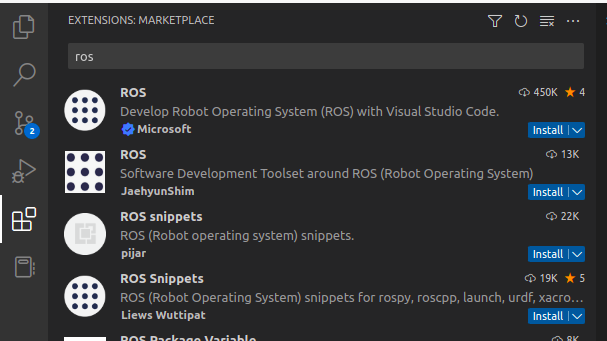
Video link: <https://www.youtube.com/watch?v=0-zTU4_S6vY>

Launch files are very important component of ROS. We need to have a very good understanding of lauch files in order to understand the bigger picture of the projects in ROS.

The subcomponents of launch files:

* Node launching
* Arguments
* Parameters
* Mapping
* Grouping
* Name spaces
* If conditions (through groups)

Before moving any forward, you need to install 4 plugins in visula studio which are great for ROS development. Here is the snippet:



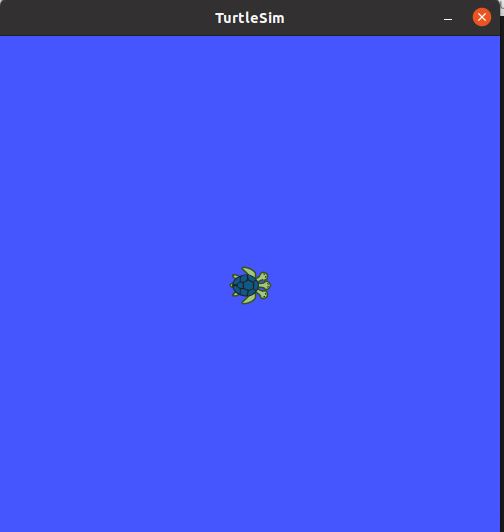
**a). Node Launching**

What we’re gonna see first is the basic usage of launch file and that is to run nodes. Not just one single nodes, but multiple nodes, or maybebe even thousands of nodes. After learning this, we will not need to run multiple nodes in terminals; one launch file can run all of those nodes.

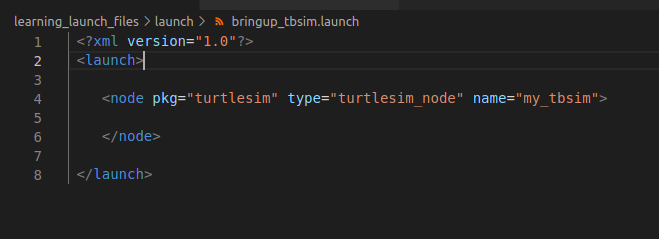
For example, one very common node that we know in ROS is:

$ rosrun turtlesim turtlesim\_node

This takes the whole terminal and opens a new window (see below):



Anyways the launch file for running this single node would look like:



