Lab 2

Getting Started with ROS

Objective

The goals are to:

- Understand basics of ROS
- Implement simple nodes to send (publish) and receive (subscribe)
- Experience Create 2 robot and compare with Gazebo simulator
- Experience different tools in ROS to debug and visualize.

Reference Materials

- ROS Website: wiki.ros.org/ROS/Tutorials: Sections 1.5-1.6, 1.11,1.13,
- ROS Website: wiki.ros.org/turtlebot_stage
- Learning ROS for Robotics Programming by Aaron Martinez, Enrique Fernandez: Chapters 2, 3
- ROS By Example by R. Patrick Goebel: Chapter 7
- Appendix to this instruction
- Create 2 ROS driver: https://github.com/AutonomyLab/create_autonomy

Prelab

1. Describe what each command does:

- $ source
- $ catkin_init_workspace
- $ rosrun
- $ rostopic
- $ printenv | grep
- $ catkin_make
- $ rosnode
- $ rosmsg
- $ catkin_create_pkg
- $ roscore
- $ rospack
- $ rossrv

2. Describe what is each line for in CMakeList.txt file:
   - include_directories
   - add_executable
   - target_link_libraries

3. Describe the difference between roscpp and rospy.

4. Explain the difference between ros::spinOnce() and ros::spin() in a .cpp code.

5. Describe the input parameters of the following functions:
   - rospy.Publisher() and rospy.Subscriber()ros::
   - ros::NodeHandle.advertise() and ros::NodeHandle.subscribe().

6. Describe the following line of codes:
   - **In a .cpp code**:
     - ros::init(argc, argv, "node_name");
     - ros::start();
     - ros::Rate()
     - ROS_INFO_STREAM("Hello, world!");
     - ros::spinOnce();
     - ros::shutdown();

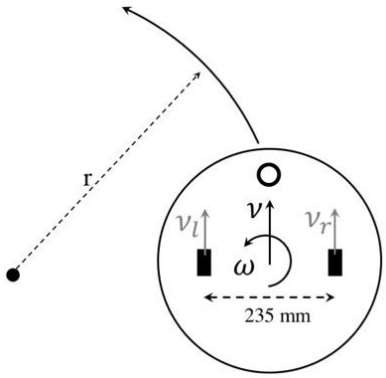
   - **In a .py code**:
     - rospy.init_node("node_name", anonymous=True)
     - rospy.loginfo("Hello, World!")
     - rospy.Rate()
     - rospy.spin()
     - rospy.on_shutdown(handler_function)

Background

The Create 2 robot can be controlled by a ros package, create autonomy, installed on raspberry pi's and lab PCs (you need to install on your laptop by copying create_autonomy folder under /usr/local/eecs_4340 to <your_workspace>/src and running catkin_make). The package communicates with robot through USB serial port providing topics that can be subscribed to get information from robot and be published to send command to the robot. To control the robot using this package, you need to launch the driver by:

$ source ros_create_ws/devel/setup.bash
$ roslaunch ca_driver create_2.launch

once you launch the driver you can see all topics available, i.e. /cmd_vel, /battery. Make sure you understand the purpose and limitations of all topics by reading the package documentation. Specifically, you will need to know the topic to send velocity commands to the robot. The geometry_msgs/Twist message type has linear and angular velocity where in this case linear x and angular z are the ones used to drive the robot since it has 2 degrees of freedom.



where ω, v are angular and linear (forward) velocities, respectively. r is the radius of turning which can be found by r = v/w.

Lab Procedure

Part 0: Creating a ROS workspace

Find the ROS distribution <**your_ros_distro**> on your hardware.
$ rosversion -d

Source the ROS installation to have access to ROS utilities every time you open a new terminal. You can also place this command line in your .bashrc file.
$ source /opt/ros/<your_ros_distro>/setup.bash

Then, run:
$ mkdir –p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ~/catkin_ws
$ catkin_make

From here, you can create as many packages as you want under the **src** folder of your **catkin** workspace. When creating your package, you can provide as many dependencies as you need to run your nodes.
$ cd ~/catkin_ws/src
$ catkin_create_pkg <your_package_name> <dependency_1> ... <dependency_n>
$ cd ~/catkin_ws
$ catkin_make

You will need to reproduce these steps when working with the real robot (**see appendix**).

Part 1: Creating Publisher/Subscriber Nodes

Write two programs to continuously publish and subscribe messages containing a counter. That is, the "talker" node publishes a message: "Message No. #X from talker" to a topic and the subscriber node subscribes to the same topic and prints the received message. You should be able to receive all messages from talker node without any missing. Do not forget to run the master node to make all your nodes communicate with each other using:

$ roscore

Part 2: Make the robot to circle around a small region

In this part, you need to write a program to drive the robot and make a circle with a specific radius of turning (r). You will examine your program using both simulator and robot.

Your program should publish a message with geometry_msgs/Twist type containing angular and linear velocities to an appropriate topic.

