Assignment 1: ROS, Gazebo, and Fetch

January 23, 2020

This is our first assignment which helps you get started and familiar with ROS and Gazebo, while running the Fetch robot in simulation. Follow the instructions below to install and run several scripts. At several points you will be asked to take screenshots and videos of your results and to receive full credits you need to upload a zip/tar.gz file containing the 4 images and 3 short videos, less than 30 seconds each (only keep the most related part to requirements) and upload them through Canvas.

If your computer is already equipped with:

- ROS indigo and Ubuntu 14.04
- ROS Medolic and Ubuntu 18.04.

Feel free to your own device. If you do not have any of these two systems, we will provide a Docker instance with ROS indigo and Ubuntu 14.04 (A quick tutorial handout about Docker will be given to help you set up). This homework is based on two main tutorials, ROS general topics can be found at `http://wiki.ros.org/ROS/Tutorials`, while the detailed tutorial by Fetch Robotics on which this assignment is based can be found at `http://docs.fetchrobotics.com/gazebo.html`.

Part I
Required Installation

For this course, we will use the Fetch Robot as our simulated mobile manipulator. The Fetch robot was developed under ROS indigo and Ubuntu 14.04, but has an stable and working version for ROS melodic and Ubuntu 18.04.

1 Install ROS

If you have a computer with Ubuntu 14.04 or 18.04, you should be able to install and run all the required elements on your machine. If you don't have a machine with any of this systems, we will provide a docker image with Ubuntu 14.04 and ROS indigo. The instructions to install ROS in Ubuntu 14.04 can be found at

- `http://wiki.ros.org/indigo/Installation`

while the instructions for Ubuntu 18.04 can be found at

- `http://wiki.ros.org/melodic/Installation`.

For this course, we will use the ROS indigo distribution as our base line, there will be several parts throughout the course, where we will ask you to install ROS packages such as

```
$ sudo apt-get install ros-indigo-fetch-gazebo-demo
```

It can be seen that the name of the package has the following format: ros-$distro$-package_name. If you are using ROS melodic distribution, you will have to change the command too

```
$ sudo apt-get install ros-melodic-fetch-gazebo-demo
```

2 Install the Fetch Robot

Once ROS indigo and Ubuntu 14.04 have been installed for you. The next step is to install the Fetch simulator. Open a command-line shell and type the following commands

```
$ sudo apt-get update
$ sudo apt-get install ros-indigo-fetch-gazebo-demo
```

This installs the package that contains some interesting demos.

Part II
Simulation

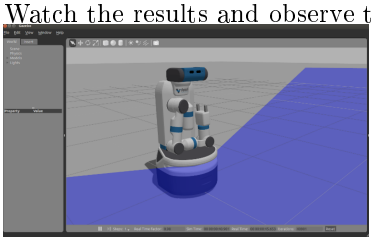
3 Start simulation in Gazebo

Start the simulator and generate a Fetch robot in simulation environment, by

```
$ roslaunch fetch_gazebo simulation.launch
```

The first time you launch this, it will take some time to obtain the model from website. Hence, if the process fails or takes a long time, press Ctrl+C to quit and relaunch again.

Watch the results and observe the Fetch robot.



4 Visualize in Rviz

Open a new shell (don't close the previous one) to visualize the Fetch robot via the ROS visualization application "rviz". You can manually set up your rviz visualization or use a predefined configuration file.

4.1 Manually set up

Type the following command line to start rviz,

```
$ rosrun rviz rviz
```

Probably you can't see anything in rviz at the beginning.

1. Click 'Add' at bottom left corner -> Under 'By display type' tab choose 'RobotModel', add it -> on left panel of rviz, change the 'Fixed Frame' from 'map' to 'base_link', you should see the Fetch robot in rviz now.

