# Assignment 2: Navigation

September 13, 2018

This assignment has 4 parts: ROS basics, mapping, localization and navigation. The first part is just setup, but to receive full credits you need to upload a zip/tar.gz file containing 3 short videos, less than 30 seconds each (only keep the most related part to requirements). You will find detailed requirements about those videos in the instructions below. The example code in this assignment will be based on the VM provided last week. Feel free to use your own computer if you already have ROS, Ubuntu, packages of Fetch installed locally.

# Part I
# ROS Basics

In the first part we will go through some basic concepts of ROS that will be useful to help you finish the assignment.

## 1 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.

## 2 Creating ROS packages

### 2.1 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 assignment2 std_msgs roscpp rospy
```

### 2.2 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 assignment2 package.

### 2.3 Organizing your package

Create a new folder *launch* in the package you created.

```
$ cd ~/catkin_ws/src/assignment2
$ mkdir launch
```

In your package, you should now have 3 folders: *launch*, *src*, *include*, and two files: *CMakeLists.txt* and *package.xml*. This is the fundamental structure of a ROS package. The *launch* folder contains all launch files. 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.

## 2.4 Launch files

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`

## 2.5 Rviz configuration

In the last assignment 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 assignment2 package. We will save your rviz file in that folder.

  ```
  $ cd ~/catkin_ws/src/assignment2
  $ 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
  ```

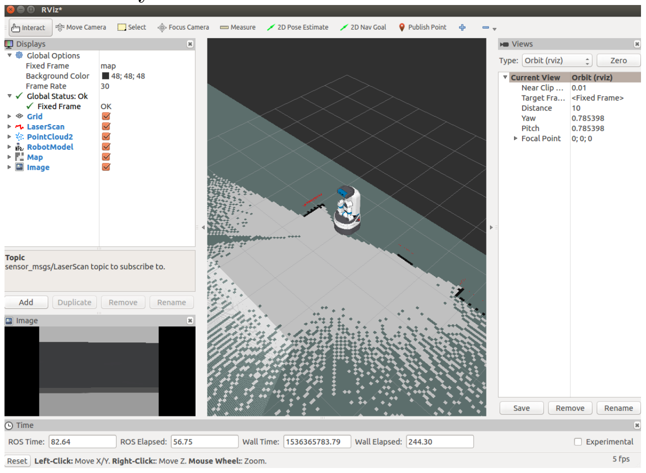
1. 'Global Options' -> 'Fixed Frame' -> map
2. 'LaserScan' -> 'Topic' -> /base_scan
3. 'PointCloud2' -> 'Topic' ->/head_camera/depth_downsample/points
4. 'Map' -> 'Topic' -> /map
5. 'Image' -> Image topic -> 'head_camera/rgb/image_raw'

- Click the top left menu, click -> [File] -> [Save Config As] -> /student/catkin_ws/src/assignment2/config -> name it as assignment2.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/assignment2/config/assignment2.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:



Global Options
Fixed Frame: map
Background Color: 48; 48; 48
Frame Rate: 30
Global Status: Ok
Fixed Frame: OK
Grid ✓
LaserScan ✓
PointCloud2 ✓
RobotModel ✓
Map ✓
Image ✓

Topic
sensor_msgs/LaserScan topic to subscribe to.

Type: Orbit (rviz)
Current View: Orbit (rviz)
Near Clip ...: 0.01
Target Fra...: <Fixed Frame>
Distance: 10
Yaw: 0.785398
Pitch: 0.785398
Focal Point: 0; 0; 0

Time
ROS Time: 82.64 ROS Elapsed: 56.75 Wall Time: 1536365783.79 Wall Elapsed: 244.30

Left-Click: Move X/Y. Right-Click:: Move Z. Mouse Wheel:: Zoom.

### 2.6 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 II
# Mapping

The second part will be related to the mapping problem. We will first learn how to move the robot in simulation, and then use that skill to create a map using rviz and gazebo.

## 3 Keyboard Teleop

### 3.1 Teleop your Fetch robot

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. 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/assignment2/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/assignment2/launch directory, create a launch file called teleop.launch
- In teleop.launch, copy or type the following code:

```
<launch>
<node pkg=''assignment2'' type=''keyboard_teleop.py'' name=''fetch_teleop'' output=''screen''>
</node>
</launch>
```

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

```
$ roslaunch assignment2 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.

