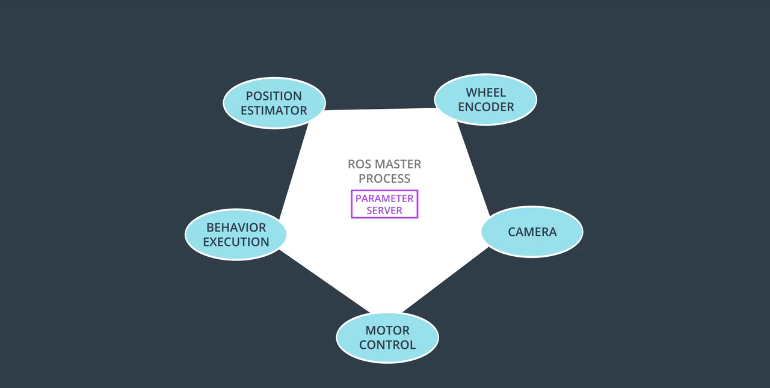
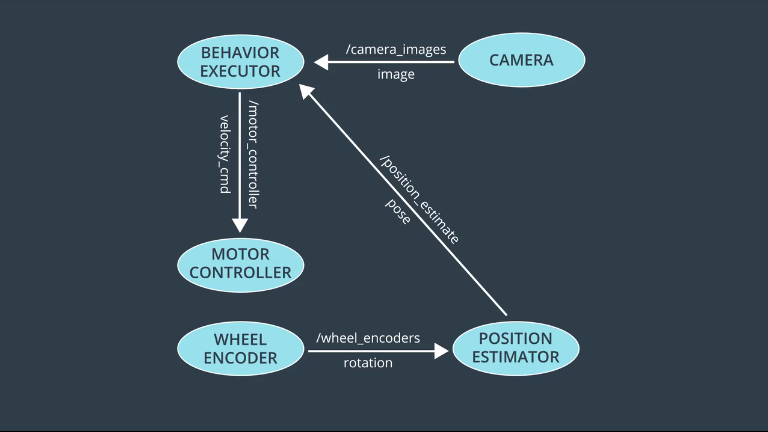
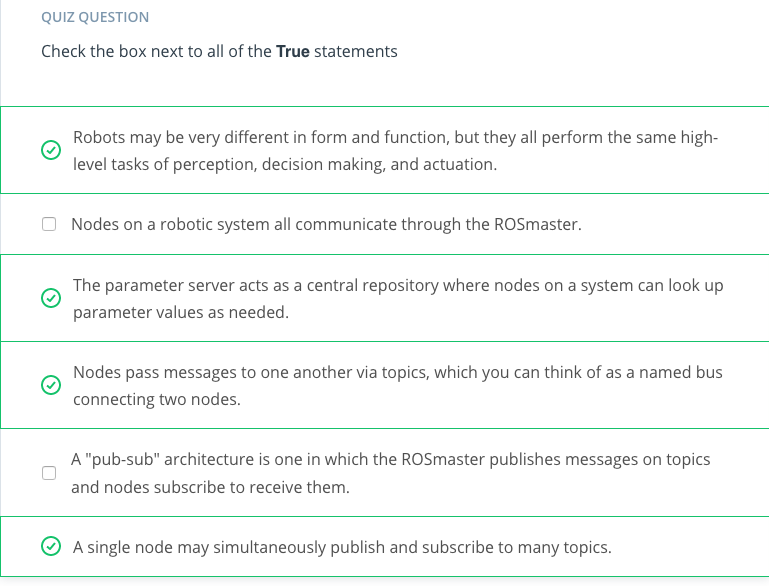
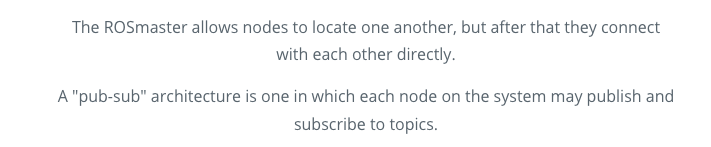
1. Nodes and Topics