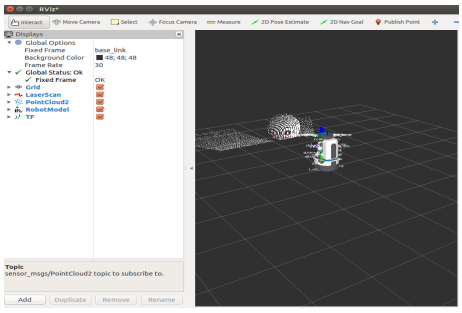
2. Add the 'TF' to visualize transformation frame, 'LaserScan' for visualizing laser , and 'PointCloud2' to generate point cloud of environment.

3. Click the left triangular button near 'LaserScan', then click the area right to the 'Topic' item, choose '/base_scan' topic to visualize the laser scan.

4. Similarly, change the topic of 'PointCloud2' as '/head_camera/depth_downsample/points'. You should see a grey area generated in front of Fetch

5. Try to add some objects into your simulation world by clicking different icons on top panel in Gazebo. Left click the icon, for example a sphere, and left click the area in front of Fetch to add that 3D sphere into your simulation world.

6. The result may be similar like this(with a sphere in front of Fetch). Take a screenshot of your simulation.



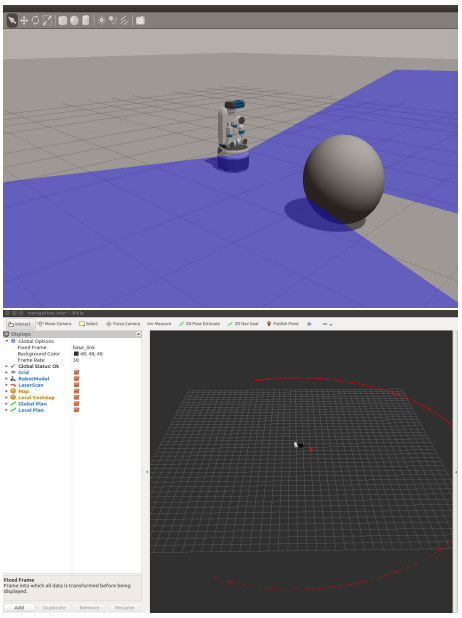
- Exercise 1: Take a screen shot of Fetch robot with TF, laser scan(red) and point clouds(white).

4.2 Use predefined configuration file

Instead of doing this manually, you can load default .rviz configuration file for Fetch. Use roscd again to navigate to the fetch_navigation/config directory and load the rviz file.

```
$ roscd fetch_navigation/config
$ rviz -d navigation.rviz
```

On the left panel, change the 'Fixed Frame' from 'map' to 'base_link', follow the instructions in 4.1 to change topics for visualization, and take a screen shot of your simulation. This time you will have 2 extra items added to the left panel, 'Map' and 'Local Costmap', we will skip this part and discuss it in navigation part.



- Exercise 2: Take a screen shot of Fetch robot with your simulation world and corresponding rviz window[default configuration file].

5 Running mobile manipulation demo

This is a demo showing how navigation, perception and Moveit! work together to achieve some complex task. Use the following lines to run this demo.

- Open a new shell (make sure you already quit the previous ones),

```
$ roslaunch fetch_gazebo playground.launch
```

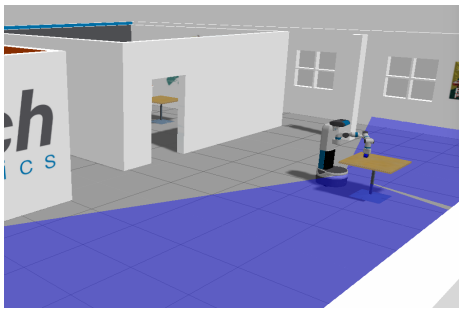
- To start the simulation in gazebo. Open a new shell and use the following line to run the demo. You could see the Fetch robot search for object and navigate itself towards the table, it will stop in front of the blue cube and try to pick it up.

  ```
  $ roslaunch fetch_gazebo_demo demo.launch
  ```

- Open a new shell, you can use rviz to visualize this process. Use the same command to open the window and change the corresponding topics as previous steps.

  ```
  $ rosrun rviz rviz
  ```

- Exercise 3: Take a screen shot of Fetch robot in gazebo, your simulation might look like the following picture.