*pkg* contains the rospack where the node is located

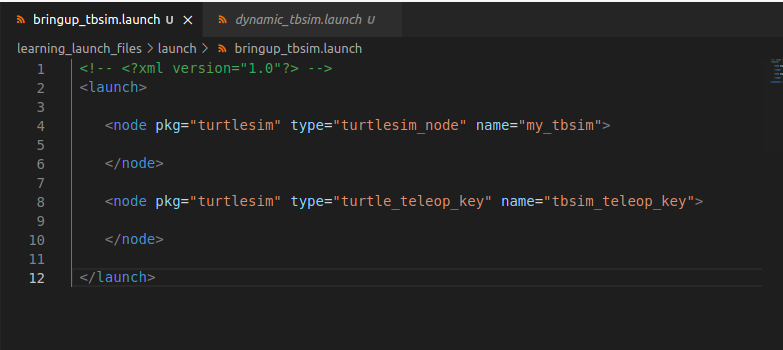
*type* is the name of node

*name* can be anything

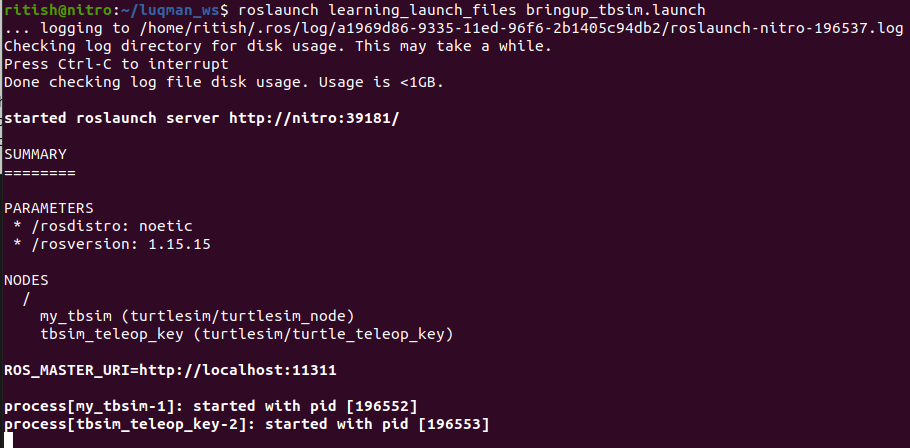
Now, to operate the turtlesim, we need to teleoperate it. So, to run a node for teleoperating it, run the following command in terminal:

$ rosrun turtlesim turtle\_teleop\_key

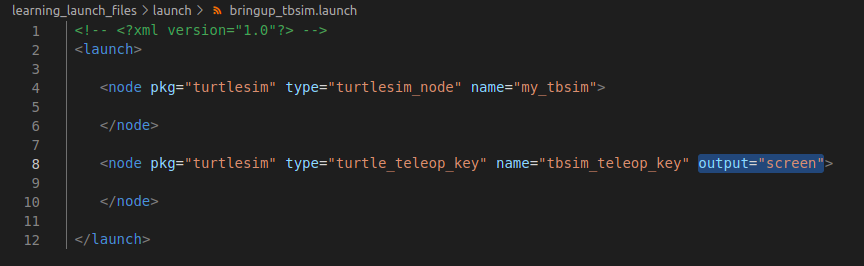
This basic process requires us to use two different terminals. However, when using robots with complex systems, we need to run multiple nodes and we don’t want to do that. So, what we can do to make it simpler is that we can bring up another node in our launch file.



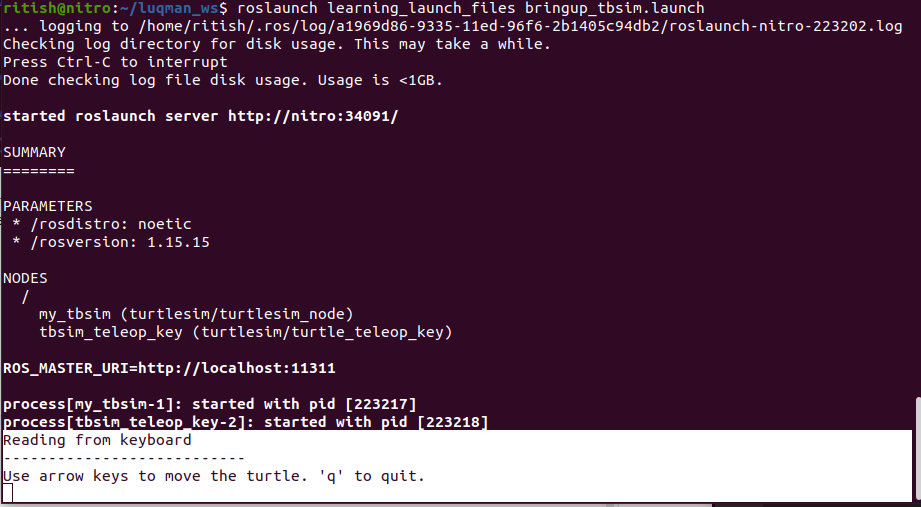
Running this would show the following output in terminal:



But it doesnot provide us the instructions as they are usually provided by running the teleop node. To see that output, we need to add one more component to the launch file.