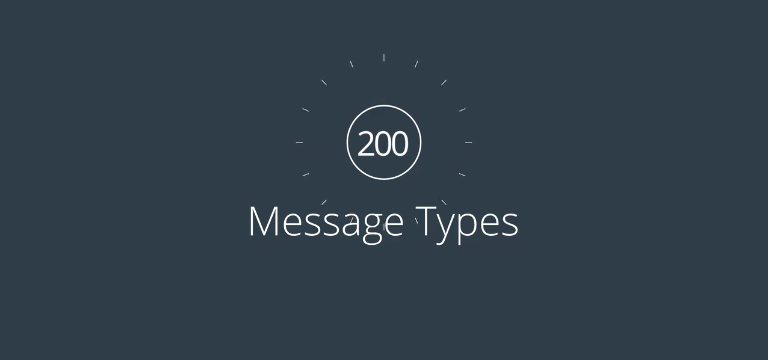


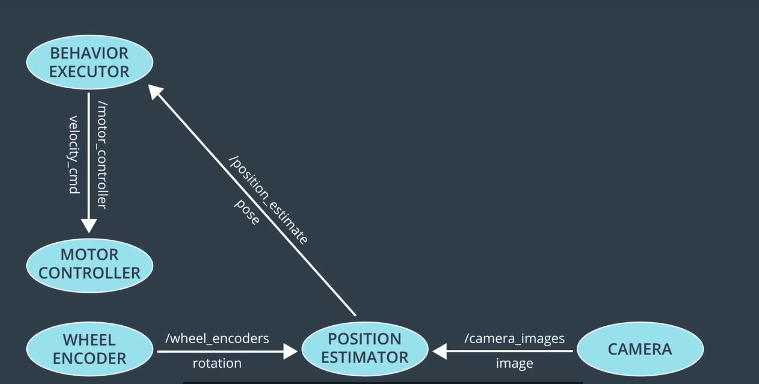


1. Message Passing

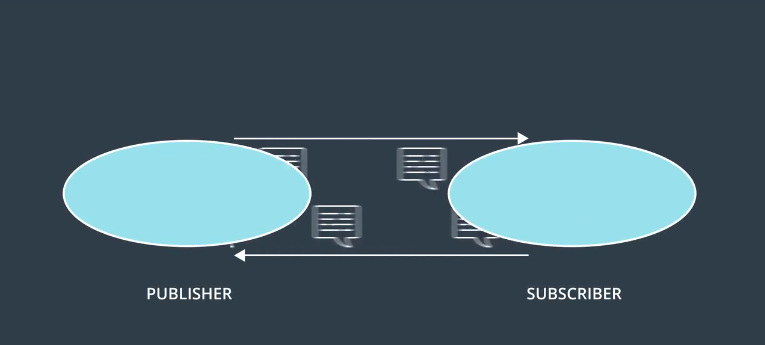


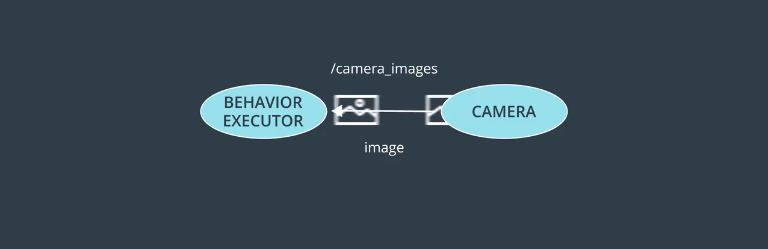


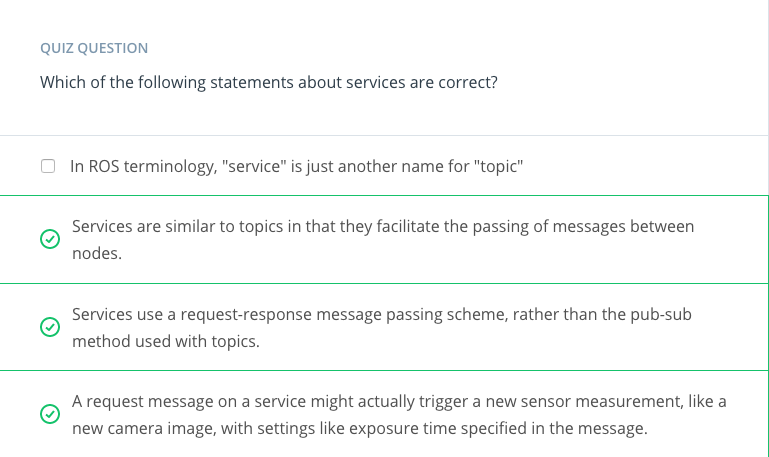




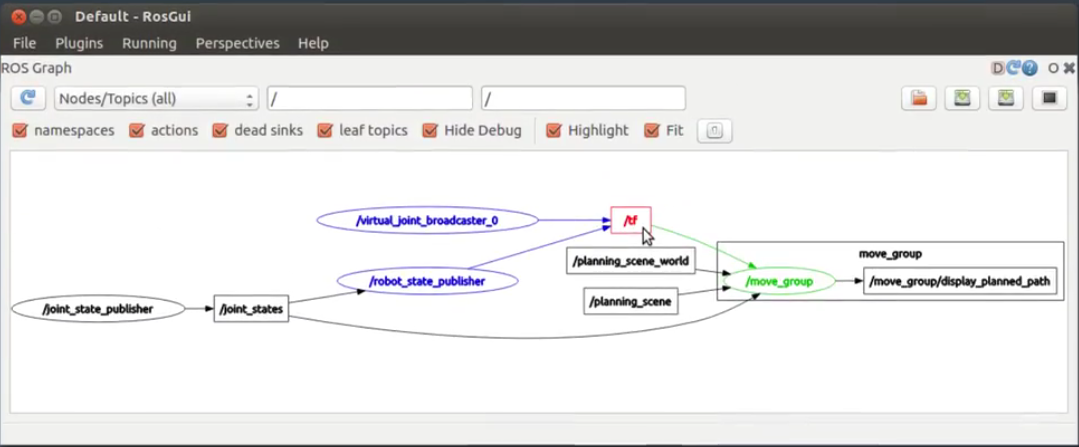
1. Services

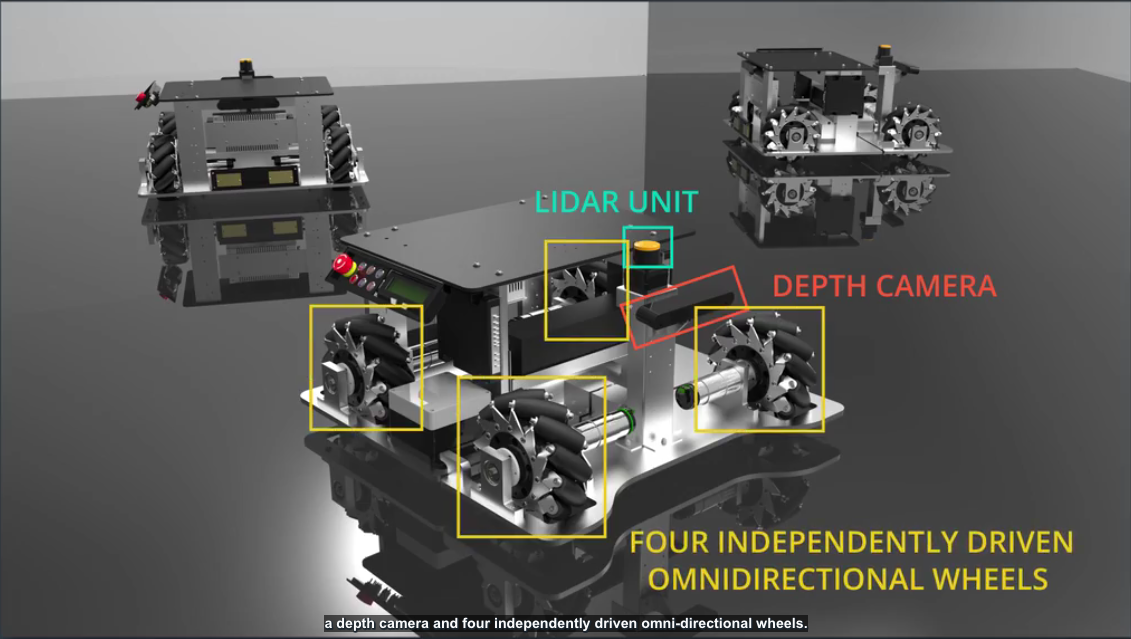






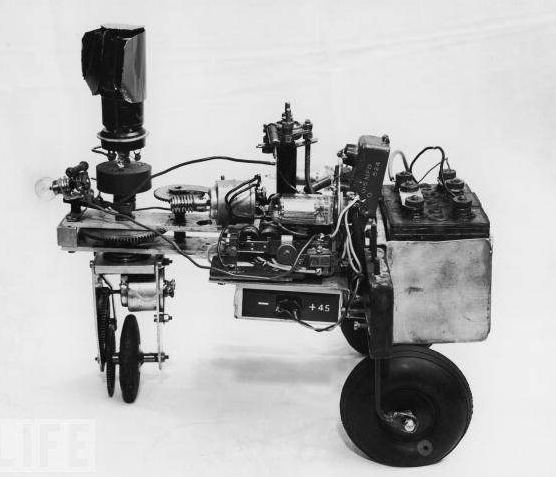
1. Compute Graph





1. Turtlesim Overview





Elsie without her protective cover

1. ROS Workspace Instructions

Throughout system integration, we will be working in ROS. To facilitate this and minimize local environment issues we are providing a GPU workspace environment. You are welcome to use this or the provided VM, but please note that support emphasis will be on the provided workspace. The workspace in the next concept is identical to that provided for the final project and contains a fully functional ROS environment. As is the case for many other workspaces. Please bookmark this workspace and use it throughout system integration lessons.

**To activate and use the ROS environment:**

* Open the workspace and enable GPU
* Click on the desktop button
* open a terminal on the desktop and follow along

Feel free to explore and have fun!

1. Your Virtual Machine

Udacity virtual download link:

<http://www.7-zip.org/download.html>

Import your VM image to VirtualBox

* Download and install VirtualBox.
* Download the image from supplied link.( <https://s3-us-west-1.amazonaws.com/udacity-selfdrivingcar/Udacity_VM_Base_V1.0.0.zip>)
* Unzip the image.
* Open VirtualBox Application.
* Click File > Import Appliance..
