**Homework 2 - Simulating a Humanoid Robot**

**COMSE6731 Humanoid Robotics**

Due February 13th, 2018 at 4:00pm

**Assignment**

This homework will help students get familiar with ROS and GraspIt! It consists of three parts.

**Setup**

You may be missing some of the Fetch Gazebo dependencies. Make sure you run this before you start the homework assignment to install all of the dependencies.

```
$ sudo apt-get install -y ros-kinetic-opencv-candidate ros-kinetic-simple-grasping
ros-kinetic-moveit* ros-kinetic-navigation ros-kinetic-slam-karto
ros-kinetic-costmap-2d ros-kinetic-rgbd-launch
```

**Part 1 Simple Simulation Demo**

**Fetch in Gazebo**

This part assumes that you have gone through the ROS tutorial and have a basic knowledge of ros concepts like ros workspace, nodes, launch files etc.

We adapted a portion of the [fetch documentation](https://github.com/fetchrobotics/fetch_ros) into the following steps. You are encouraged to go through the page for more details.

- **Create a ROS workspace**
  ```
  $ source /opt/ros/kinetic/setup.bash
  $ mkdir -p ~/fetch_ws/src
  $ cd ~/fetch_ws
  $ catkin init
  ```

- **Clone relevant ROS packages**
  ```
  $ cd src
  $ git clone https://github.com/fetchrobotics/fetch_ros.git
  $ git clone https://github.com/fetchrobotics/robot_controllers.git
  $ git clone https://github.com/HumanoidRobotics/fetch_gazebo.git
  ```

- **Build packages**
  ```
  $ catkin build
  ```

- **Run set-up (Command 1) Start Gazebo simulator with the playground:**
  ```
  $ source devel/setup.bash
  $ roslaunch fetch_gazebo playground.launch
  ```
  Wait until the simulator is fully running and then run the demo launch file in a new terminal

- **RVIZ visualization** You can visualize several details about the robot in rviz. RViz is a tool for
debugging various components in ROS. It is critical that you know the difference between RViz and Gazebo. Gazebo is used for simulation of robotic environments and replaces the need for a physical robot. RViz is just a view into what is currently being broadcast in RViz and requires either a simulated environment or a physical environment to be in use, otherwise it is useless. In a new terminal window:

```
$ source devel/setup.bash
$ rosrun rviz rviz
```

Add the robot model to the scene in rviz. You can add other components to rviz (play around with this).

**Deliverables for part 1**

- Demo the simulation of the Fetch robot to your TA. You will need to schedule a time slot to do the demo.
- List 5 properties of the Fetch robot you can access in rviz.
- What is the difference between RViz and Gazebo? List three purposes of each with sources, and then three differences between each one with sources.
- How many nodes are running in the ROS network after you've run the commands under Command 1? Command 2? After RViz? Also how many ros topics are available for each of the following commands?

**References**

- [http://docs.fetchrobotics.com/gazebo.html](http://docs.fetchrobotics.com/gazebo.html)

**Part 2 Building a map using slam**

This part assumes that you have done Part 1 and makes use of some of the packages downloaded and built for Part 1.

You are encouraged to go through this [fetch documentation page](http://docs.fetchrobotics.com) for more details.

- Run set-up Start Gazebo simulator with the playground:
  
  ```
  $ roslaunch fetch_gazebo playground.launch
  ```

  Wait until the simulator is fully running and then run the fetch navigation launch file:

  ```
  $ roslaunch fetch_navigation fetch_nav.launch map_file:=EMPTY
  ```

  If you have trouble running the map_server you will have to install it:

  ```
  $ sudo apt install ros-kinetic-map-server ros-kinetic-amcl ros-kinetic-move-base ros-kinetic-slam-karto
  ```

  Run the build map launch file:

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

- RVIZ visualization
  
  ```
  rosrun rviz rviz
  ```
Remember to add the robot model to rviz. Also add Map to rviz so you can visualize the map of
the world you will build.

Deliverables for part 2

- Demo the simulated robot to the TA to affirm you have run everything properly.
- What is the least number of navigation goals required to have 90% of the environment built?
- Display a PointCloud2 in RViz. Which topic displays what the robot is currently seeing using it
depth camera? How many navigations are required from the beginning of the simulation to view
the “Fetch Robotics Logo”? Demonstrate that you can view the logo during your demo and include
a screenshot in your submission.

References

- http://docs.fetchrobotics.com/navigation.html#building-a-map

Part 3 Grasp planning using Graspl!t!

This part is to get you familiar with using Graspl!t! for robots. There are many potential grasp
planning libraries you can use but given that we are most familiar with Graspl!t! (our robotics lab
made it!) we want you all to be familiar with simple use cases of it. You'll need to install grasplit and
graspl!t_commander and demonstrate you can grasp a simple object to a TA.