Part III
Create our own Project

Until now, we have just executed prebuilt programs to see and interact with the basic tools of Gazebo and Rviz. Now, we will create our own project.

6 Creating a catkin workspace

The ***catkin*** is the official build system, which is responsible for generating targets, including libraries, executable programs etc. The first step is to create a catkin ***workspace***. Open a shell, type following command,

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin_make
```

On the screen, you should see output like this if everything goes well (might be slightly different based on your virtual machine):

```
...
catkin 0.6.19
BUILD_SHARED_LIBS is on
Configuring done
Generating done
Build files have been written to: /home/student/catkin_ws/build
```

When it is done, you should see new folders *devel* and *build* in the *catkin_ws* directory. Take a look at `http://wiki.ros.org/catkin/Tutorials/create_a_workspace` if you need more details.

7 Creating ROS packages

7.1 Creating a new package

ROS uses packages to organizes functionality. All your files for a ROS program should be included in a package, including cpp/python files, launch files, configuration files etc. Under the catkin workspace directory *catkin_ws* there are three subdirectories: *devel*, *src* and *build*. To create a new catkin package, we use

```
catkin_create_pkg <package_name> <package_dependecies>
```

where *package_name* is the name of the package that you want to create, and *package_dependecies* are those packages which your package depends on. For more details about package creation, see `http://wiki.ros.org/ROS/Tutorials/CreatingPackage`

For this assignment, use the following command,

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg assignment1 std_msgs roscpp rospy
```

7.2 Compiling a package

You need to compile the src directory before using the package you create. This command should be executed under catkin_ws directory.

```
$ cd ~/catkin_ws
$ catkin_make
```

This updated the build products in the *build* directory to now include the assignment1 package.

7.3 Organizing your package

If you go to your newly created package

```
$ cd ~/catkin_ws/src/assignment1
```

You will have the basic structure of your package. You should now have 2 folders: *src*, *include*, and two files: *CMakeLists.txt* and *package.xml*. This is the fundamental structure of a ROS package. The *src* and *include* folder contain all source files, such as python and cpp files. The file *CMakeLists.txt* contains *cmake* rules for compilation, and *package.xml* provides information about the package and its dependencies.

ROS has several different elements such as configuration files, launch files, rosbag files between others. It is customary to create a separate folder for each type of file.

In your package,

7.3.1 Launch files

Launch files, are a script that will point to one or several other scripts to start and initialize. First we need to create a new folder *launch* in the package you created.

```
$ cd ~/catkin_ws/src/assignment1
$ mkdir launch
```

Let us first consider a simple example of what a launch file looks like. A good example is the file *build_map.launch* in the fetch_navigation package, which is available at `https://github.com/fetchrobotics/fetch_ros/blob/melodic-devel/fetch_navigation/launch/build_map.launch`

```
<launch>
<node pkg="slam_karto" type="slam_karto"
name="slam_karto" output="screen">
<remap from ="scan" to="base_scan"/>
</node>
</launch>
```

We only focus on the <launch> and <node> tags in this file. Every launch file starts with a <launch> tag, and inside the launch tag we add other tags and parameters. In the <node> tag, we usually set values for four important parameters. *pkg* is the name of the package that we want to execute, *type* is the program file we want to execute, *name* is the name of node we will launch, and *output* specifies where we want to print the output. To explain it in a more concise way,

```
<node pkg="PKG_NAME" type="PROG_FILE" name="NODE_NAME"
output="WHERE_TO_PRINT">
```

Hence, the example launch file above launches a node called *slam_karto*, executes the *slam_karto.cpp* program in a package called *slam_karto*, and outputs are printed on screen. For more details about slam_karto, see `https://github.com/ros-perception/slam_karto/tree/indigo-devel/src`

7.3.2 Configuration files

Previously, we tried to set up rviz manually, and we also tried to load the default rviz configuration file for visualization. Setting up manually every time we launch a new rviz window is time-consuming, and the default rviz configuration may not visualize the topics we want. One of the ways to handle this, is to set up and save your own rviz configuration file and load it when you run rviz.

