new navigation stack by duplicating the launch files and modifying the copied files as needed.

1. Make copies of launch files in the turtlebot3\_navigation package:
  - (a) move\_base.launch → move\_base\_path\_planning.launch: the changes made to this new file are related with adding the global\_planner. Details will be provided later.
  - (b) turtlebot3\_navigation.launch → turtlebot3\_path\_planning.launch: the main difference in this new file is launching the move\_base\_path\_planning.launch file created in step a.

You need to make the change in the "turtlebot3\_path\_planning.launch" file on the move\_base node to include the new move\_base\_path\_planning.launch file
2. Download the parameters .yaml file (global\_planner\_params\_burger.yaml) and place it in the turtlebot3\_navigation/param folder. This file allows you to make changes to the global planner parameters. Refer to the ROS wiki for details on what each parameter does.
3. In the move\_base\_path\_planning.launch file, add a global planner as a parameter of the move\_base node in the launch file:
  - (a) `<param name="base_global_planner" value="global_planner/GlobalPlanner" />`
  - (b) `<rosparam file="$(find turtlebot3_navigation)/param/global_planner_params_burger.yaml" command="load"/>`

```
GlobalPlanner:

# From the wiki.ros.org page on global_planner parameters:
allow_unknown: true # default true
use_dijkstra: true # default true
use_quadratic: true # default true
use_grid_path: false # default false
default_tolerance: 0.01 # default 0.0
visualize_potential: true # default false. For our lab, we will keep this one as true
publish_potential: true # default true
old_navfn_behavior: false # If you want to mirror previous navfn behavior, use defaults for all other parameters
lethal_cost: 253 # default 253
neutral_cost: 50 # default 50
cost_factor: 3 # default 3
orientation_mode: 0 # (None=0, Forward=1, Interpolate=2, ForwardThenInterpolate=3, Backward=4, Leftward=5, Rightward=6)
orientation_window_size: 1 # default 1
```

Figure 1: Global Planner yaml file.

## 3.2 Run the new navigation stack.

1. Launch a gazebo instance of the burger in the turtleworld3\_world:

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

2. Launch the new turtlebot3\_path\_planning.launch. Before sending any Navigation goals, remember to provide a 2DEstimate for the actual position of the robot in RViz. For this lab, we will disable the Global and Local costmaps that come as default in the RViZ configuration for navigation.

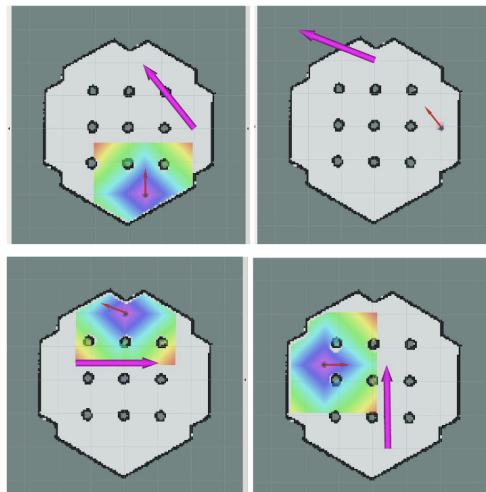
```
$ roslaunch turtlebot3_navigation turtlebot3_path_planning.launch
```

**Tip:** The 2DEstimate point with the RViz GUI provides the robot's initial position and orientation; a large mismatch between the robot's estimates and its actual position may result in a fatal error for the launched file.

3. Customize the RViz visualization components
  - (a) Turn off the Global Map and Local Map, and add by topic the GlobalPlanner/potential topic as another map. Choose costmap to be the color scheme.
  - (b) Change the topic of Planner Plan to GlobalPlanner/plan.
4. Publish four sequential points for the robot to navigate towards( $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_0$ ). You may publish in the /move\_base/goal topic through a node (path\_planning\_goal.py from Canvas), or you may use rostopic pub from the command line. You will need to pay attention to the contents of the message as the pose is given by a position vector and an orientation using quaternions.