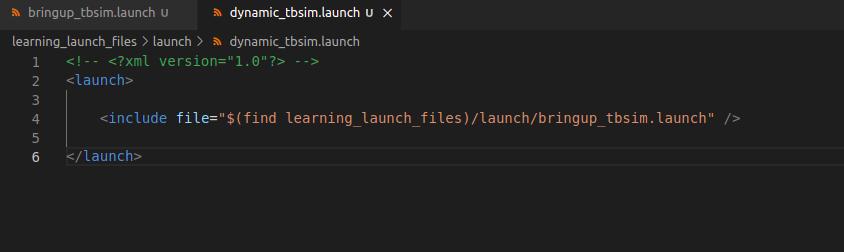
Running this updated launch file will give us the following output:



This was just the basic stuff. So, bear with me, and let’s get to the serious stuff:

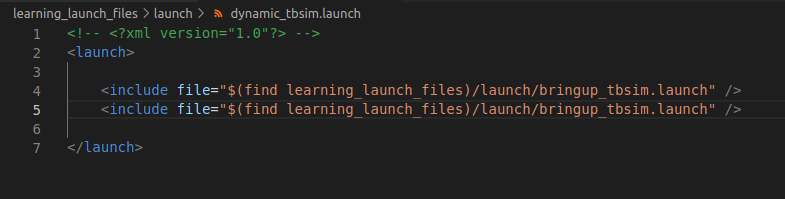
How to include other launch files within a current launch file:

This is how you can include another launch file:

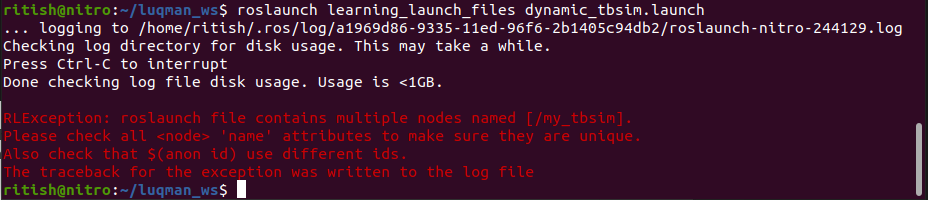


Now, what if you include the same launch file twice by accident? Will it run the same way as including it just once? Well, let’s see.

Here is my new launch file:



And this is the output:



In lamen terms, ROS is not stupid. It will verify that you’re not running the same node twice.

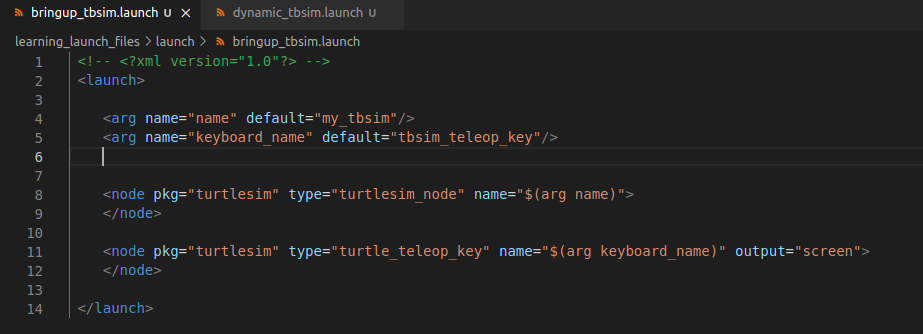
**b). Arguments:**

Now, what are arguments in ROS launch files?

Arguments are something that can be replaced with a value. To make an analogy, they are like the variables of a programming language.

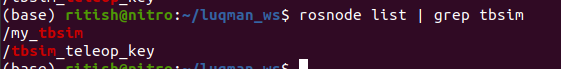
Let’s explain this better with an example. Shall we?

Do you remember how we made our first launch file to run two different nodes? Well, we are gonna replace the names of those nodes using arg feature of launch file. One nice thing about arg feature is that we can use set a default value for it.



Notice how we used $(arg *name\_of\_arg*) for naming the nodes.