- First create a *config* folder in your assignment1 package. We will save your rviz file in that folder.

  ```
  $ cd ~/catkin_ws/src/assignment1
  $ mkdir config
  ```

- Then, start the Fetch simulator using command,

  ```
  $ roslaunch fetch_gazebo playground.launch
  ```

- Second, run build_map.launch in fetch_navigation package,

  ```
  $ roslaunch fetch_navigation build_map.launch
  ```

- Third, run rviz and set up manually, in addition to the components we mentioned in last assignment, we also need 'map', and 'image' if you are interestd seeing pictures taken by Fetch's head camera. Your rviz window should look similar to this (feel free to add other components you want, but all components mentioned above are required), for each component, the corresponding topics are shown below.

  ```
  $ rosrun rviz rviz
  ```

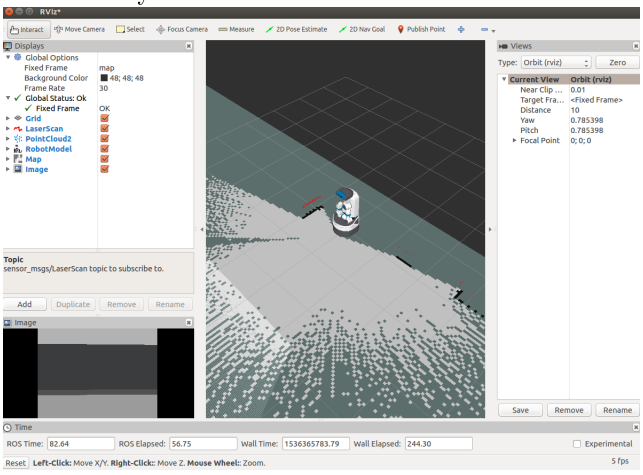
1. 'Global Options' -> 'Fixed Frame' -> map
2. 'LaserScan' -> 'Topic' -> /base_scan
3. 'PointCloud2' -> 'Topic' ->/head_camera/depth_downsample/points
4. 'Map' -> 'Topic' -> /map
5. 'Image' -> Image topic -> 'head_camera/rgb/image_raw'

- Click the top left menu, click -> [File] -> [Save Config As] -> /student/catkin_ws/src/assignment1/config -> name it as assignment1.rviz
- Quit rviz (DO NOT quit your gazebo simulator), let's try to set up rviz by using our own configuration file. Open a new shell, use the following command (remember to change the directory if you are using your own computer)

  ```
  $ rosrun rviz rviz -d ~/catkin_ws/src/assignment1/config/assignment1.rviz
  ```

- Make sure your gazebo simulator and build_map.launch are still running when you load the rviz file, otherwise there will be status errors in rviz.

Your result may look like this:



7.4 Nodes, topics, message, subscriber & publisher

For other basic concepts of ROS, please go through tutorials on website, which provides a detailed explanation of each concept and their functions. This assignment will not cover these concepts.`http://wiki.ros.org/ROS/Tutorials`

Part IV
Mapping

8 Keyboard Teleop

To build the map, we need to move the Fetch robot around the room to 'see' the layout of the room. Here comes the most exciting part! We can use our keyboard to control Fetch robot to hang around room and see the details of room. This is where the ROS structure comes to play. When we launched the simulated robot, it opens Gazebo and creates a Fetch robot. This robot will subscribe to topics such as, velocity commands which he can execute by turning its wheels and will publish topics such as, laser _scan which other script can use to reconstruct the world (mapping) around the robot.