**Notice:** The visualization of the potential map doesn't look like what the figure shows.

```
P1: [0, -2, 0] [0, 0, 0.342, 0.939] P2: [2, 0, 0] [0, 0, 0.472, 0.882]
```



```
P3: [0.55, 1.25, 0] [0, 0, -0.707, 0.707] P0: [-2, -0.5, 0] [0, 0, 0, 1]
```

Figure 2: Four sequential points.

5. This sequence you will attempt using two global planners: Dijkstra and A\* (you will need to change the parameters accordingly and report your best performing combination)
  - (a) As it will come to your attention, when using dijkstra the planner works best if you use gradient descent and not a grid; the opposite is true for A\*. Also the value for `neutral_cost` is relevant (lower for Dijkstra, medium for A\*). The option of `use_quadratic` may or may not help. The `cost_factor` might influence how much the local planner will be used.
  - (b) Some parameters can be changed dynamically using the command:

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

### Deliverables:

1. For both global planners, you will also gather the following quantitative information (highly recommend to create a table):
  - (a) For each chunk of the path ( $P_0 \rightarrow P_1$ ,  $P_1 \rightarrow P_2$ ,  $P_2 \rightarrow P_3$ ,  $P_3 \rightarrow P_0$ ), report the travel time (rough estimation is ok.)

NOTICE:

For Dijkstra algorithm, gradient descent was used for optimization.

For A\* algorithm, grid path was used, and `neutral_cost` set to 100.

Path	Algorithm	
	<i>Dijkstra</i>	<i>A* Search</i>
$P_0 \rightarrow P_1$	23s	27s
$P_1 \rightarrow P_2$	16s	25s
$P_2 \rightarrow P_3$	22s	26s
$P_3 \rightarrow P_0$	18s	39s
Total	1m19s	1m57s

- (b) Describe qualitatively whose path length was shorter for A\* and Dijkstra.

**Answer:** The paths of the robot planned by both algorithms are demonstrated in Figure 3. It's quite hard to tell just by eyes, which path is longer. Each algorithm performs differently depending on the condition, in accordance with their strategy. The A\* path made detours in the first chunk, but walked in straighter lines in the next two.

- (c) What did you observe from the potential map?

**Answer:** The potentials of the path calculated by both algorithms are demonstrated in Figure 4. It's obvious that the colored areas are larger along the path of the Dijkstra one than the A\* one. It means that A\* algorithm considers fewer potential paths, or only expands certain nodes closer to the goal, while Dijkstra explores in all directions.

- (d) Discuss the advantages and disadvantages of the two path planning algorithm.

DIJKSTRA: a special case for A\* without heuristics

**Pros:**

- finds the optimal path by exploring all possibilities
- requires only source and goal node
- less memory usage

**Cons:**

- more operation time and resource consumption in general
- assumes a finite graph
- works only with non-negative edge costs or weights

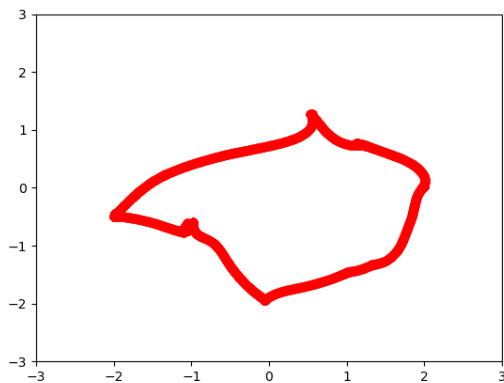
A\* SEARCH: greedy best-first search with two cost functions

