Getting Familiar with the Robot Operating System (ROS)

The Robot Operating System is a flexible framework for writing robot software. It is a collection of tools and libraries that simplifies the task of creating complex and robust robot behavior across a wide variety of robotic platform.

ROS by itself offers a lot of value to most robotics projects, but it also presents an opportunity to network and collaborate with the world class roboticists that are part of the ROS community. Over the past several years, ROS has grown to include a large community of users worldwide. ROS being an open source project, the code within it is the result of the combined efforts of this international community.

Getting started with ROS:

There are many ROS distribution available for use. Some are older releases with long term support, making them more stable, while others are newer with shorter support life times. Considering this factor, we will be using **ROS Kinetic Kame** for the implementation of the theme. This distribution is only available for Ubuntu (16.04 LTS).
We strongly recommend you to use ROS kinetic as the tasks have been tested against this distribution.

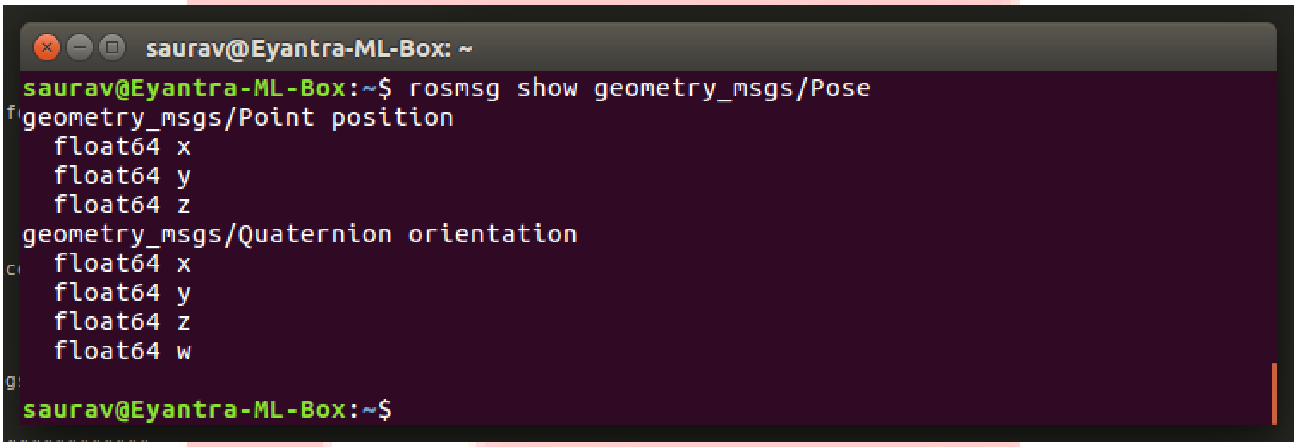
Understanding ROS:

1. Please go through this video tutorial to understand :
   i) Creating a workspace and creating a Package
   ii) Writing a simple subscriber and publisher in ros-python
   iii) What is a launch file? Different tags in launch file
2. You can go through this ROS cheatsheet to find various ROS commands. The following ros commands will come handy.
   i) `rosnode list`
   ii) `rostopic list`
   iii) `rosnode info`
   iv) `rostopic info`
   v) `rostopic type`
   vi) `rostopic pub`
   vii) `rosmsg show`

3. **Rosmsg**: Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs, as you can see in Figure 1).

   i) Nodes can also exchange a *request* and *response* message as part of a ROS service call. These request and response messages are defined in srv files. for more details, visit link.

   ii) Now, learn how to decipher callback messages using `rosmsg show` <message_type > to know about the messages of corresponding topics. Refer to Figure 1 to see an example of the message_type = geometry_msgs/Pose.



```
saurav@Eyantra-ML-Box: ~
saurav@Eyantra-ML-Box:~$ rosmsg show geometry_msgs/Pose
geometry_msgs/Point position
  float64 x
  float64 y
  float64 z
geometry_msgs/Quaternion orientation
  float64 x
  float64 y
  float64 z
  float64 w

saurav@Eyantra-ML-Box:~$
```

Figure 1: geometry_msgs/Pose

   iii) In order to access this message in the callback function, use dot operator to access member variables. Eg : To access x, you will have to use something like *data.position.x* where, *data* is the argument of the callback function.

4. You can go through ROS wiki tutorial page for understanding ROS in more detail.
5. Some of the additional resources which may help you in understanding ROS are:
   i) Mastering ROS for Robotics Programming - Lentin Joseph
   ii) Programming Robots with ROS: A Practical Introduction to the Robot Operating - Brian Gerkey, Morgan L. Quigley, and William D. Smart
   iii) Youtube Tutorials