8.1 Teleop your Fetch robot

To do this, first look at tutorial about Robot Teleop on `http://docs.fetchrobotics.com/teleop.html` and follow the steps below to create a launch file for tele-

operation:

- Under ~/catkin_ws/src/assignment1/src directory, create a python file named keyboard_teleop.py.
- Simply copy the teleop_twist_keyboard code on `https://github.com/ros-teleop/teleop_twist_keyboard/blob/master/teleop_twist_keyboard.py` into the python file you just created and save it.
- Under ~/catkin_ws/src/assignment1/launch directory, create a launch file called teleop.launch
- In teleop.launch, copy or type the following code:

```
<launch>
<node pkg="assignment1" type="keyboard_teleop.py" name="fetch_teleop" output="screen">
</node>
</launch>
```

(NOTE: Sometimes when copying into using nano or gedit, the quotation marks are not copyed correctly, which will throw an error when trying to compile.)

Now that we have a launch file, we can run it. Open a new shell, and type

```
$ roslaunch assignment1 teleop.launch
```

After doing this, you should be able to control the fetch using the I,J,K, and L keys. The instructions show the other keys you can use to control the speed etc.

8.2 Common issues

- When you roslaunch teleop.launch, you may see this error message: cannot launch node of type[assignment1/keyboard_teleop.py]: can't locate node [keyboard_teleop.py] in package [assignment1]. To fix this, use the following command,

```
$ cd ~/catkin_ws/src/assignmetn1/src
$ chmod +x keyboard_teleop.py
```

The reason for this is that if your python file may be created without execution permissions and ROS cannot find those files. You can give permissions to those files by typing the above command. If you are interested in this, more details about this could be found on `https://stackoverflow.com/questions/41843622/changing-python-file-permission-with-chmod-x`

- If you see any error message like this: [SOME_FILE] is not a launch file. [SOME_PKG] is neither a package or a launch file. One possible solution is using source command or add it to your ~/.bashrc file so you don't need to type this everytime you open a new shell, more details could be found on `https://answers.ros.org/question/206876/how-often-do-i-need-to-source-setupbash/`

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

9 Building a map!

9.1 Create a mapping launch file

So far we have created several separate shells for each command we used, and it might be difficult to organize if we have more and more shell windows. Instead, combining several commands in one launch file is a better way. We will do this below for teleop, visualization, and mapping.

Create a launch file named *mapping.launch* under *~/catkin_ws/src/assignment1/launch*. edit the file *mapping.launch*, and copy or type the following commands:

```
<launch>
<include file="$(find fetch_navigation)/launch/build_map.launch">
</include>
<node pkg="assignment1" type="keyboard_teleop.py"
name="Fetch_teleop" output="screen">
</node>
<node pkg="rviz" type="rviz" name"$(anon rviz)"
args="-d $(find assignment1)/config/assignment1.rviz">
</node>
</launch>
```

This launch file creates two nodes, one for teleop, and one for visualization. But the real magic happens in the first line, where the *build_map.launch* file from the *fetch_navigation* package is included. Remember that we looked at that file in Section 7.3.1, and saw that it in turn launches the *slam_karto* node, which will do the actual mapping based on the robot's (simulated) measurements.