* Click the folder icon on the right and navigate to your unzipped image (the .ovf file).
* Follow the prompts to import the image.
* From the VirtualBox Manager, select your VM and press start.

Before getting your VM up and running, you will need to make sure that you allocate enough resources to allow it to operate efficiently. With your VM shut down, navigate to the VM settings by clicking on the icon in the VirtualBox Manager. In the "System" cateogory Look for tabs labeled "Motherboard" and "Processor"; this is where you will change the amount of RAM and number of cores that you allocate from your own machine to the VM. If you have the resources on your machine, we recommend allocating at least 2 processor cores and 4 GB of RAM. If you are able to allocate more resources, feel free to do so. The more you allocate the better performance you will get! You're now all set to boot up your VM!

Upon your first boot you will be prompted to choose a keyboard layout of your choice. Once you select a keyboard you will not be prompted for this again. If you would like to change your option you can reset this feature by entering udacity\_reset in a terminal of your choice and restarting the VM.

Once you are up and running, you might be asked to input a password to enter the VM. **The password for the VM is udacity-nd**

To open a terminal in your VM press ctrl-Alt-t (or ctrl-option-t on a Mac). You should get a terminal window that looks like this:

## Troubleshooting Tips

* **Keyboard Mappings:** Use of certain keyboards can create issues unless the corresponding keyboard has been set in the VM. This is due to keyboard mappings. A frequent issue is special characters in passwords not being entered correctly when logging in. An example useage for VirtualBox is setting up an Italian keyboard. To do this, execute the following in a terminal localectl set-keymap it; localectl set-x11-keymap it.
* **roscore ip:** If the host network interface has multiple addresses (ex: ipv6 enabled) roscore will fail since hostname -I returns multiple ip, resulting into a invalid URL. One solution to this is to replace this line in .bashrc, export ROS\_IP=`echo $( hostname -I)' , with this export ROS\_IP=$( hostname -I | awk '{print $1}').

1. Source the ROS environment

Before we begin using ROS in a terminal, we must first ensure that all of the environment variables are present. To do this, we must source the setup script provided by ROS:

source /opt/ros/kinetic/setup.bash

1. Run Turtlesim

# Starting the Master process

Before any ROS nodes can be run, the Master process must be started.

The Master process is responsible for the following (and more)

* Providing naming and registration services to other running nodes
* Tracking all publishers and subscribers
* Aggregating log messages generated by the nodes
* Facilitating connections between nodes

To run the master process, you must execute the command roscore. If all goes well (and it should), you will see an output similar to following:

# 

# Running Turtlesim Nodes

Now that the ROS master is running, we can run our first two ROS nodes.

To do so, we will execute the rosrun command in a new terminal window, passing as parameters the name of the package we wish to run, and the name of the actual node.