**Pros:**

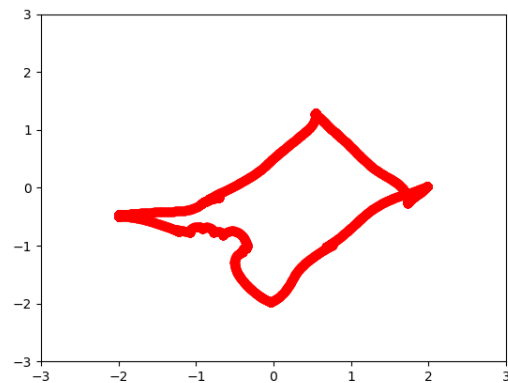
- achieves a trade-off between memory, time, and length
- explores the fewest number of nodes and paths
- less time complexity in theory

**Cons:**

- efficiency relies heavily on a good heuristic function
- finds optimal path only with admissible heuristic function
- more space complexity in theory



(a) Dijkstra



(b) A\* Search

Figure 3: Plotting robot's paths planned by different algorithms.

2. Use your pillar model (created in Lab1), and add it as an obstacle in the turtlebot3\_world. Turn on the Local Map visualization in Rviz. Attempt to send a 2DNavigationGoal where the robot has to circumvent the new obstacle.
  - (a) Report your choice for obstacle location by taking a screenshot, and your robot's initial position as well as the selected goal position and orientation. (Tip: you may want to use the topic /move\_base/goal, and echo it, to obtain position and orientation vectors)

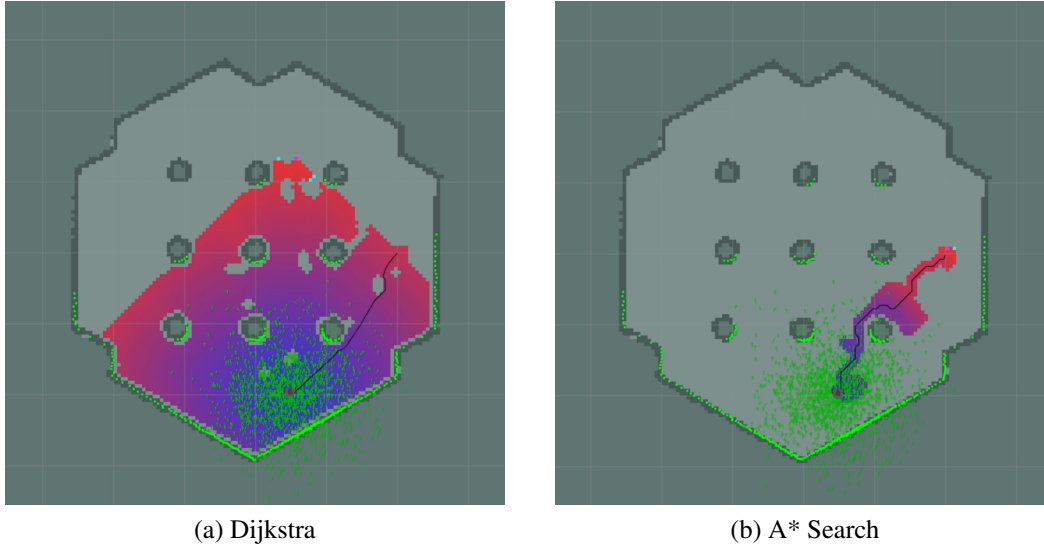


Figure 4: Calculating potentials by different algorithms.

**Answer:** The obstacle, a pillar of which the radius is  $0.2m$ , is placed right to  $P_2$ , at the point  $(1.68, -0.68, 0.25)$ , as shown in Figure 5. The initial and goal nodes are set to  $P_1$  and  $P_2$  respectively for easier comparison. The obstacle is set such that it blocks the optimal path from  $P_1$  to  $P_2$  previously computed by the algorithm. It has to dynamically replan a new path that avoids the obstacle.

$P_1: (0, -2, 0) (0, 0, 0.342, 0.939);$

$P_2: (2, 0, 0) (0, 0, 0.472, 0.882).$

(b) Describe the actions of the robot:

- i. Does the robot reach its intended goal in A\* and Dijkstra?

