Lab 4: Navigation

ME 597: Autonomous Systems

I. Introduction

Mobile robots are used in different environments to perform a wide range of tasks such as cleaning, patrolling, material handling, rescue operations, etc. These tasks, require the robots to move freely and autonomously through either static or dynamic environments. Therefore, Navigation is a crucial aspect for robots.

Navigation allows robots to move through a predefined path, to adjust its trajectory due to changes in the environment, or to correct motion online due to measurement errors. As part of navigation, it is also important to consider extreme cases in which the robot gets lost or crashes into something, by implementing recovery strategies that bring the robot back to a safe known state.

This lab will show you some of the available tools in ROS to perform the Navigation task such as 1) Mapping, 2) Localization, 3) Planning and 4) Execution of a trajectory using the ROS Navigation Stack Fig.1, which implements the nodes and algorithms to solve the planning, mapping, localization, and recovery.

The ROS Navigation stack outputs `Twist` data through the `\cmd_vel` topic based on odometry, sensor data, and a goal pose. Although the planning and the localization algorithms are already part of the stack, it is possible to tune/modify the behavior by means of the ROS parameter server via `*.yaml` files.

Finally you will implement your own path planning and path follower to better understand the aforementioned navigation concepts.

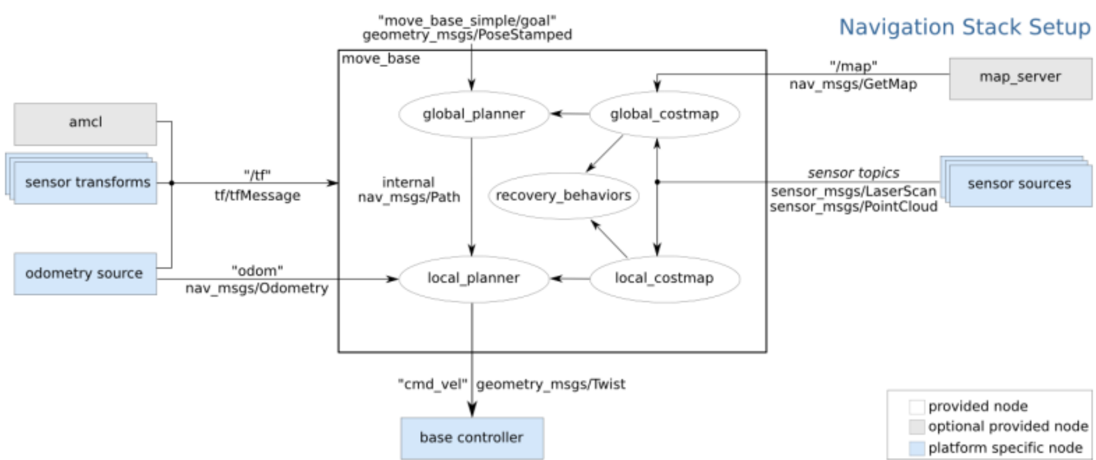


Fig. 1: ROS Navigation Stack

II. Assignment

A. ROS Navigation Stack

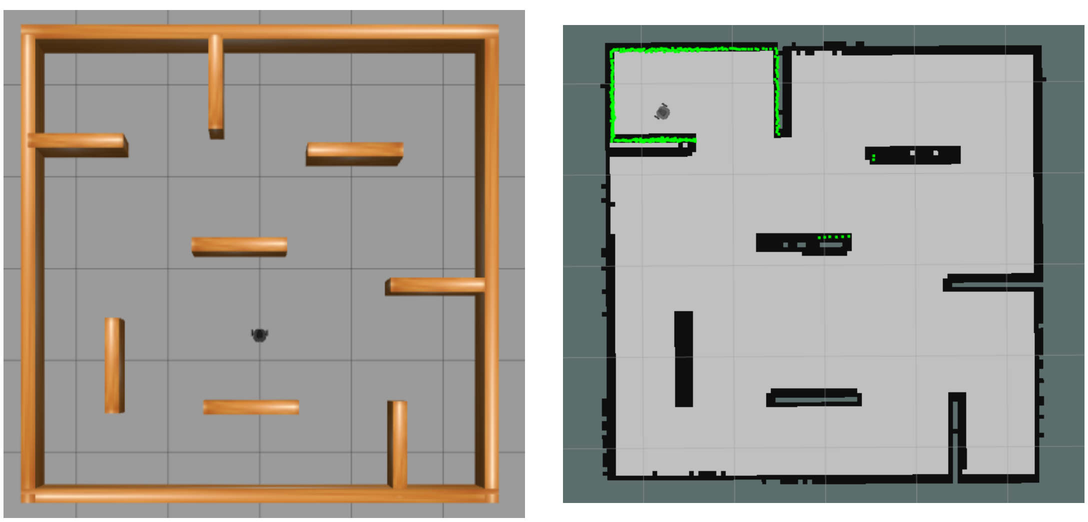
This assignment will help you understand and use ROS Navigation Stack. You need to create a map and the following assignments will need the same map file.

