COMP-4510-project-2

Table of Contents

- catkin_ws
  - The main workspace for the project
- frontier-exploration
  - The primary package including nodes and launch files for the project
- frames.pdf
  - A pdf displaying the transform view frame connections
- waffle_tf_listener.py
  - Python transformation listener file to display the current position of the robots base_footprint in reference to the map.
- moveActionClient.py
  - Python action client to move the robot with respect to its base_footprint reference frame.
  - Translation parameters (goals) are entered in the terminal, see setup files tutorial.
- RvizProjectTwoConfig.rviz
  - setup config file for RviZ, includes robot camera and global/local path markers.
- frontiers_finder.py
  - Subscribes to /map to take an occupancy grid and locate candidates for frontiers (locations where unoccupied known space meets unknown space).
  - Publishes an occupancy map showing the location of each frontier
  - Publishes color coordinated points to display the distinct clusters of segmented frontiers and their centroids.
- util.py
  - utility function for frontiers_finder.py contains morphological functions for dilation, erosion and line detection on images (occupancy grids)
  - contains connected component analysis algorithms to detect continuous or semi-continuos regions to binary occupancy grids.
- Video of setup files at [Watch the video]
- Video of frontier location algorithm (part 2) here [Watch the video for part 2]

Instructions for running part 2

This section covers how to display the segmented frontiers in the RviZ simulation. First run the setup files, each in a new terminal

```
$ roslaunch turtlebot3_gazebo turtlebot3_stage_4.launch
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

Then run the package launch file (you may or may not need to resource the

terminal.)

```
$ cd catkin_ws
$ source devel/setup.bash
$ roslaunch frontier-exploration turtlebot3_navigation.launch
```

This should open a Gazebo and RviZ window.
It is important that the RviZ window config loads correctly,
the displays tab should contain,

- Map/local path/ global path from the start up files section (PART 1)
- frontiers_map
  - Subscribed to /frontiers_map
  - Displays the occupancy grid for the group of all frontiers
- MarkerArray
  - Subscribed to /visualization_marker_array
  - Displays color coordinated dotes representing distinct clusters of frontiers and their centroids. If the RviZ config does not load properly, try opening the config directly from,

```
frontier-exploration/RvizProkectTwoConfig.rviz
```

Next, run the frontiers identification script

```
$ rosrun frontier-exploration frontiers_finder.py
```

(you may need to resource the terminal)
This will automatically detect and segment the frontiers currently visible by the the robot.
We can watch the frontiers update by giving the robot a movement command.

```
$ rosrun frontier-exploration moveActionClient.py -x -1 -y 1
```

You can also input goals in the RviZ window with the 2D Nav Goal button on the top panel.

Running setup files

In their own terminal run the following

```
$ roslaunch turtlebot3_gazebo turtlebot3_stage_4.launch
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

This should open a Gazebo and RviZ window.
The RviZ window config should include a pre-saved camera display and global/local path markers.
If the RviZ config does not laod, it can be added by opening the config.

```
RvizProkectTwoConfig.rviz
```

Next, in a new terminal run,

```
$ roslaunch frontier-exploration turtlebot3_navigation.launch
```

If this command errors you may need to resource the terminal

```
$ cd catkin_ws
$ source devel/setup.bash
$ roslaunch frontier-exploration turtlebot3_navigation.launch
```

The pdf file frames.pdf displays a tf tree of the objects on the module.
To generate a new pdf, run,

```
$ cd catkin_ws
$ sudo apt install ros-noetic-tf2-tools
$ rosrun tf2_tools view_frames.py
```

Next, we can see the position of the robot with,

```
$ rosrun tf tf_echo /map /base_footprint
```

Another way to display the position of the robot is with a listener script in terminal

```
$ rosrun frontier-exploration waffle_tf_listener.py
```

This will display the current position of the base of the robot with respect to the map.
Next, we want to issue a command to the robot to move.
This can be done with the moveActionClient.
Running

```
$ rosrun frontier-exploration moveActionClient.py -x <goal in x> -y <goal in y>
```

will translate the position of the robots base frame by (x,y) units.
Note: this translation is with respect to the robots frame not the map so for example translating (x=1,y=1) will move the robot right and up one unit. (instead of moving the coordinate (1,1) on the map.)
Here are a few examples to run.

```
$ rosrun frontier-exploration moveActionClient.py -x -1 -y 1
$ rosrun frontier-exploration moveActionClient.py -x -1 -y -1
```

You can also issue commands without specifying the argument, in the order (x,y)

```
$ rosrun frontier-exploration moveActionClient.py -1 1
$ rosrun frontier-exploration moveActionClient.py -1 -1
```

This concludes the setup for the project.

PART 2 running frontiers finding algorithm

This section covers how to display the segmented frontiers in the RviZ simulation.
First run the setup files, each in a new terminal

```
$ roslaunch turtlebot3_gazebo turtlebot3_stage_4.launch
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

Then run the package launch file (you may or may not need to resource the terminal.)

```
$ cd catkin_ws
$ source devel/setup.bash
$ roslaunch frontier-exploration turtlebot3_navigation.launch
```

This should open a Gazebo and RviZ window.
It is important that the RviZ window config loads correctly,
the displays tab should contain,
- Map/local path/ global path from the start up files section (PART 1) - frontiers_map - Subscribed to /frontiers_map - Displays the occupancy grid for the group of all frontiers - MarkerArray - Subscribed to /visualization_marker_array - Displays color coordinated dotes representing distinct clusters of frontiers and their centroids.

If the RviZ config does not load properly, try opening the config directly from,

```
frontier-exploration/RvizProkectTwoConfig.rviz
```

Next, run the frontiers identification script

```
$ rosrun frontier-exploration frontiers_finder.py
```

(you may need to resource the terminal)
This will automatically detect and segment the frontiers currently visible by the the robot.
We can watch the frontiers update by giving the robot a movement command.

```
$ rosrun frontier-exploration moveActionClient.py -x -1 -y 1
```

You can also input goals in the RviZ window with the 2D Nav Goal button on the top panel.

Troubleshooting

- Cannot find package error
  - This is a problem that is liklely caused by the terminal not being sourced correctly. Which can be resourced by

    ```
    $ cd catkin_ws
    $ source devel/setup.bash
    ```

- Problems with tf package for view_frames.py
  - many of the original tf functions are depriciated and so tf2 has been used in exchange.

```
$ sudo apt install ros-noetic-tf2-tools
$ rosrun tf2_tools view_frames.py
```

- If the application is having trouble connecting to the robot try running the following to change the environment to use the waffle_pi robot. `console` `$ export TURTLEBOT3_MODEL=waffle_pi`
- If anything is added to the package, re-make the workspace `console $ catkin_make $ cd project/catkin_ws $ source devel/setup.bash`
- If you want to ensure a publisher is publishing `console rostopic echo <topic name>`