EECE 5554 ROS Tutorial+
This doc summarizes and further explains the ROS beginner tutorial. All the links are preserved in case you want to revisit a step. There are other helpful links in here for your background knowledge.

1. Installing and Configuring Your ROS Environment
    a. You installed ROSand set up the ROS environment variables using `source /opt/ros/noetic/setup.bash` Environment variables in Linux and Unix are operating system variables that we use as aliases for paths (file locations), which programs to use to complete a task, etc. We need to set the ROS environment variables to tell Linux where to look for packages that we'll use (and other stuff).
    b. You created and built a catkin workspace. Catkin "builds" programs, scripts, and libraries from code. Catkin extends CMake (other software for generating build files) to manage ROS package dependencies. *(Why the name Catkin? It's a tribute to the now-defunct company that created it, Willow Garage)*

    Catkin workspaces have a few components:
    - catkin_ws/src contains source code for the packages you want to use
    - catkin_ws/build is where you'll invoke CMake to build from code in the catkin_ws/src space
    - catkin_ws/devel is where build targets go before they're installed
    - catkin_ws/install is where targets can be installed with `make install` after being built

It's ok if you don't understand everything related to packages, dependencies, and builds *yet*—you'll get it soon!

2. Navigating the ROS Filesystem

This step took you through a few commands in rosbash, which are commands you can use in a BASH shell (one type of command line interpreter in Linux) along with ROS. You learned about:
    a. `rospack find roscpp`, which returned the path of the ROS C++ package. You can use `rospack find` to get the path to other packages. `rospack` does more than `find`—check out other options here.
    b. `roscd`, which changes directory (`cd`) to where the package you're interested in is located
    c. `rosls`, which lists (`ls`) in a package directly, rather than ls to the directory

Still not sure what all this Linux and command line stuff does? Check out the Linux tutorial posted to Piazza!

3. Creating a ROS Package

This step covered roscreate-pkg or catkin to create a new package, and rospack to list package dependencies.
    a. You created a new package called beginner tutorials in the `catkin_ws` workspace you created in step 2 using `catkin_creage_pkg`. Your package dependencies (the packages you use) are `std_msgs rospy and roscpp`
    b. You built beginner_tutorials package using `catkin_make`
    c. You used `rospack depends1 beginner_tutorials` to verify the dependencies that you listed when you made the new package
    d. You then used `rospack depends beginner_tutorial` to see ALL package dependencies (including the dependencies of the packages on which your beginner_tutorial are dependent)
    e. You modified the package.xml file (the package manifest) using an editor like nano to update the description, maintainer, and license tags, as well as to add `<exec_depend>` to our package dependencies to make sure they're available at run time

4. Building a ROS Package

Inside your catkin workspace is the directory `src`. You used `catkin_make` to build everything in the `src` directory (in this case, `beginner_tutorial`). Your first build is complete!

Up to this point, the tutorial has spent a lot of time on packages, dependencies, build tools, and the Linux environment. If you're confused, you likely need to do a bit more reading and studying (start with the links above) about ***Linux*** rather than ROS. If everything is clear, let's move onto more ROS-specific topics!

5. Understanding ROS Nodes

The tutorial gives the definition of a node as:

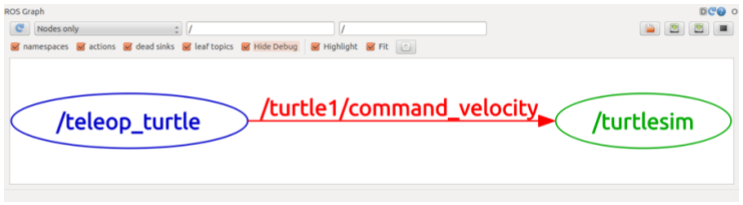
> "… a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes. For example, one node controls a laser range-finder, one node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provides a graphical view of the system, and so on."

a. You ran `roscore` and opened a new terminal, since the `roscore` process will need to keep running in your first terminal.
b. You ran `rosrun turtlesim turtlesim_node` and renamed the `turtlesim_node` to a different name

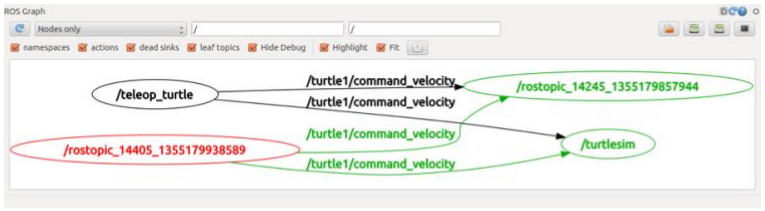
This step introduced a few different nodes: `rosout, turtlesim_node`, and `ping`. Each node does computation and can publish (send) information to a topic to send information to other nodes or receive (subscribe) to a topic to get information from other nodes.

