University of Girona

Probabilistic Robotics Course

Lab0-guide report

By: Mohammad Rami Koujan

February 28, 2016

Contents

0.1 Introduction

During this Lab session, I have started to be familiar with ROS (Robot operating System) program by going through a number of Beginner-level tutorials which helped me in understanding the basics of this program and learning some important commands. Eventually, I wrote a ROS node in python programming language, the aim of which is to move the turtle node that exists in the turtlesim package to a predefined location. So, basically, this step was helpful to thoroughly comprehend the concepts that I had learned.

0.2 Analysis of the node program

Starting from some basic python code, the task is to steer the turtle node to a destination determined by the program user. In order to do so, it was initially necessary for me to determine the required types of messages to be communicated (Twist,turtlesim.msg.Pose) among the used nodes, which assisted me to import the corresponding essential libraries in addition to the libraries already included. Then, the idea is to define a node that will publish the command velocity messages to the turtle node using '$/turtle1/cmd-vel'$ topic and listen (subscribe) to another topic (/turtle1/pose) to get the current position of the turtle node being steered. In fact, these two topics are created by the turtle node, which is a publisher of the first one and a subscriber to the second one, when running it. Therefore, the created node will initially check the passed position parameters. If they are not given, the node will search for the default values inside the parameter server or do nothing in case there are no default ones. After that, it will wait for getting the current position of the turtle, which is done by the callback function that sets three global parameters according to the turtle x,y, and theta parameters values. The next step is to set the the value of the twist message's linear and angular velocities variables. This is performed by first calculating the Euclidean distance between the current and desired position of the turtle and then making the linear velocity variable x equal to this value as follow:

$$twist.linear.x = 2 * \sqrt{(x - x\_desired)^2 + (y - y\_desired)^2} \tag{1}$$

$$twist.linear.y = 0 \tag{2}$$

$$twist.linear.z = 0 \tag{3}$$

where number 2 in eq(1) plays the role of a control factor to adjust the speed of the linear velocity. One of the main advantages of setting the linear velocity variable x in the mentioned way is that the turtle will start with higher velocity and then its acceleration will decrease as it gets closer to the destination point. The other parameter to be set is the angular velocity, as shown below:

$$twist.angular.x = 0 \tag{4}$$

$$twist.angular.y = 0 \tag{5}$$

$$twist.angular.z = \arctan(\frac{y\_desired - y}{x\_desired - x}) - theta \tag{6}$$

here also the value of the angular velocity z is set in a relative manner, with respect to current orientation theta. This will achieve a decreasing acceleration as well. The only tricky part is that the value of the angular parameter z should be checked if it is smaller than -π or greater that π in order to add or delete 2π respectively. Otherwise, the turtle will tend to rotate for longer distance and more importantly be constrained in infinite loops in some cases, rotating left and right continuously if it is in front of a corner. Experimentally, I chose to set a control factor of value 4 for z parameter in order to reach the required point more efficiently. Finally, the node should publish the command message and keep checking the euclidean distance between the current and desired positions in order to exit the loop in case it is smaller than the tolerance variable.

References

[1] http://wiki.ros.org/.