CS3027/5095 Robotics

Practical I

2016

Introduction

The purpose of this practical is to familiarise you with the ROS environment. It should also illustrate some of the power of ROS, allowing you to switch between robots and simulators (nearly) seamlessly.

Before the Practical

Please install ROS indigo, either by following the instructions at `http://wiki.ros.org/indigo/Installation/Ubuntu`, or by installing Virtualbox (`https://www.virtualbox.org/`) and installing a virtual machine from here: `http://nootrix.com/software/ros-indigo-virtual-machine/`. Alternatively, you can use ROS from one of the PCs in Meston 204.

Don't forget to set up ROS for your user by running `rosdep update` and `source /opt/ros/indigo/setup.bash`. You'll need to run the latter command every time you open a new terminal, unless you configure it to run automatically, to do so, edit the file `.bashrc` in your home directory, and put the `source` command in there.

Tasks

This practical aims to familiarise you with the ROS concepts of packages, nodes, topics and messages, as well as most important ROS command line tools. If time permits, we'll also get you to write your first ROS program.

1. Read, and follow the instructions, of Chapter 2 of A Gentle Introduction to ROS, starting at Section 2.3. Ensure you understand the relationship between packages and nodes, and nodes and topics.

2. While we'll typically create and run individual nodes within this course, it's good to get familiar with the ROS package system, and to develop good programming practices by ensuring that your code lives within an appropriate package. We'll follow instructions similar to those found in Section 1-4 of the ROS tutorial on creating a ROS package.

(a) You'll create your packages within a directory called a *workspace*. Use the UNIX `mkdir` command to create such a workspace (you could for example, name it `catkin_ws`.

(b) After creating this directory, change into it, and create a subdirectory called `src`. Your packages will live within this `src` directory.

(c) Change into the `src` directory, and run the following command

```
catkin_create_pkg hello std_msgs rospy roscpp
```

The `catkin_create_pkg` command creates a package (in this case named `hello`) and says that it depends on the `std_msgs`, `rospy` and `roscpp` packages.

(d) Now change directory into your workspace, and run `catkin_make` to compile the package (this won't do much at the moment).

(e) If you run `ls`, you'll see that several new directories have been created.

(f) To add the workspace to your ROS environment, you need to `source` a setup file, to do so, run `source devel/setup.bash`.

(g) You can now add code to the `hello` package, and this code can be compiled via the `catkin_make` command (if you're using C/C++), and run via rosrun.

3. We'll now create our first two ROS nodes. These nodes will communicate with each other by creating an appropriate topic and sending messages on it. I'm assuming you're using Python, and so follow the instructions from the tutorial found at `http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(python)`. If you're using C/C++, follow the instructions from `http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c++)`. In both cases, start by using `roscd hello` to change to the `hello` package rather than to the `beginner_tutorials` package as instructed there (note that `roscd` allows you to change to a ros package directory). Similarly, at the end, rather than changing directory to `catkin_ws`, change directory to your workspace.

4. You can run the code using `rosrun hello listener.py` and `rosrun hello talker.py`.

5. Finally, let's put everything together: write a program which causes the turtle from the first task to move around according to programmatic (rather than keyboard) commands. You'll need to publish messages of type `Twist` (from the python `geometry_msgs.msg` package) to the {/turtle1/cmd_vel} topic. A sample solution can be found on the next page.

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist

def twister():
    rospy.init_node('twister', anonymous=True)
    pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    rate = rospy.Rate(5) # 5hz
    while not rospy.is_shutdown():
        tw=Twist()
        tw.linear.x=1 #move forward
        tw.angular.z=1 #and turn left
        pub.publish(tw)
        rate.sleep()

if __name__ == '__main__':
    try:
        twister()
    except rospy.ROSInterruptException:
        pass
```