6. Understanding ROS Topics

a. You used rosrun rqt_graph rqt_graph to see what node was publishing (/teleop_turtle), what node was subscribing (/turtlesim), and the topic name (/turtle1/cmd_vel).



b. You used `rostopic echo /turtle1/cmd_vel` to print the messages being published to the rostopic `/turtle1/cmd_vel` to the terminal. We can now see a second node (`/rostopic…`) that is subscribing to the topic `/turtle1/cmd_vel` in the rqt_graph (image not copied here, check the tutorial to see).
c. A rosmsg is a defined data type. Here, you sent a message (geometry_msgs/Twist) that is defined as three float64 values that contain the x, y, and z linear velocities of the turtle, as well as the three float64 values that contain the x, y, and z angular velocities.
d. In step 4.1, you published a geometry_msgs with the turtle velocities to the topic `/turtle1/cmd_vel`, just as you had done earlier with the teleop node and your keyboard. We now see the rqt_graph has been updated to reflect that two nodes are publishing messages, and two nodes are subscribing to both the publisher nodes


ROS Graph
Nodes only
namespaces
actions
dead sinks
leaf topics
Hide Debug
Highlight
Fit
/teleop_turtle
/turtle1/command_velocity
/turtle1/command_velocity
/rostopic_14245_1355179857944
/turtle1/command_velocity
/rostopic_14405_1355179938589
/turtlesim
/turtle1/command_velocity

e. You also used `rqt_plot` to look at the data that were being published to a particular ros topic. rqt_graph and rqt_plot are helpful tools to see what data is being sent where.

7. Understanding ROS Services and Parameters

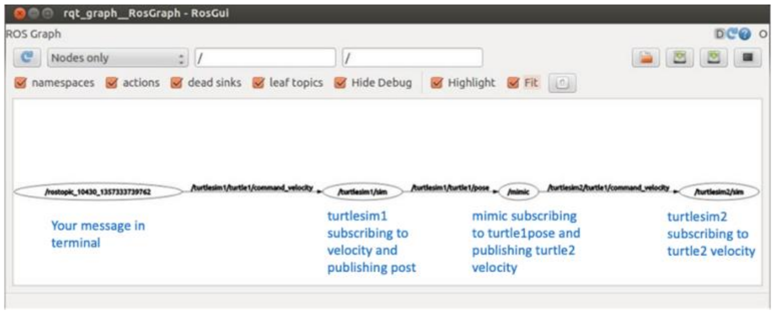
From the tutorial about services:

"The publish / subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for RPC request / reply interactions, which are often required in a distributed system. Request / reply is done via a *Service*, which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Client libraries usually present this interaction to the programmer as if it were a remote procedure call."

a. You called two services to clear the first turtle's path and to spawn a second turtle.
b. You then edited the parameter server to change the background color of the turtles' screen. Nodes can store (set) and retrieve (get) parameters from this server. The parameters can be stored in a .yaml file to easily save (dump) and load parameter sets.

8. Using rqt_console and roslaunch
Note: We use rqt_console and roslaunch here. These topics aren't *particularly* connected, but the RQT tools are really useful to understand the node graph. RQT (i.e., ROS Qt) is a suite of GUI tools for ROS.

a. You used `rqt_console` and `rqt_logger_level` to get information in the console and logger about your turtlesim nodes. `rqt_logger_level` lets you use a GUI to define what logs are posted to the `rqt_console` about each node. Once you changed the logger_level to warn and ran the turtle into the wall, the console told you the warning that the turtle had hit the wall. You could also try this again after changing the logger_level back to info to see what happens.
b. You also used roslaunch to start (launch) your turtle node. The launch file you made launches two turtles and remaps the turtle1 output to the turtle2 input. When you published the geometry_msgs/Twist message to the turtle1/cmd_vel topic, it also controlled turtle2, as shown in the node graph below.


rqt_graph__RosGraph - RosGui
ROS Graph
Nodes only
/
/
namespaces
actions
dead sinks
leaf topics
Hide Debug
Highlight
Fit
/rostopic_10430_1357333739762
/turtlesim1/turtle1/command_velocity
/turtlesim1/sim
/turtlesim1/turtle1/pose
/mimic
/turtlesim2/turtle1/command_velocity
/turtlesim2/sim
Your message in terminal
turtlesim1 subscribing to velocity and publishing post
mimic subscribing to turtle1pose and publishing turtle2 velocity
turtlesim2 subscribing to turtle2 velocity

9. Using rosed to edit files in ROS
This tutorial showed how to use rosed to make editing easier. You can also use nano for editing files.

