CSCI 4972/6972: Algorithmic Robotics
Programming Assignment One (PA1)
Deadline: 2pm GMT-05 February 7, 2019

Goal

In this programming assignment, you'll write your first program that uses ROS. You will make and execute programs that use ROS in increasingly complex ways.

- The "Hello World!" program.
- A program that publishes random twist commands for which turtlesim listens.
- A program that subscribes to twist messages and prints them to a terminal window.
- Separately, you'll use a ROS bag to collect the velocity commands and rqt_plot to plot the commands against time.

Part 1: A Few Things to Remember

You should have already installed ROS, set up your ROS environment, and created a catkin workspace. Navigate into the root folder of your workspace (the one containing src as a subdirectory) and source its environment. Before doing that, print out the environment variables containing the string "ROS." Then do it again after sourcing the environment and notice the difference in the ROS package path environment variable.

```
$ printenv | grep ROS
$ source ./devel/setup.bash
$ printenv | grep ROS
```

Once this is done, the ROS command line tools can help you quickly navigate your workspace. If you don't want to type the source command every time you open a terminal, put it as the last line in .bashrc in your home directory. Below are some useful ROS commands. For more information, go to http://wiki.ros.org/.

rosdep update
rosls
roscd
rospack
rosnode
roscore
rosrun
rqt_graph
rostopic
rosmsg
roswtf

Remember that <tab><tab> is a useful way to complete some ROS commands. For example

$ rospack find turtle <tab><tab>

Part 2: The "Hello World!" Program

Navigate to your ROS workspace source code directory (mine is ~/ws/src/) and create a new package directory with the command

$ cd ~/ws/src
$ catkin_create_pkg hello

Two template files have been created: package.xml and CMakeLists.txt. Read OKane 3.2.1 if you want more info. Read package.xml using the linux **more** command (the **less** command is more or less (haha) equivalent).

$ cd hello
$ more package.xml

Read O'Kane 3.2.2 to learn how to add dependency information and declare executables. You should insert the following lines into the so-called ROS manifest file - package.xml.

<buildtool_depend>rospy< /buildtool_depend>
<buildtool_depend>geometry_msgs< /buildtool_depend>
<buildtool_depend>turtlesim< /buildtool_depend>
<exec_depend>rospy< /exec_depend>
<exec_depend>geometry_msgs< /exec_depend>
<exec_depend>turtlesim< /exec_depend>

By clicking on the blue box, open hello.py and study its contents.

$ more hello.py

Run hello. You need two terminals: one to run roscore and one to run hello.

In terminal 1: **$ source ~/ws/devel/setup.bash**
In terminal 1: **$ roscore**

In terminal 2: **$ source ~/ws/devel/setup.bash**
In terminal 2: **$ python ~/ws/src/hello/hello.py**

Open a third terminal and test some of the ROS commands.
In your report for this assignment, write the output of each of the following commands (while the previous two terminals are still running).

In terminal 3: **$ source ~/ws/devel/setup.bash**
In terminal 3: **$ rosnode list**
In terminal 3: **$ rostopic list**

If you didn't edit your .bashrc file as suggested above, you are probably annoyed with typing **source ~/ws/devel/setup.bash** over and over. If so, put that command as the last line in your ~/.bashrc now. Then every time you open a new terminal, setup.bash will be sourced automatically.

Part 3: A ROS Rublisher Program

Next you will use the pubvel.py to control the turtle in turtlesim. This is done by publishing geometry_msgs/Twist messages, to which turtlesim subscribes. The source code is linked to the blue box containing "pubvel.py" pubvel.py and is listed below for your convenience.

```
#!usr/bin/env python

''' This program publishes randomly-generated velocity
messages for turtlesim. '''

import rospy
from geometry_msgs.msg import Twist
import random

# Initialize the ROS system and become a node.
rospy.init_node('publish_velocity')

# Create a publisher object
pub = rospy.Publisher('turtle1/cmd_vel', Twist, queue_size=1000)

# Loop at 2Hz until the node is shut down.
r = rospy.Rate(2)
while not rospy.is_shutdown():
    # Create and fill the message. The other fields
    # will default to 0.
    msg = Twist()
    msg.linear.x = random.random()
    msg.angular.z = 2*random.random() - 1

    # Publish the message.
    pub.publish(msg)

    # Send a message to rosout with the details
    rospy.loginfo("Sending random velocity command: "
                  "linear = %.2f "
                  "angular = %.2f", msg.linear.x, msg.angular.z)

    # Wait until it's time for another iteration.
    r.sleep()
```

Run pubvel.py while turtlesim is running and answer the following questions.
1. Describe the behavior of the turtle under the control of the original pubvel.
2. Modify pubvel to make the turtle move on a circle. How could you make the circle larger?
3. What is the code's intended rate of the publishing loop in pubvel?
4. What is the actual rate of publishing during execution? (The command rostopic hz topicname will be helpful)

Part 4: A ROS Subscriber Program

The last part of this assignment is to give you experience with ROS subscriber code. Specifically, we'll subscribe to the /turtle1/pose topic, which is published by turtlesim after it processes a twist message. The code is linked to the blue box subpose.py and listed below.

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

''' This program subscribes to turtle1/pose and shows its
messages on the screen.
'''

import rospy
from turtlesim.msg import Pose

def poseMessageReceived(msg):
    ''' A callback function. Executed each time a
        new pose message arrives. '''

    rospy.loginfo("position=(,%.2f, %.2f) "
                  "direction=%.2f", msg.x, msg.y, msg.theta)

# Initialize the ROS system and become a node.
rospy.init_node('subscribe_to_pose')

# Create a subscriber object.
rospy.Subscriber('turtle1/pose', Pose, poseMessageReceived)

# Let ROS take over.
rospy.spin()
```

Run subpose.py while turtlesim is running and answer the following questions.
5. What is the purpose of rospy.spin() in this code?
6. Use a ROS bag to collect the twist messages and plot them with rqt_plot. Plot x-position vs time and x-velocity vs time. Based on your plots, comment on whether turtlesim correctly computes position from velocity.
7. After runing the simulator, use rtq_plot to plot y-position vs x-position. Compare this with a screenshot from the turtle simulator to show that your plot is correct.

Submitting

This programming assignment will have a short write-up that should be included in the zip file with your code, which you send to trinkle@gmail.com by the deadline. I expect your write-up to be about three to four pages long (that long only because of the plots and pictures you might want to include). Make sure to include an acknowledgment statement describing any resources that you used or assistance that you received. The format of your submission is:

- Please name the zip file you submit as: pa1-*name*, where *name* is replaced by your name.
- Your name, date, programming assignment title
- List the questions asked above and give your answers
- List and briefly describe useful things you discovered, that go beyond the questions of this assignment.
- Format your assignment as a PDF file and email to trinkle@gmail.com

Grading

Submissions will be graded according to the following rubric:

Clarity, correctness, and thoroughness of your answers	85%
Legibility of your document	15%