## 3.2 Common issues

- When you roslaunch teleop.launch, you may see this error message: cannot launch node of type[assignment2/keyboard_teleop.py]: can't locate node [keyboard_teleop.py] in package [assignment2]. To fix this, use the following command,

```
$ cd ~/catkin_ws/src/assignmetn2/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
```

# 4 Building a map!

## 4.1 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/assignment2/launch*. edit the file *mapping.launch*, and copy or type the following commands:

```
<launch>
<include file=''$(find fetch_navigation)/launch/build_map.launch''>
</include>
<node pkg=''assignment2'' type=''keyboard_teleop.py''
name=''Fetch_teleop'' output=''screen''>
</node>
<node pkg=''rviz'' type=''rviz'' name=''$(anon rviz)''
args=''-d $(find assignment2)/config/assignment2.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 2.4, and saw that it in turn launches the *slam_karto* node, which will do the actual mapping based on the robot's (simulated) measurements.

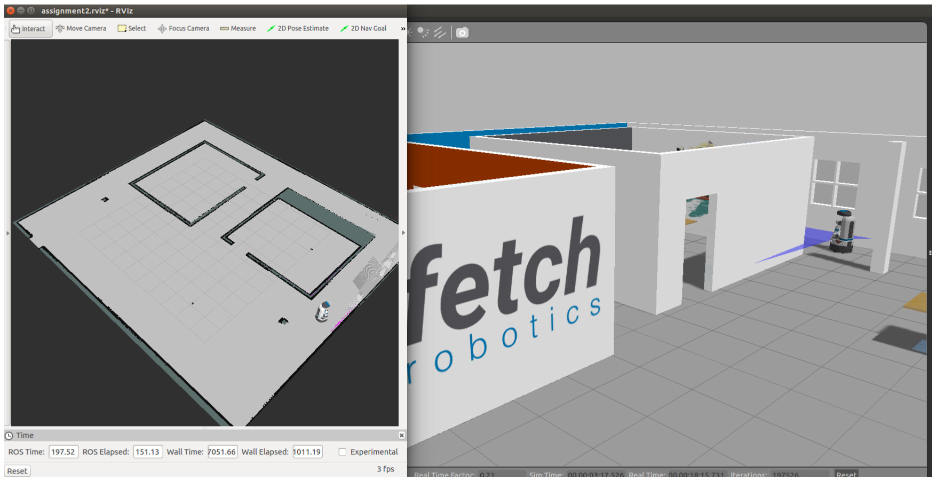
## 4.2 Move the Fetch to build the map

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 assignment2 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 1: Move your robot around the room and record a video of your mapping simulation, the length should be less than 30 seconds.



ROS Time: 197.52 ROS Elapsed: 151.13 Wall Time: 7051.66 Wall Elapsed: 1011.19

## 4.3 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 *assignment2* 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/assignment2/map/playground
```

## Part III
# Localization

In the third part, you will run robot localization.

