Autonomous Robots

Lab 3 : Part 1 - Introduction to ROS

Hassan Saeed
Shubham Wagh
Anirudh Puligandla
Nayee Muddin Khan Dousai

Dated : May 7, 2017

Escola Politècnica Superior (UdG) - VIBOT

1 Introduction

Purpose of this lab was to get acquainted with the ROS framework. This lab was essential to proceed with the other two labs.

2 Catkin Workspace

Objective of this lab was to get familiar with the ROS "framework", a meta operating system that is well known in the field of robotics and widely used these days. The foremost features of the ROS framework that were used in this task were the nodes and topics. Nodes represent the processes that control various tasks. It is like a specific block of program that deals with a specific thing e.g. in robotics node can represent a wheel odometer or a path planner. The way these nodes communicate to each other, is through topics. So topics are like a channel that nodes can send data on the other nodes. Data sent on the topics are called **messages** e.g. wheel odometer can send x, y, z position of a robot. **Topics** are used, to publish (send) and subscribe (receive) these messages. A set of request and reply is known as a **service**. This lab was initialized by creating the *catkin* workspace. Following the commands stated in the lab-guide, workspace directory was created. This directory stores all the created packages. Following the initialization of the *catkin* workspace using **catkin_make**, workspace is compiled and completed.

3 Catkin ROS package

In this sectoin, we had to create the ROS packages. Package is a self-contained directory contating source, makefiles. Several ROS packages are available such as sensor drivers, controllers etc. This created package must be in a catkin workspace. Using stated ROS commands in lab-guide, package is created.

4 ROS, hello world node (C++)

Using *roscd hello_world_pkg* hello world package was created.Then using *geany* code was implemented in C++. Following this roscore was ran in one terminal whereas in second terminal *helloworld* node was executed.Following output was obtained.

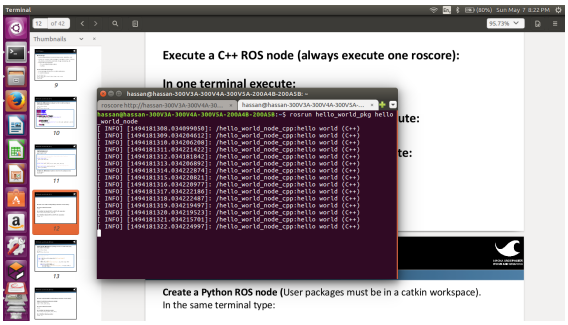

Figure 1: Hello world note output

Same *helloworld* package was initialized but this time using Python ROS node. Similar output was obtained as shown for C++ ROS node.

5 Robot example package

Next in the queue was to create the Robot example package. For this package we defined standard messages and geometry messages as: *catkin_create_pkg robot_example_pkg std_msgs geometry_msgs rospy roscpp* Using geany navigator head header and cpp file were created, with controller header and cpp file. Using *rosrun robot_example_pkg navigator* ros node navigator initalized.

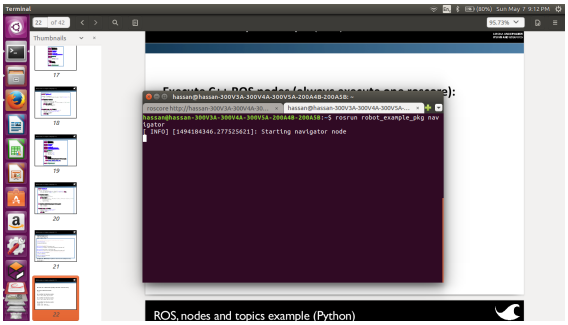

Figure 2: Navigator node Initalized

Following navigator node, controller node was initialized form robot example package by the command: *rosrun robot_example_pkg controller*

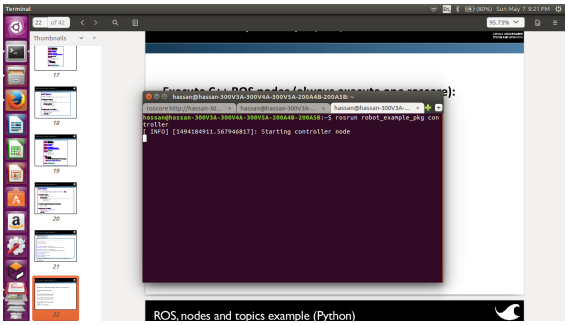

Figure 3: Controller node Initalized

Typing rosnode list provides us with all the current nodes running. While both controller and navigator node were active, running rosnode returned:

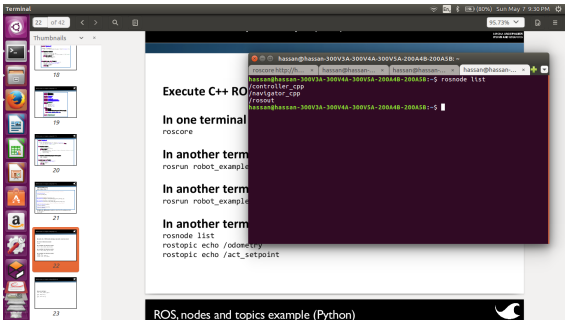

Figure 4: ROS nodelist

When navigator node was echoed it was observed it was publishing the pose message $x = 1, y = 2$ and $z = 5$ from */Odom* topic. Similarly controller node was publishing the $x = 3, y = 6$ and $z = 15$.

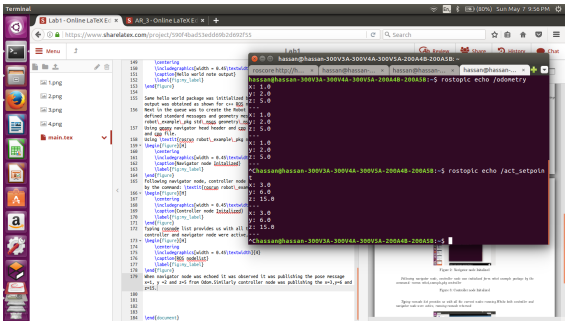

Figure 5: RosTopics

6 ROS, Defining Custom Messages

Here we had to create a message folder.For this first *roscdrobot_example_pkg* command was ran in terminal. Then we had to modify *CMakeLists.txt* file via running *roscdrobot_example_pkg*. Following this messages were compiled using *catkin_make* command. Then both the controller header and cpp files were edited before running navigator and controller node from robot example package. In new terminal rosnode list command was ran which returned all the current active nodes. Following this certain node topics were echoed, result is shown in the following diagram.

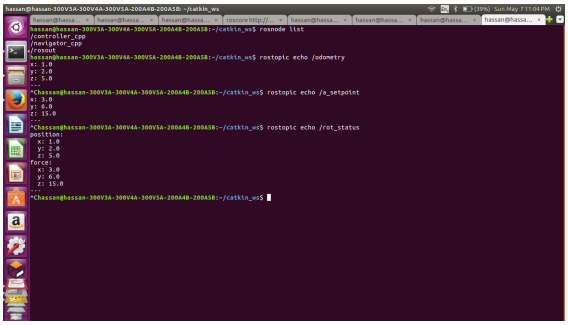

Figure 6: RosTopics and nodes echoed responses

7 ROS, Defining Services

First task here was to create the message folder using *roscd robot_example_pkg* followed by make directory command: *mkdir -p srv* Then we had to define a custom message follwoed by editing the controller header and .cpp file.

8 Conclusion

This lab though composed of basic ROS commands gave us good exposure towards ROS. We learned the basics about ROS, including ROS nodes, topics, services, messeages and clients.Then we learnt how to create a ROS package.