Note:

Tab completion is your friend. Each ROS distribution comes with a staggering number of packages, and an even more staggering number of nodes. In the bash shell, a single-tap of the tab key will cause the command on the command-line to be completed, if there is a single match. A double-tap of the tab key will result in a list of all possible matches, in the case that a single match cannot be found.

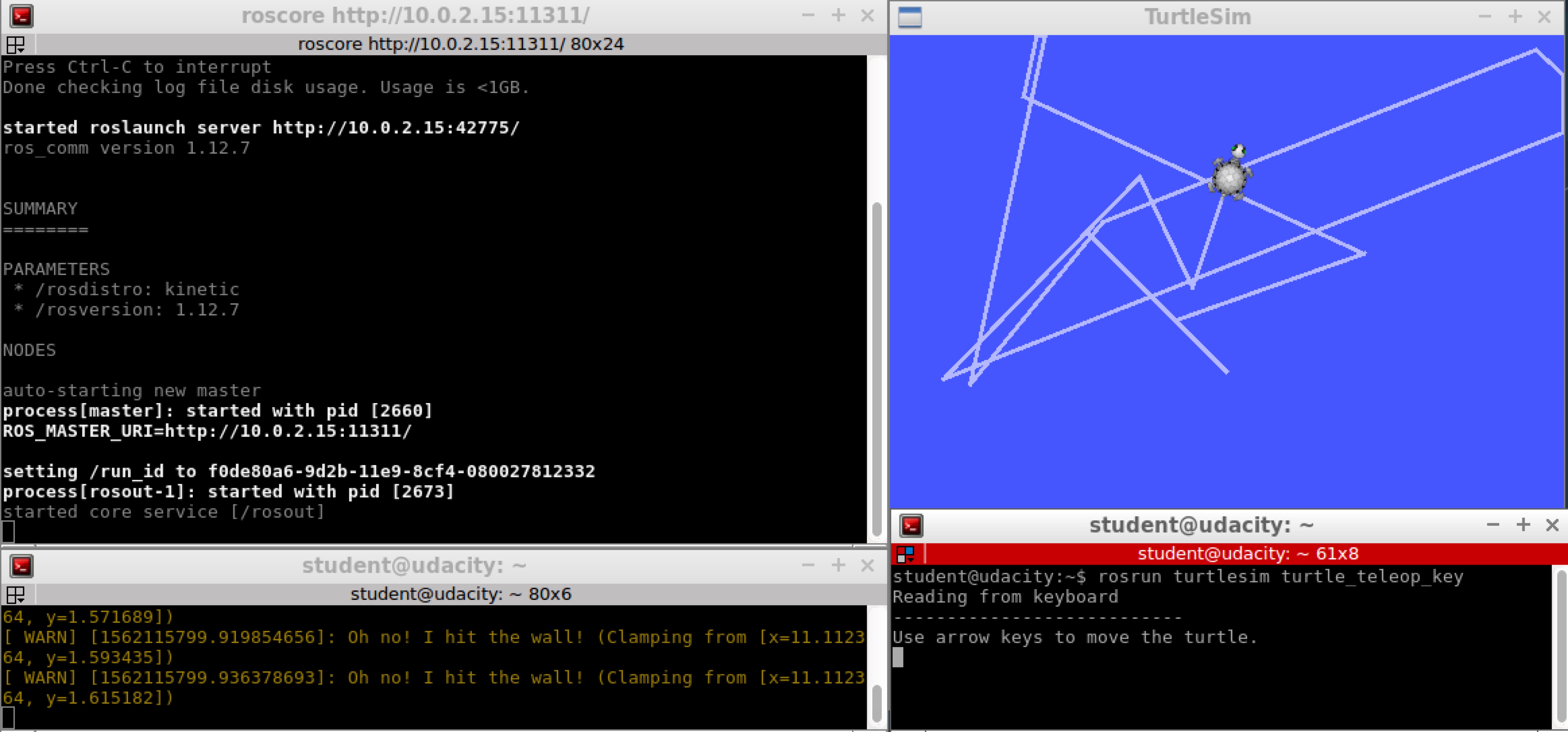
First we will start the turtlesim\_node, in the turtlesim package using the following command.

$ rosrun turtlesim turtlesim\_node

Next, we will start the turtle\_teleop\_key node, also from the turtlesim package.

$ rosrun turtlesim turtle\_teleop\_key

By using the arrow keys with the turtle\_teleop\_key node’s console selected, we are able to move the turtle in turtlesim!