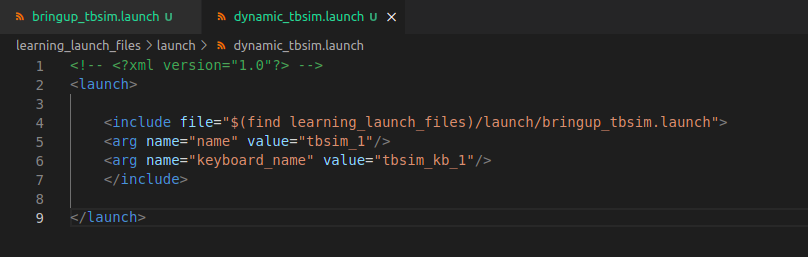
You can verify these default names by using the following command:



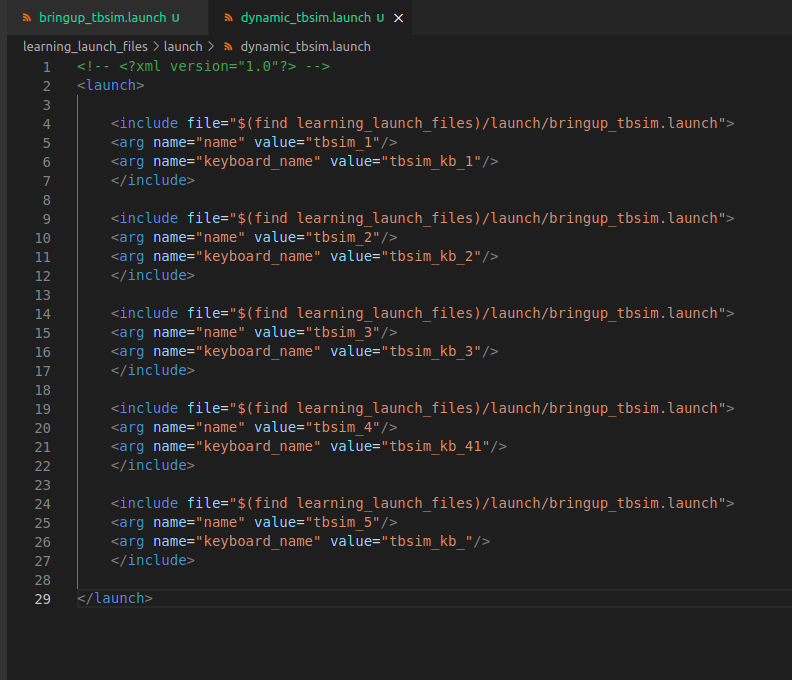
**c). Parameters**

Now let’s try using a value different than the default value.

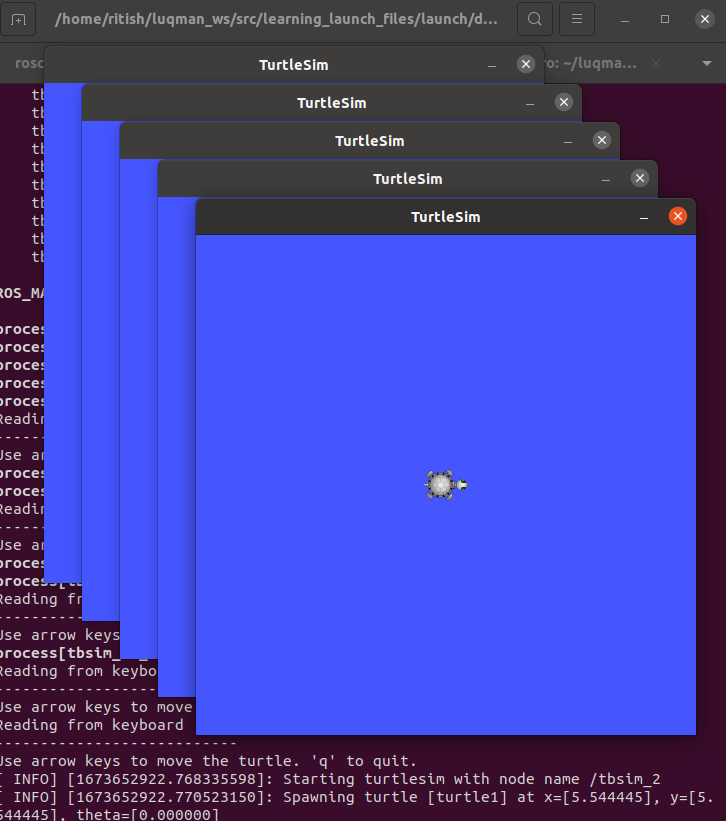
Use an argument inside <include> in a different launch file to open the above launch file. Remember to add arguments before you close </include>



