CMPSC 383
Multi-Agent and Robotic Systems
Spring 2017
Janyl Jumadinova

In-class Exercise
5–7 November 2019
DUE: Tuesday, November 12th

Starting with ROS

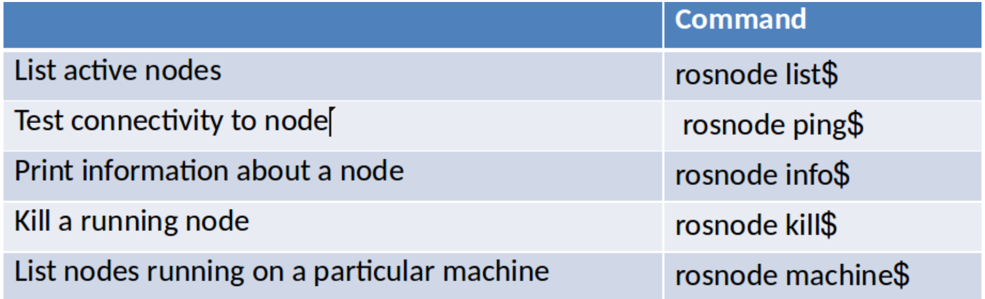
1. In the terminal, you may need to open your bashrc file: `vim ∼/.bashrc` and put the following command that sets up ROS environment: `source /opt/ros/kinetic/setup.bash`. Otherwise, you may need to type: `source /opt/ros/kinetic/setup.bash` every time you open a new terminal.

2. Type `$ roscore`. *roscore* is the first thing you should run when using ROS. *roscore* will start up: a ROS Master, a ROS Parameter Server, and a rosout logging node.

3. Next, you can run *rosrun* command. *rosrun* allows you to run a node. Usage: **rosrun <package> <executable>**. Open a new terminal window and type:
   ```
   $ rosrun turtlesim turtlesim_node
   ```

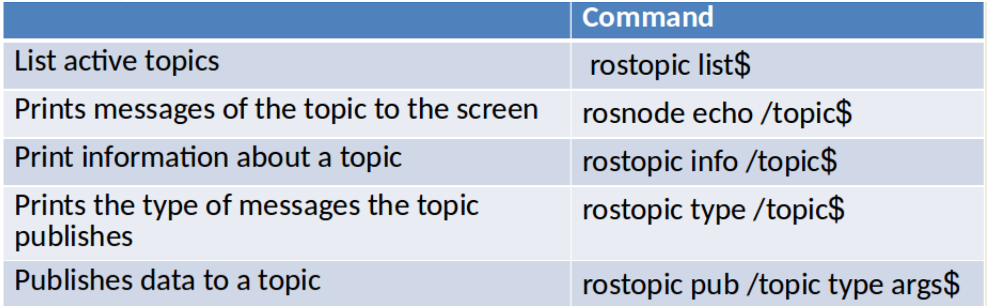
4. In separate terminal window run: `$ rosrun turtlesim turtle_teleop_key`

5. *rosnode* displays debugging information about ROS nodes, including publications, subscriptions and connections.



	Command
List active nodes	rosnode list$
Test connectivity to node[	rosnode ping$
Print information about a node	rosnode info$
Kill a running node	rosnode kill$
List nodes running on a particular machine	rosnode machine$

6. *rostopic* gives information about a topic and allows to publish messages on a topic.



	Command
List active topics	rostopic list$
Prints messages of the topic to the screen	rosnode echo /topic$
Print information about a topic	rostopic info /topic$
Prints the type of messages the topic publishes	rostopic type /topic$
Publishes data to a topic	rostopic pub /topic type args$

7. Use the `rostopic pub` command to publish messages to a topic. For example, to make the turtle move forward at a 0.2m/s speed, you can publish a cmd_vel message to the topic /turtle1/cmd_vel:
   ```
   $ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist ’linear: x: 0.2, y: 0, z:
   0, angular: x: 0, y: 0, z: 0’
   ```
   Or to specify only the linear x velocity:
   ```
   $ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist ’linear: x: 0.2’
   ```