1. **Using the simulator:**
    a. **Using the Gazebo Simulator:**

The Gazebo simulator can be used to simulate the turtlebot robot and test your program. It can be launched on your machine by running the following command on a terminal:

$ roslaunch turtlebot_gazebo turtlebot_world.launch

From the terminal, you can run the following command to publish linear velocity along x and angular velocity along z to make the robot move in a circle motion:

$ rostopic pub -r 10 /cmd_vel geometry_msgs/Twist '{linear: {x: 0.4, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.4}}'

Hint: You can remap arguments of the node when you are running it by below command. By doing that you don't need to create different nodes for simulator and robot.

$ rosrun package_name node_name old_topic:=new_topic

    b. **Using the Stage Simulator:**

The Stage simulator can also be used to simulate the turtlebot robot and test your program. It can be launched on your machine by running the following commands on a terminal:

$ roscd turtlebot_stage
$ rosrun stage_ros stageros maps/stage/<choose_the_map_to_launch>

From the terminal, you can run the following command to publish linear velocity along x and angular velocity along z to make the robot move in a circle motion:

2. **Using the real robot:**

To drive the robot, login to raspberry pi and run your node (for information about how to login to raspberry pi see the appendix section at the end of the document). Make sure that the driver has been launched before running your program. Try to measure the error made by slippage when robot makes 10 loops.

Part 3: Make the robot to draw a polygon

For this part, you need to write functions that can control the robot to move forward/backward given a distance and turn left/right given an angle. That is, by monitoring the pose of robot (from odometry) you can find the distance/angle traversed. Once you have these functions, you can call them multiple times and make the robot to draw a polygon (e.g. square). Make sure you run your program on the simulator before running on the real robot. Try to draw the robot's path by attaching a dry erase marker to the robot while moving on the hall way and report your observation (**Clean the floor after the experiments!**).

Part 4: Using ROS tools

In this part, you will try to use graphical tools like rviz which cannot be run on raspberry pi. Fortunately, ROS has the solution using network communication allowing nodes to be run on multiple computers in the same network.

ROS Networking: By making one computer (desktop PC or your laptop) as master and raspberry pi as slave, you can reach to all topics that are running on raspberry pi. To do that following steps need to be done:

1. Desktop PC (laptop) Configuration:

   You need to configure the PC (laptop) as the master. To do that, enter the following two lines on a terminal.

   $ export ROS_MASTER_URI=http://<ip_address>:<port>
   $ export ROS_IP=<ip_address>

   where <ip_address> is the ip address of PC (can be found by $ifconfig command, e.g. 10.14.1.20X) and <port> is an arbitrary number in the range of 1025 and 65535. Note that you need to do this every time you open a new terminal. Make sure you are not using same port as other groups.

2. Raspberry pi Configuration:

Raspberry pi needs to be configured as well. Log in to raspberry pi and enter following:

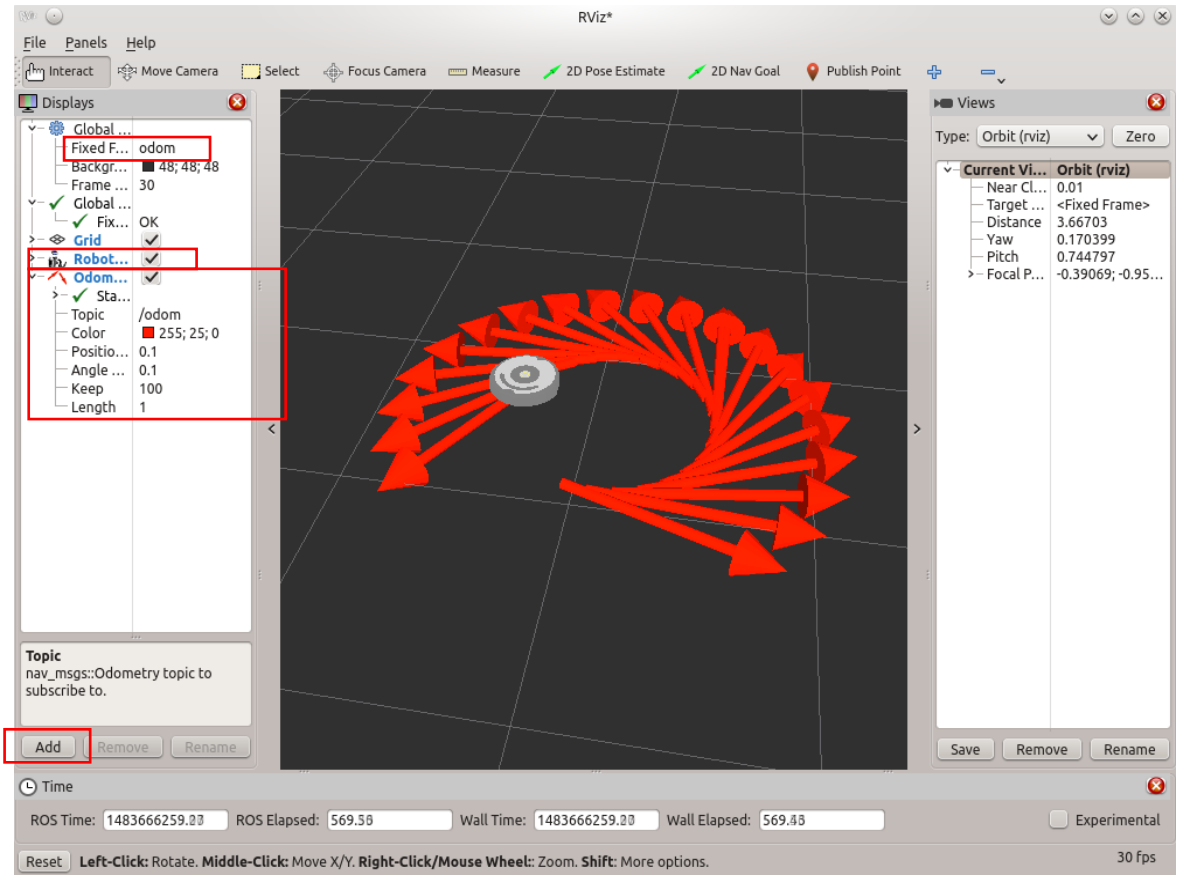
export ROS_MASTER_URI=http://<ip_address>:<port>