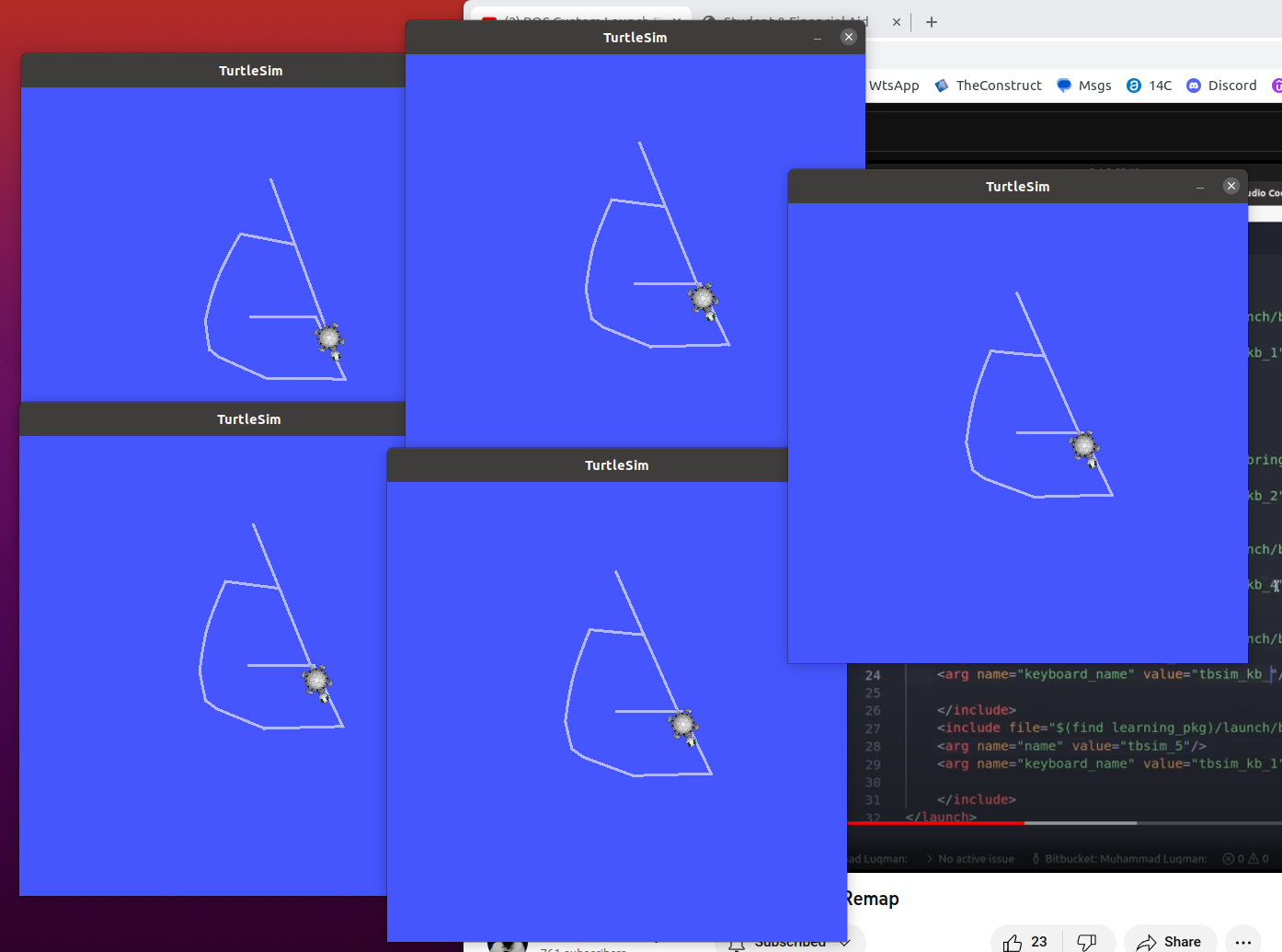
Now, let’s try opening the first launch file multiple times in the second launch file but with different arg names:



Running the above file would output the following:



Please note that the teleop commands received by each turtlesim node will be the same as the same terminal provides output for all turtlesim.



Even if we go to a different terminal and publish to the topic of just our first node, all the simulations will show the same result as they all share the same topic.