10. Creating a ROS msg and srv
a. You created a msg (a file that describes the format of a ROS message). The command places "int64 num" into the message file Num.msg within the msg directory you created.
b. You edited the package manifest (package.xml) to add the package dependencies of `message_generation` (during build) and `message_runtime` (during execution).
c. You edited CMakeLists.txt to 1) add `message_generation` to the `find_package` call so that it's added when you build, 2) added your message file Num.msg to `add_message_files`, and 3) uncomment `generate_messages` so it gets called.
d. You used `rosmsg msg/Num.msg` to verify you could see the new message you created.
e. You made a directory and copied an existing service, then edited the package manifest to add build and execution dependencies of `message_generation` and `message_execution`.
f. Just as above, you also added `message_generation` to the `find_package` call in CMakeLists.txt so it is built, and you uncommented the `add_service_files` call and added your service AddTwoInts.srv.
g. Finally, you used rossrv (similar to rosmsg above) to confirm you can see the new service.

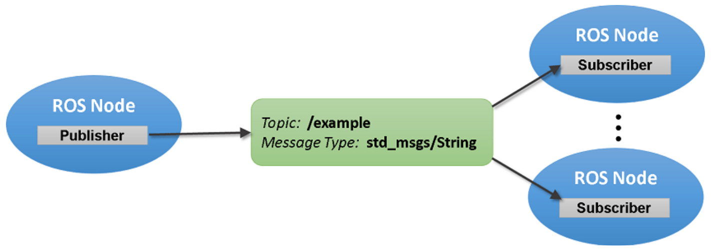
The common steps on the page for srv and msg are helpful reminders of what we need to do when we make new messages or services!

So far, the tutorial has defined ROS nodes, topics, messages, and services. This is a lot of terminology, so you might take a few minutes to write out short definitions to help cement these concepts in your brain. The next section will detail how to write publisher and subscriber nodes and writing services and clients.

11. Writing a Simple Publisher and Subscriber (Python)
This step in the tutorial breaks down the publisher node code nicely, but here are some critical points to note about the publisher node code:

- The line `#!/usr/bin/env python` is necessary to indicate a Python script
- `import rospy` (and potentially other libraries) come next
- The talker() function: (lines 6-14)
  - defines a Python function that takes in no inputs
  - tells ROS that it will be a publisher node with the name talker publishing at 10 Hz
  - Publishes the string "hello world" and the system time. (string formatting may be useful here)
  - Sleeps for the specified rate

- The subscriber node goes through similar steps:
  - Initializing the listener node with a random name
  - Subscribing to the chatter topic
  - Calling the callback function with input data and putting the hello world string into the log
- After writing the publisher and subscriber, you need to build your nodes

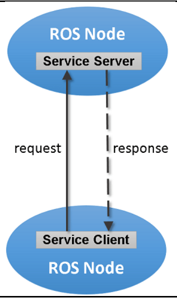


An example graph showing the relationship between publishers and subscriber nodes

12. Examining the Simple Publisher and Subscriber
This tutorial examines running the simple publisher and subscriber.

15. Writing a Simple Service and Client (Python)
This step in the tutorial breaks down the subscriber and client node code nicely, but here are some critical points to note:

- The client sends the service two integers
- The service/server node receives two integers, adds them together, and returns the sum of the two integers
- The client prints the addition of the two integers.
- As with your publisher/subscriber, you need to make your .py files executable, edit your CMakeList.txt, and re-build



16. Examining the Simple Service and Client
This tutorial examines running the simple service and client.

14. Recording and playing back data
This tutorial covered how to record data from a running ROS system into a .bag file. Specifically, please note:
```
rosbag record -O subset /turtle1/cmd_vel /turtle1/pose
```

which will allow you to record a subset of published topics into a rosbag file. subset is the rosbag file name

15. Reading messages from a bag file
Covers two ways to read messages from desired topics in a bag file, including the `ros_readbagfile` script.