Summary:

* 1. Starting master process by:

roscore

* 1. Open new terminal window and start turtlesim Nodes:

rosrun turtlesim turtlesim\_node

* 1. Open new terminal window and start turtlesim Nodes:

rosrun turtlesim turtle\_teleop\_key

* 1. Use arrow keys to move the turtle in the turtlesim

1. Turtlesim Comms:List Nodes

In the following concepts, we will investigate Turtlesimm Comms, and will cover the following steps:

* Listing all active nodes
* Listing all topics
* Getting information about topics
* Showing message information
* Echoing messages in real-time

## Now that we’ve launched turtlesim\_node, and played around with sending commands via the turtle\_teleop\_key node, let’s dig deeper, to see what’s actually happening underneath the surface... Listing all Active Nodes

To get a list of all nodes that are active and have been registered with the ROS Master, we can use the command rosnode list. Let’s do so now:

We can see that there are three active nodes that have been registered with the ROS Master, /rosout, /teleop\_turtle, and /turtlesim.

* /rosout This node is launched by roscore. It subscribes to the standard /rosout topic, the topic to which all nodes send log messages.
* /teleop\_turtle This is our keyboard teleop node. Notice that it’s not named turtle\_teleop\_key. There’s no requirement that a node’s broadcasted name is the same as the name of it’s associated executable.
* /turtlesim The node name associated with the turtlebot\_sim node

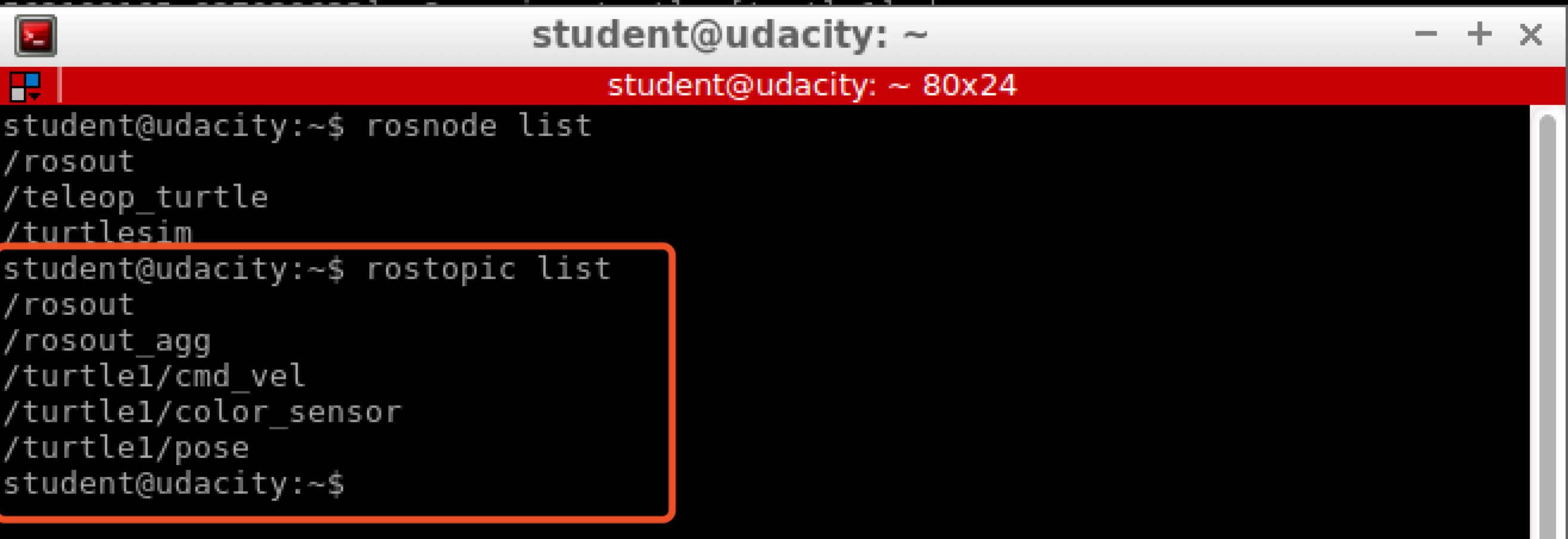
Summary:

1. To get a list of all nodes that are active and have been registered with the ROS Master by:

rosnode list

1. other
2. Turtlesim Comms:List Topics

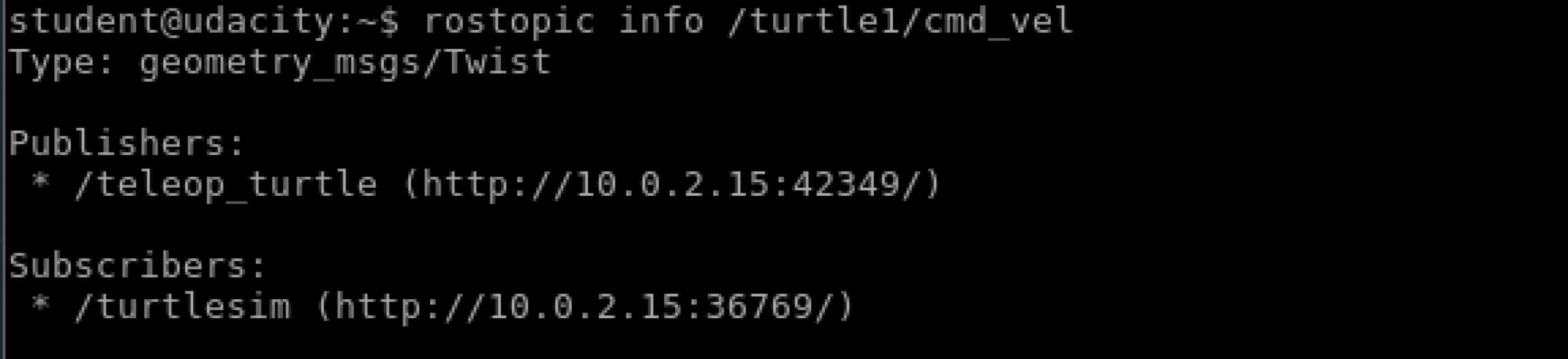
In a similar fashion, we are able to query the ROS Master for a list of all topics. To do so, we use the command rostopic list.



* /rosout\_agg Aggregated feed of messages published to /rosout.
* /turtle1/cmd\_vel Topic upon which velocity commands are sent/received. Publishing a velocity message to this topic will command turtle1 to move.
* /turtle1/color\_sensor Each turtle in turtlesim is equipped with a color sensor, and readings from the sensor are published to this topic.
* /turtle1/pose The position and orientation of turtle1 are published to this topic.

1. Turtlesim Comms:Get Topic Info

If we wish to get information about a specific topic, who is publishing to it, subscribed to it, or the type of message associated with it, we can use the command $rostopic info. Let’s check into the /turtle1/cmd\_vel topic:



As would be expected, there are two nodes registered on this topic. Our publisher, the teleop\_turtle node, and our subscriber, the turtlesim node. Additionally, we can see that the type of message used on this topic is geometry\_msgs/Twist.

Summary:

* 1. Get information about a specific topic by:

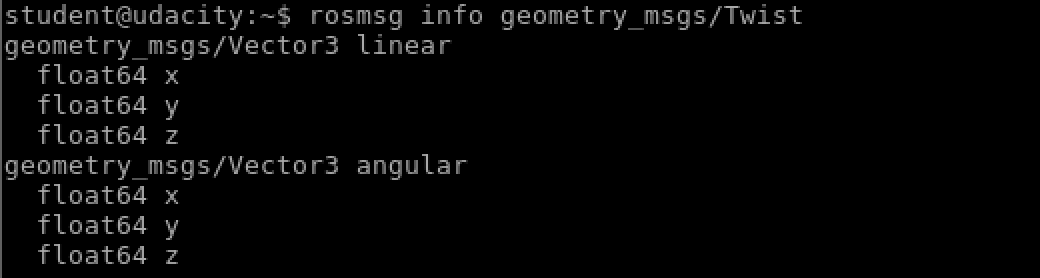
Rostopic info [one of “rostopic list” item]

Example: rostopic info /turtle/cmd\_vel

* 1. other

1. Turtlesim Comms:Message Information

Let’s get some more information about the geometry\_msgs/Twist message on the /turtle1/cmd\_vel topic, to do so, we will use the rosmsg info command.



We can see that a Twist message consists nothing more than two Vector3 messages. One for linear velocity, and another for angular velocities, with each velocity component being represented by a float64.

**Note 1**: Sometimes, the message definition won’t provide an ample amount of detail about a message type. For example, in the example above, how can we be sure that linear and angular vectors above refer to velocities, and not positions? One way to get more detail would be to look at the comments in the message’s definition file. To do so, we can issue the following command: rosed geometry\_msgs Twist.msg.