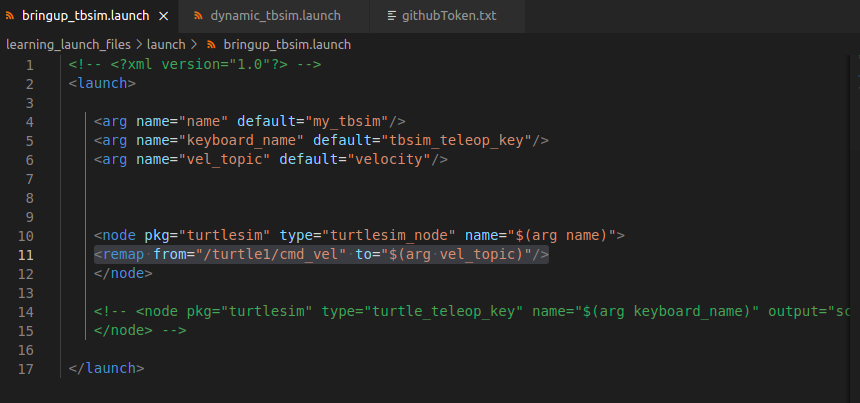
If we want to avoid all the simulations having the same input, we should use remapping and groups. These are really important if we want to work with multiple robots.

**d). Mapping:**

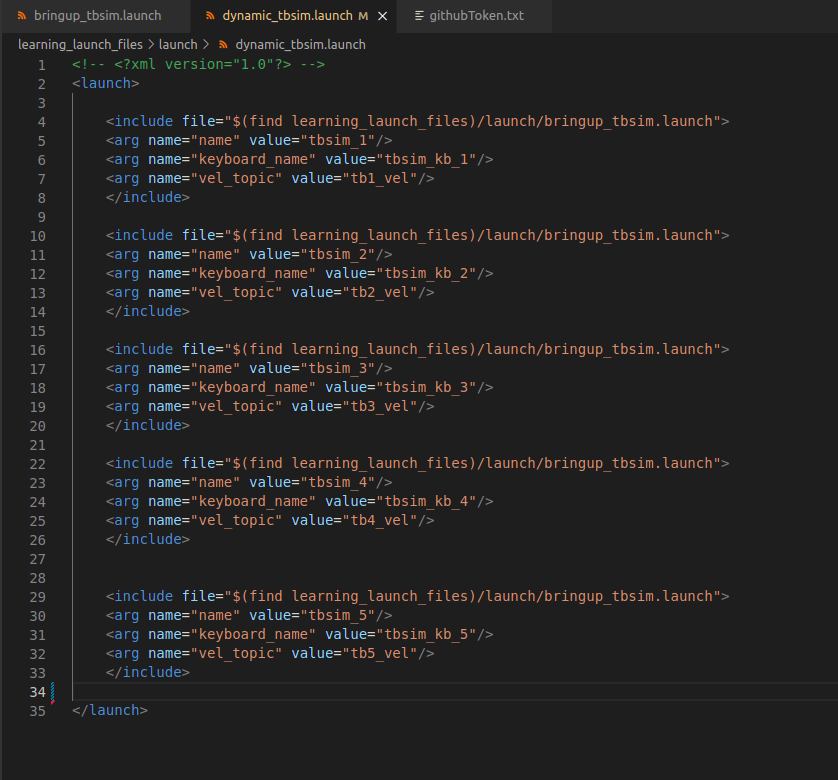
The topic belonging to a node can be mapped to a different cuson topic using the following line in launch file:

$ <remap from="/topic\_to\_map" to="$(arg new\_custom\_topic\_variable)"/>

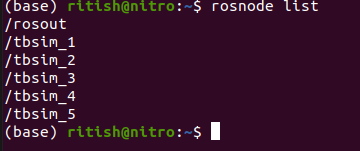
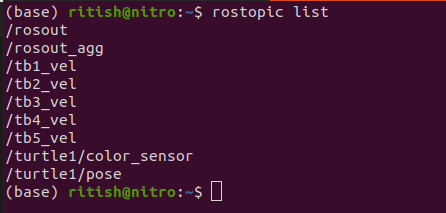
Here is an example from bringup\_tbsim.launch file

****

This way we can also make multiple nodes with each node corresponding to a unique custom topic.