8. Some of the messages like cmd_vel have a predefined timeout. If you want to publish a message continuously use the argument -r with the loop rate in Hz. For example, to make the turtle turn in circles continuously, type:
   ```
   rostopic pub /turtle1/cmd_vel -r 10 geometry_msgs/Twist ’angular: z: 0.5’
   ```

ROS Topics

This information was taken from: `http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics`

1. In the terminal, run `roscore`

2. We will again use the turtle simulation. In the new terminal, run:
   ```
   rosrun turtlesim turtlesim_node
   ```

3. We need something to drive the turtle around. In the new terminal, run:
   ```
   rosrun turtlesim turtle_teleop_key
   ```

4. Now, let’s see what’s going on behind the scenes. The two nodes: *turtlesim_node* and the *turtle_teleop_key* node, are communicating with each other over a ROS Topic. *turtle_teleop_key* is publishing the key strokes on a topic, while *turtlesim* subscribes to the same topic to receive the key strokes. We can use *rqt_graph* to display the nodes and topics that are currently running. *rqt_graph* creates a dynamic graph of what’s going on in the system. In the new terminal, run:
   ```
   rosrun rqt_graph rqt_graph
   ```
   You should see that the *turtlesim_node* and the *turtle_teleop_key* nodes are communicating on the topic named */turtle1/command_velocity*.

5. The *rostopic* tool allows you to get information about ROS topics. To see the available commands/topics for the *rostopic*, in the new terminal, type:
   ```
   rostopic -h
   ```

6. *rostopic echo* shows the data published on a topic. Usage: `rostopic echo [topic]`. To observe the command velocity data published by the *turtle_teleop_key* node, in the new terminal, run:
   ```
   rostopic echo /turtle1/cmd_vel
   ```
   You probably won’t see anything happen because no data is being published on the topic. Let’s make *turtle_teleop_key* publish data by pressing the arrow keys. If the turtle isn’t moving you need to select the *turtle_teleop_key* terminal before pressing keys.

7. Go to the *rqt_graph* again. Press the refresh button in the upper-left to show the new node. You should see *rostopic* is now also subscribed to the *turtle1/cmd_vel* topic.

8. Communication on topics happens by sending ROS messages between nodes. For the publisher (*turtle_teleop_key*) and subscriber (*turtlesim_node*) to communicate, the publisher and subscriber must send and receive the same type of message. The type of the message sent on a topic can be determined using *rostopic type*. Usage: `rostopic type [topic]`. In the new terminal, run:

   ```
   rostopic type /turtle1/cmd_vel
   ```

   You can view the details of the message (e.g. message *geometry_msgs/Twist*) using `rosmsg`:

   ```
   rosmsg show geometry_msgs/Twist
   ```

9. Now, we will use rostopic with messages. *rostopic pub* publishes data on to a topic currently advertised. Usage: `rostopic pub [topic] [msg_type] [args]`. For example, to send a single message to *turtlesim* telling it to move with an linear velocity of 2.0 and an angular velocity of 1.8, run the following command in the terminal:

   ```
   rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- ‘[2.0, 0.0, 0.0]’ ‘[0.0, 0.0, 1.8]’
   ```

   Argument description: *rostopic pub* publishes the message, -1 will only publish one message, name of the topic to publish to is */turtle1/cmd_vel*, message type is *geometry_msgs/Twist*, double-dash tells the option parser that none of the following arguments is an option, *geometry_msgs/Twist* msg has two vectors of three floating point elements (x,y,z) each: linear and angular making up the last argument.

10. During the run of the previous command, the turtle has stopped moving; this is because the turtle requires a steady stream of commands at 1 Hz to keep moving. We can publish a steady stream of commands using *rostopic pub -r* command:

    ```
    rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- ‘[2.0, 0.0, 0.0]’ ‘[0.0, 0.0, -1.8]’
    ```

11. While the turtle is going in circles, we can update the *rqt_graph* and see new communications.

12. While the turtle is going in circles, we can also use *rostopic hz* to find the rate at which data is published. Usage: `rostopic hz [topic]`. Run:

    ```
    rostopic hz /turtle1/pose
    ```

13. To visualize the data in running time, we can use *rqt_plot*, which displays a scrolling time plot of the data published on topics. We'll use *rqt_plot* to plot the data being published on the */turtle1/pose* topic. First, start *rqt_plot*:

    ```
    rosrun rqt_plot rqt_plot
    ```

    The text box in the upper left corner gives us the ability to add any topic to the plot. Typing `/turtle1/pose` will highlight the plus button, previously disabled. Press it and using plus and minus sign vary the displayed graphs.