1) `map_server` and `gmapping` are some of the required tools to complete this activity, for those using a virtual machine, these tools should be already preinstalled. For those with Ubuntu natively installed, you might need to run the following commands.

```
sudo apt install ros-noetic-gmapping
sudo apt install ros-noetic-map-server
sudo apt install ros-noetic-amcl
sudo apt install ros-noetic-move-base
sudo apt install ros-noetic-dwa-local-planner
sudo apt install ros-noetic-turtlebot3
sudo apt install ros-noetic-turtlebot3-simulations
```

2) Follow the mapping tutorial to learn how to create and save a map and navigation tutorial to learn how to import the map, create a navigation goal, and use `move_base` to move to the destination.
3) In `aut_sys_lab` create a package called `lab_4_pkg`. In the package, create four folders, `launch`, `worlds`, `maps`, and `script`.
4) Download lab_4_slam.launch and save it in `launch`. Download turtlebot3_plaza.world and save it in `worlds`. Compile the package.
5) Launch simulation environment

```
roslaunch lab_4_pkg lab_4_slam.launch
```



(a) (b)

Fig. 2: (a) simulation environment (b) map

6) Open a new terminal and start to create a map for the environment. This will launch RVIz and it can visualize the map you are creating.

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=
    gmapping
```

7) Open a new terminal and run `turtlebot3_teleop_key.launch` to use the keyboard to control the robot to scan the whole environemnt.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

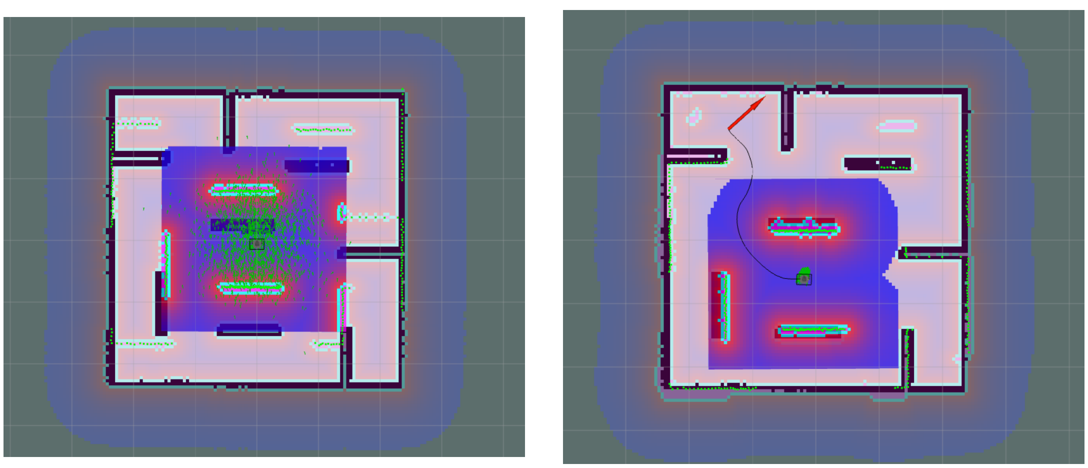
8) Save the map in `maps` after the map is created successfully.

```
cd catkin_ws/src/aut_sys_lab/lab_4_pkg/maps/
rosrun map_server map_saver -f map
```

9) Shut down all the launch files. Launch `lab_4_slam.launch` to reset the enviroment.
10) Run the following command to read the map and start the ROS navigation package.

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch
   map_file:=$HOME/catkin_ws/src/aut_sys_lab/lab_4_pkg/maps/map.
   yaml
```

11) Correct the initial pose of the robot.
12) Create a `2D Nav Goal` and see if the robot can move to the destination.



(a) (b)

Fig. 3: (a) Incorrect initial pose (b) 2D Nav Goal

13) At this point, you should have a couple of map files, one containing an image of the map(`*.pgm`) and another one containing meta data of such map(`*.yaml`). You will need these two files in order to implement your own A* algorithm. Go to the course site on Brightspace, and download the jupyter notebook. You will need to upload this file into Google Colab to work with it. Go through the jupyter notebook and implement the A* algorithm for the map obtained during previous tasks with the navigation stack.
14) For this part of the lab you will need to integrate your A* algorithm into the Gazebo environment. For that it is necessary that you figure out how:
    - RVIZ passes waypoints to the turtlebot.
    - How to map waypoints from the map frame (pixels) to the global frame (in meters), and viceversa.

15) For this task we have created the skeleton of a node that receives a goal pose. We have also included the topic from which you will be getting the position of the vehicle. Additionally a launch file has been created to run the required nodes to do this implementation.
16) You will find the node file `navigation.py` and the launch file `lab_4_navigation.launch` on Brightspace. The node file has placeholders wherever you need to develop code.
17) Your job here is to get the goal pose, use the algorithm previously created, and implement a path follower to drive the vehicle through the path. You have been provided with the required topics to complete tthe task so that you only worry about the logic of the path planning and path follower.
18) It is assumed that the path is free of obstacles, in case your path is no collision free during the testing on gazebo, you might need to increment the safe margin with respect to the walls or improve your map accuracy.

III. Deliverables

1) The `lab_4_pkg` package with your solution.
2) The jupyter notebook including your solution for the A* algorithm.

IV. Rubric

- 50 pts - Mapping and A* Algorithm.
  - 10 pts - Successful map creation.
  - 40 pts - Correct implementation of the A* algorithm
- 50 pts - Custom Navigation.
  - 20 pts - Proper generation of the A* algorithm.
  - 20 pts - Proper implementation of the path follower.
  - 10 pts - No collisions during testing.