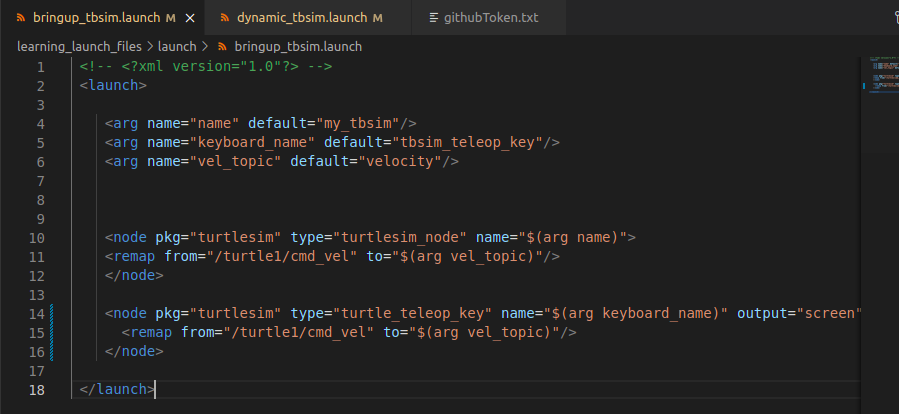


To verify run ‘rostopic list’ command on the terminal.

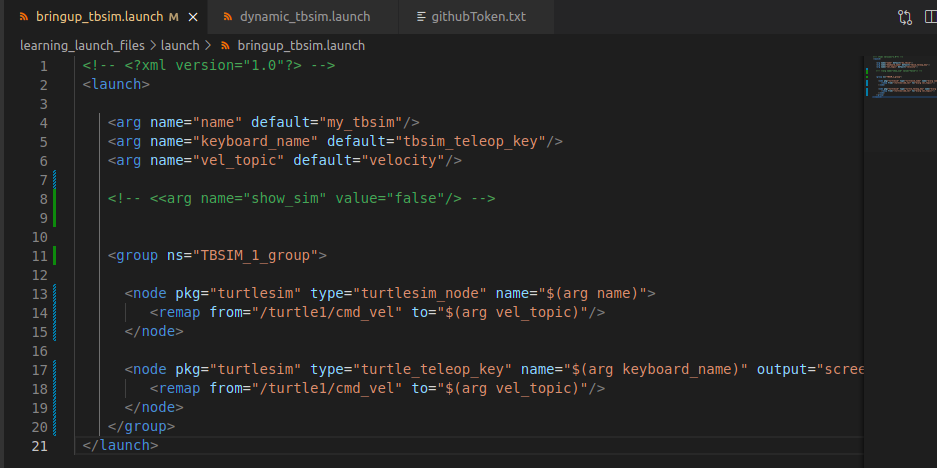


**e). Grouping:**

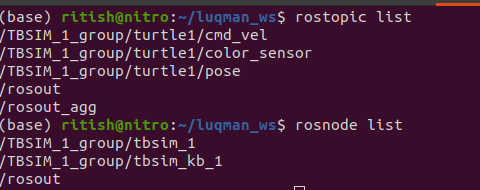
Alright, Alright, yes we can map a node’s topic to a custom topic. But, how can we publish into those custom topics? We can do that by grouping the previous topic being published to this new custom topic

****

Now let’s put both these publisher and subscriber into a group.



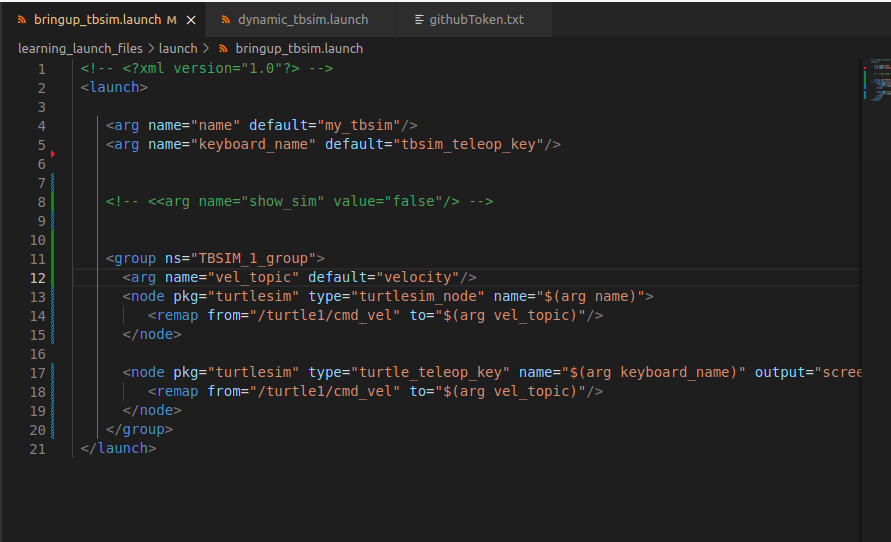
Running this launch file would activate the topics and nodes as shown below:



