

## Lab0 - Introduction to ROS

### 1 The goal

The goal of this lab session is to get started with ROS that will be used in all the lab sessions of Probabilistic Robotics. The Robot Operating System (ROS) is a software framework for robot software development. ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS will be used in the next lab exercises, so it is fundamental to understand all the basics given in this session.

### 2 Pre-lab: Installing Ubuntu and ROS

Currently the only official supported platform for ROS is Ubuntu. Ubuntu is a computer OS (Operating System) based on Debian Linux. As the computers in the Robotics Lab doesn't have Ubuntu installed, the first steps you need to do is install Ubuntu and ROS in your machines. In order to install Ubuntu you can either:

1. (preferred) Install Ubuntu in a partition alongside your other OS (hard disk install). Make sure that you install the distributions specified below:
  - Ubuntu 12.04.5 LTS (download iso).
  - ROS Hydro Desktop Full install (install instructions).
2. (not recommended) Install Ubuntu in a virtual machine inside your normal OS. In this case you can download a Virtualbox and follow the same installation instruction than the previous point, but this time on the virtual machine. Note that this kind of installation can lead to problems with graphics drivers in some or all labs.

### 3 Lab: ROS tutorials

Now that we have Ubuntu and ROS successfully installed, please follow the most fundamental ROS tutorials. Please have in mind that your ROS workspace, the place where you need to put all your packages is the `catkin_ws` folder inside the home directory.

- Creating a workspace for catkin.  
[http://wiki.ros.org/catkin/Tutorials/create\\_a\\_workspace](http://wiki.ros.org/catkin/Tutorials/create_a_workspace)
- Navigating the ROS FileSystem.  
<http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem>
- Creating a ROS package.  
<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>
- Understanding ROS Nodes.  
<http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes>
- Understanding ROS topics.  
<http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics>
- Understanding ROS services and Parameters.  
<http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams>
- Writing a simple Publisher and subscriber (Python).  
[http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber\(python\)](http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(python))
- Examining the Simple Publisher and Subscriber.  
<http://wiki.ros.org/ROS/Tutorials/ExaminingPublisherSubscriber>

Please note that the ROS Cheat Sheet, you can find in the files for this Lab might be useful for you.

### 3.1 Turtlesim Waypoint Node

During the Lab session you will need to create a node to interact with the turtle simulator seen in Understanding ROS nodes tutorial. Run again turtlesim:

```
roscore
```

In a new terminal:

```
roslaunch turtlesim turtlesim_node
```



Figure 1: Turtlesim visualization.

And using the ROS tools you've seen in the tutorials (rostopic, rosmmsg, rqt\_graph) identify:

1. The name of the topic used to send velocity commands to the turtle and the type of these messages.
2. The name of the topic where the turtle position is published at, and the type of these messages.

Clone the repository of the Probabilistic labs in your workspace. This repository contains all the code for this labs. After download compile the code:

```
cd ~/catkin_ws/src
git clone https://bitbucket.org/gvallicrosa/probabilistic_labs.git
cd ~/catkin_ws
catkin_make
```

Remember before starting any lab to update the repos with:

```
cd ~/catkin_ws/src/probabilistic_labs
git pull
cd ~/catkin_ws
catkin_make
```

Run the node in the turtle\_waypoint package:

```
roslaunch turtle_waypoint turtle_waypoint.py
```

You will see that nothing happens. You need to modify the *turtle\_waypoint.py* node inside the package in order to publish commands to the turtle to make it go to a desired point. This desired position will be specified when running the node through the command line like:

```
roslaunch turtle_waypoint turtle_waypoint.py 1.2 5.0
```

being the desired point is x: 1.2, y: 5.0. If the point is not given when calling the node, the node will check the ROS parameter server for the parameters */default\_x* and */default\_y* and head to that point. In the case that the parameters are not set the turtle won't move. You can set ROS parameters like:

```
rosparam set default_x 5
rosparam set default_y 10
```

## 4 Lab report

After the lab session: Write a brief report (max 2 pages) explaining your solution and problems faced. Include the final *turtle\_waypoint.py* file.