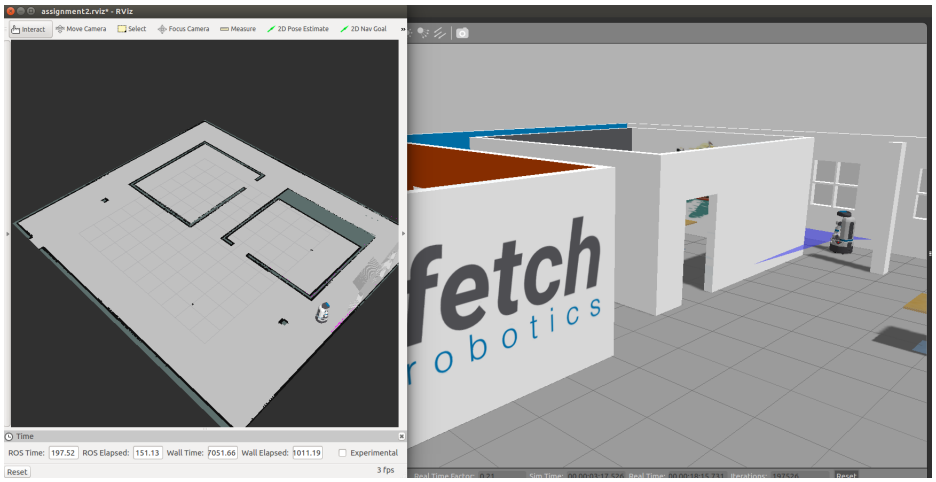
9.2 Move the Fetch to build the map

Keep the simulator running and close all previous shells (except the one for gazebo simulation). Open a new shell and type the following command:

```
$ roslaunch assignment1 mapping.launch
```

Now move the Fetch robot around to build the map! As before, follow the information and instructions printed in the shell to control the Fetch, make sure the shell running mapping.launch is focused. Use your keyboard to move Fetch robot around. It may take a while to build a complete map of the room. Your result may look like this,

- Exercise 4: Move your robot around the room and record a video of your mapping simulation, the length should be less than 30 seconds.



9.3 Save map file

When you finish the map building process (your rviz window should look similar to the pictures shown above), create a new folder *map* in *assignment1* package. Open a new shell, type the following command to save your map. This command will generate 2 files in map folder, a pgm file and a yaml file. For more details about generating and saving maps, `http://wiki.ros.org/map_server`

```
$ rosrun map_server map_saver -f ~/catkin_ws/src/assignment1/map/playground
```

Part V
Localization

Now that we have a map and a robot, we need to be able to localize the robot in the map to find its position and orientation.

10 Create a launch file for navigation

In the Fetch robot tutorial, there is a launch file called *fetch_nav.launch* in the fetch_navigation package. Take a look at the code of this launch file, `https://github.com/fetchrobotics/fetch_ros/blob/melodic-devel/fetch_navigation/launch/fetch_nav.launch` In this assignment, we only need to replace the value of *map_file* with our own map. Create a new launch file named *navigation.launch* in *~/catkin_ws/src/assignment1/launch* folder. In *navigation.launch*, type the following code,

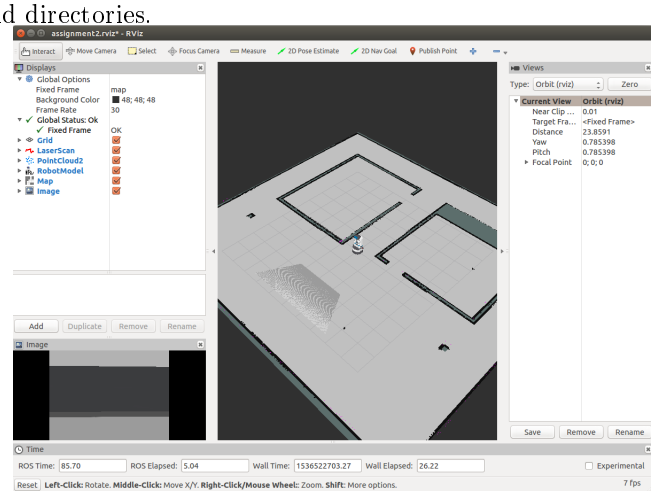
```
<launch>
```

```
<include file="$(find fetch_navigation)/launch/fetch_nav.launch">
<arg name="map_file"
value="/home/student/catkin_ws/src/assignment1/map/playground.yaml"/>
</include>
<node pkg="rviz" type="rviz" name="$(anon rivz)"
args="-d $(find assignment1)/config/assignment1.rviz">
</node>
</launch>
```

Open a new shell and run this launch file, you should keep gazebo simulator running at the same time.

```
$ roslaunch assignment1 navigation.launch
```

If everything goes well, you should see something like the following picture, this result is based on previous steps, if you skip previous steps and get a very different result (if there are status errors in rviz or you fail to visualize the Fetch robot), please go back to previous parts and carefully double check your files and directories.