14. In ROS, *tf* is a special topic that keeps track of coordinate frames, and how they relate to each other. So, our simulated turtle starts at (0,0,0) in the world coordinate frame. When the turtle moves, its own coordinate frame changes. Generally, anything on the robot that is not fixed in space, will have a *tf* describing it. In the *rqt_graph* section, you can see that the

tf topic is published to and subscribed from by many different nodes. One intuitive way to see how the *tf* topic is structured for a robot is to use the *view_frames* tool provided by ROS. In a new terminal window, type:

```
rosrun tf view_frames
```

Wait for this to complete, and then type in:

```
evince frames.pdf
```

Submit this diagram (**frames.pdf**) to your class participation repository.

ROS Package

This information was taken from: Creating a ROS Package. Here you will learn how to use the `catkin_create_pkg` script to create a new `catkin` package.

1. Change to the source space directory of the catkin workspace you created during the last class:

   ```
   cd ~/catkin_ws/src
   ```

2. Now use the `catkin_create_pkg` script to create a new package called ‘communication’ which depends on `std_msgs,` `roscpp`, and `rospy`:

   ```
   catkin_create_pkg communication std_msgs rospy roscpp
   ```

   This will create a *communication* folder which contains a *package.xml* and a *CMakeLists.txt*, which have been partially filled out with the information you gave *catkin_create_pkg*.

3. Now you need to build the packages in the catkin workspace:

   ```
   cd ~/catkin_ws
   catkin_make
   ```

4. To add the workspace to your ROS environment you need to source the generated setup file:

   ```
   . ~/catkin_ws/devel/setup.bash
   ```

Simple Publisher and Subscriber

This information was taken from: Writing a Simple Publisher and Subscriber. Here you will create a publisher (“talker”) node which will continually broadcast a message and a subscirber (“listener) which will continually listen to the broadcasted message.

1. Change directories to your *communication* package you just created in the previous step:

   ```
   roscd communication
   ```

2. Create a *src* directory in the *communication* package directory:

   ```
   mkdir src
   ```

3. Copy the `talker.cpp` and `listener.cpp` programs from the shared repository into the `/communication/src/` directory.

4. Find `CMakeLists.txt` file in the communication package and add the following few lines to the bottom of your `CMakeLists.txt`:

   ```
   include_directories(include ${catkin_INCLUDE_DIRS})

   add_executable(talker src/talker.cpp)
   target_link_libraries(talker ${catkin_LIBRARIES})
   add_dependencies(talker communication_generate_messages_cpp)

   add_executable(listener src/listener.cpp)
   target_link_libraries(listener ${catkin_LIBRARIES})
   add_dependencies(listener communication_generate_messages_cpp)
   ```

5. Now run `catkin_make`:

   ```
   cd ~/catkin_ws
   catkin_make
   ```

6. Now, we can run the publisher and the listener. First, start `roscore`:

   ```
   roscore
   ```

7. Make sure you have sourced your workspace's `setup.sh` file after calling `catkin_make` but before trying to use your applications. In a new terminal:

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

   To avoid typing this command in every new terminal, I suggest that you add the command to your *bashrc* file:

   ```
   vim ~/.bashrc
   source ~/catkin_ws/devel/setup.bash
   ```

   Remember to restart the terminal after modifying the *bashrc* file.

8. Now, run the publisher:

   ```
   rosrun communication talker
   ```

9. Now we need a subscriber to receive messages from the publisher. In a new terminal:

   ```
   rosrun communication listener
   ```

   Remember if you did not add your `setup.sh` sourcing command to your bashrc file, you will need to run the following before you are able to run listener:

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

10. Submit a screenshot if your *talker* and *lister* running to your course participation repository.