export ROS_IP=<**pi**_ip_address>

where <ip_address> and <port> are the same ip address and port as PC Configuration and <**pi**_ip_address> is the ip address of raspberry pi. Note that this configuration is only valid for the current logged in terminal where you export. New terminals need to be configured similarly to be able to talk to master.

3. Now if you run roscore -p <port> on PC and run other nodes in raspberry pi you will be able to see topics of nodes that you run on raspberry pi from PC and vice versa.

ROS Rviz: Another tool you will need for this part and next labs is Rviz which allows you to visualize your robot movement and its sensors data. After setting up your PC and raspberry pi to communicate over network you need to launch robot driver on raspberry pi and rviz on PC. By changing the Fixed Frame to odom and adding RobotModel to the environment you can see the simulated robot which mimics the actual robot using sensory data coming from pi. You can add odometry to see the path made by robot.



run your make_circle and make_polygon nodes and visualize the movement of the robot on Rviz.

Post Lab Questions:

Part 1:

1. Try different orders of running subscriber and publisher nodes. What are you observing? Was it expected?
2. What is the difference between ros::spin() and ros::spinOnce()? Explain?
3. How did you program to receive all messages (including the first one) sent by publisher? Explain?

Part 2:

1. Derive the equation of r=v/w and report error made by slippage by experiencing different r values. Is there any relation between r and error? Is there any other factor (e.g. velocity)? Report all your experiments.
2. Report error you have experienced in terms of distance and angle by calling the functions with different values of distance and angle of movement. Can you find a relation between the amount of movement (forward/backward) and the error? How about the amount of turning (left/right) and the error? How about combination of both movements? Find a way of reporting uncertainty for robot's movement. (Hint: use covariance concept)

Part 3:

1. Explore Rviz and describe features that it provides for you. What type of sensors can be used on robot and visualized by Rviz?
2. Run your codes on stage simulator.

Appendix: Setup up your robot

1. Each team (group of two) will be given an SD card which can be inserted to raspberry pi assembled on the Create 2 robots along with a set of sensors. You need to set YOUR password for raspberry pi after first login. The default password is "password". Open a new terminal and log in to raspberry pi assembled on your robot by typing:
   $ slogin pi@IP
   where IP is written on the raspberry pi and belongs to wifi USB dongle not to the SD card! Once logged in change the password by typing:
   $ passwd
   follow instruction and set a new password.
2. **Set up ros environment**. Every time you open a new terminal you need to set the environmental variables:
   $ source /opt/ros/<ros_distro>/setup.bash
   You can add above line to the end of ~/.bashrc so that you don't need to run it every time. After sourcing you can check by typing: $ printenv | grep ROS
3. **Create a new workspace**. You need to create new directory on raspberry pi by typing
   $ mkdir –p ~/catkin_ws/src
   where catkin_ws is the name of your workspace directory. Initialize your workspace by typing:
   $ cd ~/catkin_ws/src
   $ catkin_init_workspace
   $ cd ~/catkin_ws
   $ catkin_make
4. **Create your first ROS hello world program!**
   $ cd ~/catkin_ws/src
   $ catkin_create_pkg hello_world roscpp
   a. **If using C++:**
   Now go to src directory of the package you just built and write a hello world program into a new .cpp file (hello_world.cpp). Uncomment and edit following lines in CMakeList.txt file under the package directory.
   include_directories(${catkin_INCLUDE_DIRS})
   add_executable(hello_world_node src/hello_world_node.cpp)
   target_link_libraries(hello_world_node ${catkin_LIBRARIES})
   b. **If using Python:**
   Now go to src directory of the package you just built and write a hello world program into a new .py file (hello_world_node.py). Make your python script executable.

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

every time you create new package/node or open new terminal you need to source devel/setup.bash in your workspace directory to be able to run your nodes. The catkin_make command compiles your program. Open another terminal and run

```
$ roscore
```

This will run the ros master which will be in charge of managing nodes. Open a new terminal and run the node you just created.

If using C++:

```
$ rosrun hello_world hello_world_node
```

If using Python:

```
$ rosrun hello_world hello_world_node.py
```