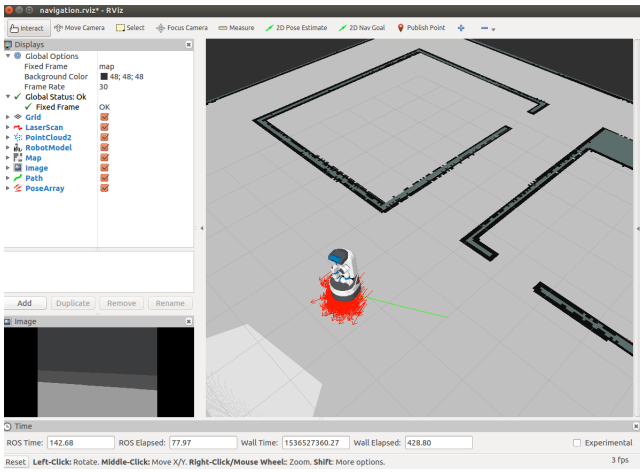
11 See more in Rviz

11.1 update rviz configuration

We could visualize more topics in rviz by adding some new components related to navigation. One is the path planner for Fetch, including the *global path plan* and the *local path plan*, the other one is the pose array(particles) from *amcl* node we launch.

- Add PoseArray -> Topic -> /particlecloud, you should see many red arrows around Fetch after you subscribe to the /particlecloud topic

- Add Path -> Topic -> /move_base/TrajectoryPlannerROS/global_plan
- Save the new rviz configuration under *config* folder , name it as *navigation.rviz*. Click the top left menu, click -> [File] -> [Save Config As] -> /student/catkin_ws/src/assignment1/config -> name it as *navigation.rviz*



11.2 Robot localization

Adaptive Monte Carlo Localization helps robot to localize itself while moving. At the beginning, *amcl* will spread particles throughout the map uniformly. While robot moves, it will detect landmarks/objects which could help it localize itself autonomously. More information about *amcl* could be found on `http://wiki.ros.org/amcl` Follow these steps to see how it works,

- Open a shell, type *rosnode info /amcl* and look for a service called *global_localization*
- Type *rosservice info global_localization* to see more detailed information about this service
- Type *rosservice call global_localization* and see what happens in rviz
- There are red arrows uniformly spread on the map, right? Now open a new shell and type *roslaunch assignment1 teleop.launch*
- Move (both rotation and translation) Fetch robot to see how those red arrows change
- Exercise 5: Follow these steps, move your robot and record a video to show how particles (red arrows) change. The expected result is that all red arrows will eventually gather around Fetch robot. The length of the video should be less than 30 seconds.

Part VI
Navigation

The last part will let you implement navigation of Fetch robot in gazebo and rviz.

12 Fetch navigation in rviz

First we update the rviz config file we use. Open navigation.launch, change the args value to *"-d $(find assignment1)/config/navigation.rviz"*. Keep running gazebo simulator. Close all previous shells, open a new shell and type,

```
$ roslaunch assignment1 navigation.launch
```

On the top menu there is a button called 2D Nav Goal. Click this button and choose a random point on map, release the mouse, the robot will autonomously navigate itself to the goal.

In the *fetch_nav.launch* file, it will first launch a ROS node *move_base,* which provides interfaces for configuring, interacting with *navigation stack* on a robot. The name of plugin for the global planner to use with *move_base* is *base_global_planner*. The default value of *base_global_planner* is *navfn/NavfnROS*. This planner uses Dijkstra's algorithm to compute the navigation function. Now you can enjoy playing with navigation and record a video for your Fetch robot!

- Exercise 6: Record a video ($<$ 30 seconds) to show navigation process.