Setup

All of the following instructions can be found at https://github.com/CRLab/graspl!t_commander and

First clone Graspl!t! and install it on your computer.

```
$ sudo apt install libqt4-dev libqt4-opengl-dev libqt4-sql-psql libcoin80-dev
libsoqt4-dev libblas-dev liblapack-dev libqhull-dev libeigen3-dev
$ cd
$ git clone https://github.com/graspl!t-simulator/graspl!t.git
$ cd graspl!t
$ mkdir build && cd build
```
$ cmake ..
$ make -j$(nproc)
$ sudo make install
$ echo "export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH" >> ~/.bashrc
$ echo "export GRASPIT=~/.graspit" >> ~/.bashrc
$ source ~/.bashrc

Go to your fetch_ws/src directory and clone the graspit_commander repository and the graspit_interface repository.

$ cd ~/fetch_ws/src
$ git clone https://github.com/CRLab/graspit_commander.git
$ git clone https://github.com/graspit-simulator/graspit_interface.git

Rebuild fetch_ws

$ cd ~/fetch_ws
$ catkin build

Make sure your installation works

$ cd ~/fetch_ws
$ source devel/setup.bash
$ roslaunch graspit_interface graspit_interface.launch

And in another terminal

$ cd ~/fetch_ws
$ source devel/setup.bash
$ roscd graspit_commander
$ py.test

Using GraspIt!

There are two ways to use GraspIt!. You can either use the graspit_commander python library or through the GraspIt! GUI. For the purposes of this homework and most likely your projects should you choose to do a project involving grasping, it is best if you use the python interface. First load up graspit_interface

$ cd ~/fetch_ws
$ source devel/setup.bash
$ roslaunch graspit_interface graspit_interface.launch

You will see a blank workspace show up similar to this image
You can interact with the workspace using a python terminal, personally I prefer to use IPython for the tab completion. You can also use an IPython Notebook if you would like.

```python
In [1]: from graspit_commander import GraspitCommander
In [2]: gc = GraspitCommander()
```

This initializes a GraspitCommander object which gives you access to a bunch of functions to interact with GraspIt!

```python
In [3]: dir(gc)
Out[3]:
[....
 'approachToContact',
 'autoGrasp',
 'autoOpen',
 'clearWorld',
 'computeEnergy',
 'computeQuality',
 'dynamicAutoGraspComplete',
 'findInitialContact',
 'forceRobotDof',
 'getBodies',
```
Now that we have an instance of `GraspitCommander` we can test grabbing an object.

```python
In [10]: gc.loadWorld("plannerMug")
In [11]: gc.approachToContact()
In [12]: gc.autoGrasp()
```

You will see that the hand properly grabs the mug. We can test this for a variety of objects, anything loaded as an STL, VTK, or PLY, as well as for a variety of robots.

**Deliverables for part 3**

- Demo GraspIt! running on your computer to the TA. You will need to show the `plannerMug` scene as well as how to use the `graspit_commander` interface.
- Write a python script called `part3.py` that:
  - Loads in the fetch gripper and fully extend out the fingers
  - Loads in a mug into the scene
  - Places each into a position where the Fetch hand can grasp the mug
  - Have the fetch hand grasp the mug
  - Prints the pose of the grasp (hint: `gc.getRobot(0)`
- You will need to demo this to the TA

**Submission Instructions**

Each group will need to submit a tarball with the following content:

```
hw2_group<number>_<uni1>_<uni2>_<uni3>.tar.gz
  hw2_group<number>_<uni1>_<uni2>_<uni3>/
    answers.<txt or pdf>
    part3.py
```
This means that your tarball is not a tarbomb, does not exist in a subdirectory of subdirectories, and is not a zip/7zip/rar file. This means that if I run `tar -xvf hw2_group<number>_<uni1>_<uni2>_<uni3>.tar.gz`, a folder with the name `hw2_group<number>_<uni1>_<uni2>_<uni3>` will be present in my cwd. This makes my life way easier when grading and means I can get your grades back to you much faster. Anybody that deviates from this format will not be graded for this assignment so if any of this is unclear please post in the Piazza.

Each group’s answers should be of the following or similar format:

```
Homework 2
Group <Number>
Members: <Name 1> <UNI 1> - <Name 2> <UNI 2> - <Name 3> <UNI 3>

# Part 1
1. List 5 properties of the Fetch robot you can access in rviz.
2. What is the difference between RViz and Gazebo? List three purposes of each with sources, and then three differences between each one with sources.
3. How many nodes are running in the ROS network after you've run the commands under Command 1? Command 2? After RViz? Also how many ros topics are available for each of the following commands?

# Part 2
1. What is the least number of navigation goals required to have 90% of the environment built?
2. Display a PointCloud2 in RViz. Which topic displays what the robot is currently seeing using it depth camera? How many navigations are required from the beginning of the simulation to view the "Fetch Robotics Logo"? Demonstrate that you can view the logo during your demo and include a screenshot in your submission.
```