**Answer:** For the aforementioned setting, in which the obstacle has a medium radius, the robot almost always reaches the goal. There were a few edge cases, where the robot was stuck in a narrow aisle between pillars.

Doubling the radius of the obstacle, however, brings the success rate down to zero. With a larger pillar, the original path is completely blocked and a major retreat and detour is required. The robot under such algorithms is obviously not prepared.

- ii. What did you observe from the global planner and local planner?

**Answer:** The indicator for the global planner is a thin, black, and solid black line between the start and end nodes, while that for the local planner is a much shorter line in a lighter color that constantly changes.

Although both indicators change, the local planner does it at a much faster speed. Based on their definition, it can be deduced that the global planner calculates the

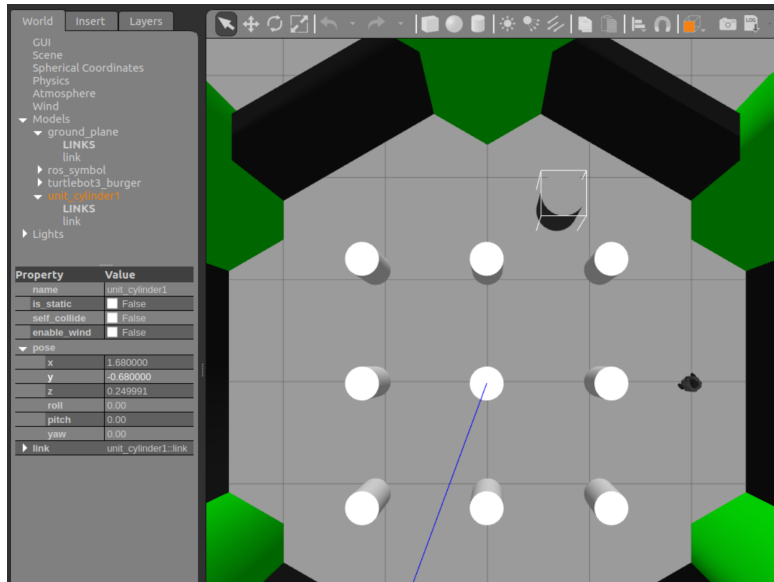


Figure 5: Position of pillar obstacle.

full path prior to operation and dynamically adjusts path during the process; the local planner generally follows the global planner, but modifies the full path if obstacles are detected along the way to prevent collisions.

Specifically, the global planner subscribes to the map, current position, goal position, and publishes a path between both positions; the local planner subscribes to the map and the path planned by global planner, and publishes command velocities for the robot to best accomplish the goal. Should there be unexpected objects on the path not detected or considered by global planner, the local planner would dynamically adjust `cmd_vel` for the robot to avoid them.

## 4 Path planning with the physical robot

This portion of the lab assignment will entail repeating the same process for testing the two global planners, now using the GIX map. You have already completed a map for this environment. If you do not have the .pgm and .yaml files associated with the GIX map you will have to run the gmapping demo and teleoperate the robot around the map before getting started.

We will have to repeat the connection instructions for the physical robot. Changes to be made in the environment variables and the `/.bashrc` file.

Instead of providing a sequence of 4 points as navigation goals, the physical implementation will include only 2 points, with two paths: One from  $P_0 \rightarrow P_1$ , and another  $P_1 \rightarrow P_0$ .  $P_0$  will be your starting position (marked with a black X),  $P_1$  will be marked by a blue X.

### Deliverables:

1. Using the global\_planner with both algorithms (Dijkstra and A\*) to report:
  - (a) the travel time (rough estimation is ok.)
  - (b) If any collisions occur, note them and comment on possible causes of the collisions.
2. Reset the robot to the initial position  $P_0$ , and then add an obstacle in the map environment. Attempt the navigation to  $P_1$  and comment on the execution using both planners:
  - (a) Does the robot reach its intended goal in A\* and Dijkstra?
  - (b) What did you observe from the global planner and local planner?