## 5 Create a launch file for navigation

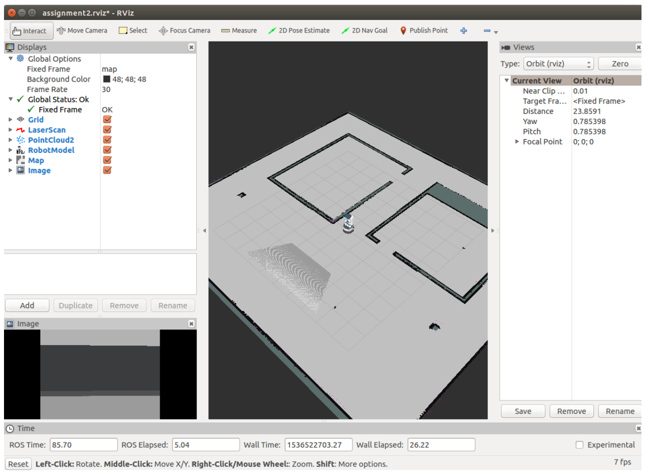
In the Fetch robot tutorial, there is a launch file called *fetch_ nav.launch* in the fetch_navigation package. Take a look at the code of this launch file, `https://github.com/fetchrobotics/fetch_ros/blob/melodic-devel/fetch_navigation/launch/fetch_nav.launch` In this assignment, we only need to replace the value of *map_ file* with our own map. Create a new launch file named *navigation.launch* in *~/catkin_ ws/src/assignment2/launch* folder. In *navigation.launch*, type the following code,

```
<launch>
<include file=''$(find fetch_navigation)/launch/fetch_nav.launch''>
<arg name="map_file"
value="/home/student/catkin_ws/src/assignment2/map/playground.yaml"/>
</include>
<node pkg=''rviz'' type=''rviz'' name=''$(anon rivz)''
args=''-d $(find assignment2)/config/assignment2.rviz''>
</node>
</launch>
```

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

```
$ roslaunch assignment2 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.



Global Options
Fixed Frame map
Background Color ■ 48; 48; 48
Frame Rate 30
Global Status: Ok
✓ Fixed Frame OK
Grid ✓
LaserScan ✓
PointCloud2 ✓
RobotModel ✓
Map ✓
Image ✓

Type: Orbit (rviz) Zero
Current View Orbit (rviz)
Near Clip ... 0.01
Target Fra... <Fixed Frame>
Distance 23.8591
Yaw 0.785398
Pitch 0.785398
Focal Point 0; 0; 0

ROS Time: 85.70 ROS Elapsed: 5.04 Wall Time: 1536522703.27 Wall Elapsed: 26.22

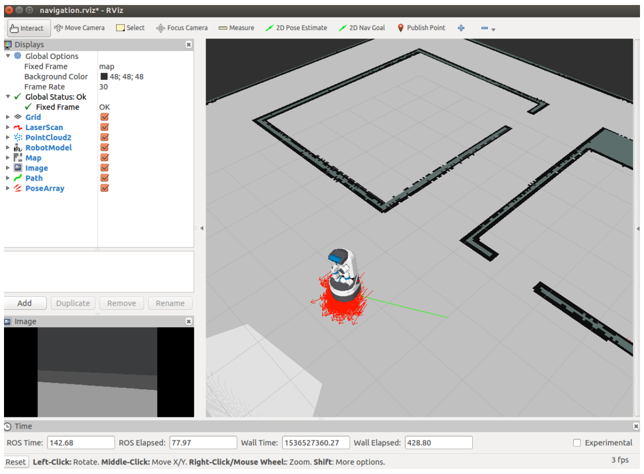
**Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options. 7 fps

## 6 See more in Rviz

### 6.1 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/assignment2/config -> name it as *navigation.rviz*



Global Options
Fixed Frame: map
Background Color: 48; 48; 48
Frame Rate: 30
Global Status: Ok
Fixed Frame: OK

ROS Time: 142.68 ROS Elapsed: 77.97 Wall Time: 1536527360.27 Wall Elapsed: 428.80

**Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

## 6.2 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 assignment2 teleop.launch*
- Move (both rotation and translation) Fetch robot to see how those red arrows change
- Exercise 2: 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 IV
# Navigation

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

## 7 Fetch navigation in rviz

First we update the rviz config file we use. Open navigation.launch, change the args value to *"-d $(find assignment2)/config/navigation.rviz"*. Keep running gazebo simulator. Close all previous shells, open a new shell and type,

```
$ roslaunch assignment2 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.

```
will first launch a node called move_base
```

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 3: Record a video ($<$ 30 seconds) to show navigation process.