**Note 2:** More information about rosed, including how to select which editor is used by default can be found [here](http://wiki.ros.org/ROS/Tutorials/UsingRosEd).

Summary:

* 1. Get more detail information about message by:

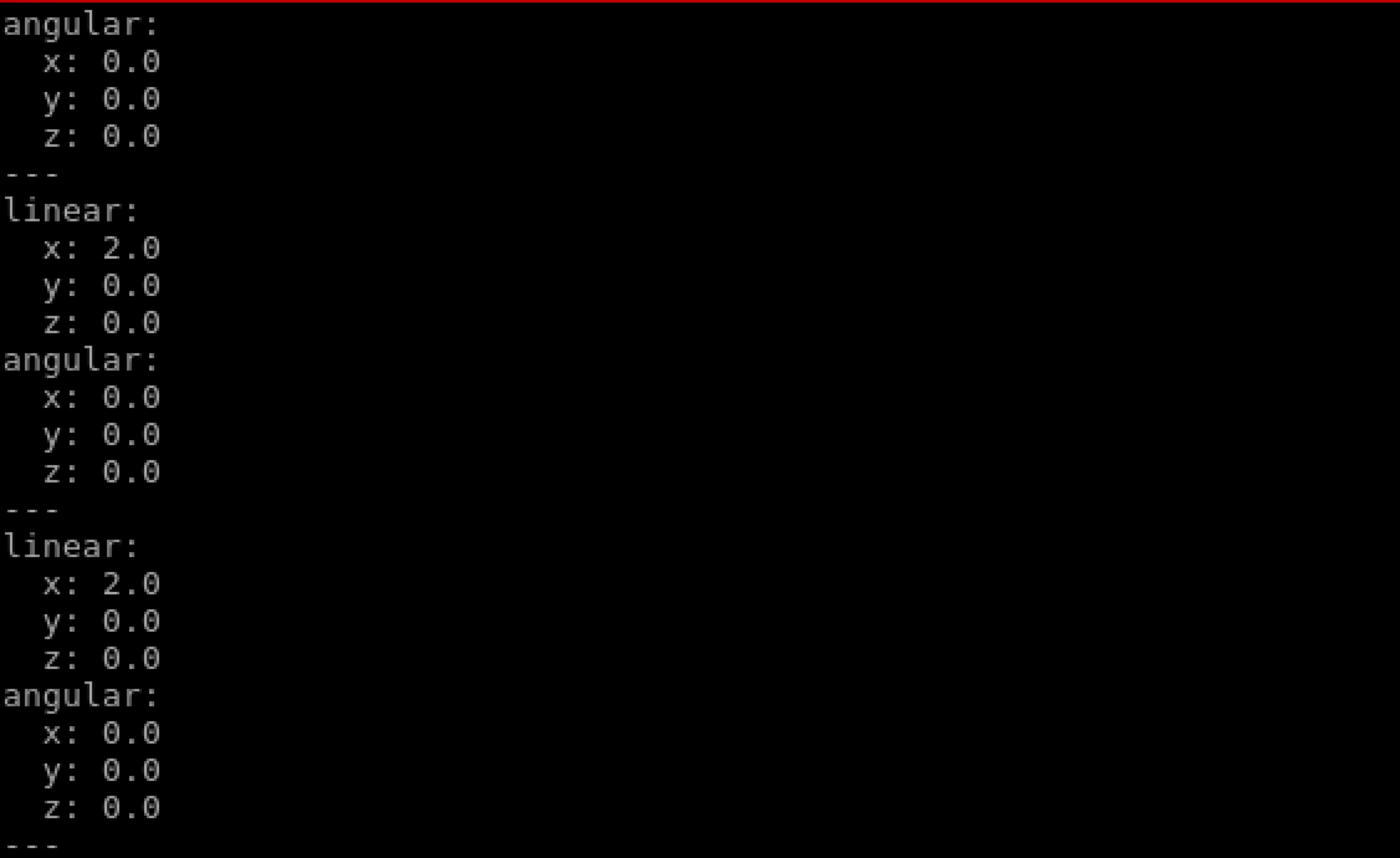
rosmsg info [type of message]

* 1. other

1. Turtlesim Comms:Echo a Topic

Rostopic echo [specific topic]

Example: rostopic echo /turtle1/cmd\_vel



1. Recap