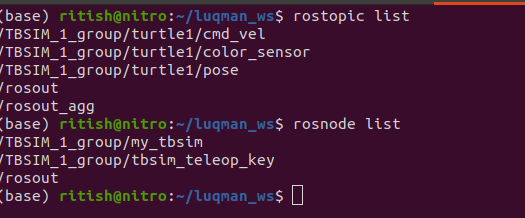
Please note the addition of TBSIM\_1\_group in the topics and nodes. In addition, the topics which were supposed to be remapped earnier are not remapped now as ‘vel\_group’ argument could not be found inside the group.

So if we place the vel\_topic argument inside the group it should work. Let’s verify.

Updated launch file:

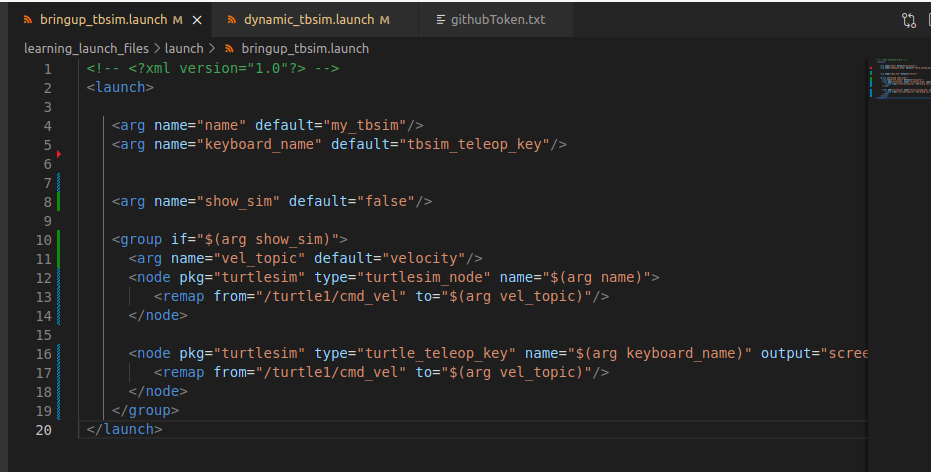


Here are the topics and nodes:



**f). If statements**

We can activate the whole group using if statements as well.



It would not run the simulation if we do not set the value of the *show\_sim* to be true.

Therefore, we need to add the following line to the other launch file in order to run the simulation.

<arg name="show\_sim" value="true"/>

**g). Parameters and ROS Params**

First we need to understand the difference between parameters and arguments

Parameters are represented in this manner:

<param name="" value=""/>

Arguments are represen=ted in this manner:

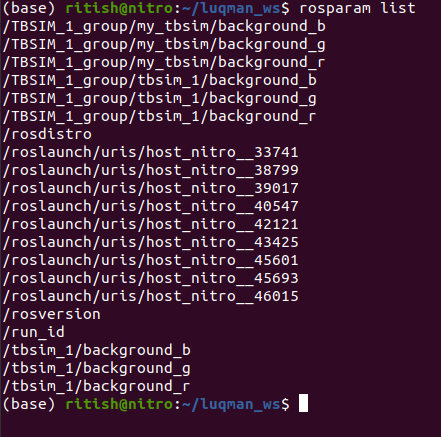
<arg name="" default=""/>

Arguments are basically variables that hold the value. Same is the case with parameters, but what is the major difference?

When we run a launch file with arguments, it changes the value of what was set before, but this can be done only once in the program. If a value has been set it cannot be changed until the program has stopped.

The value of a parameter can be changed dynamically i.e. a same parameter can be changed multiple times in a program.

Go to the terminal and type ‘*rosparam list*’.



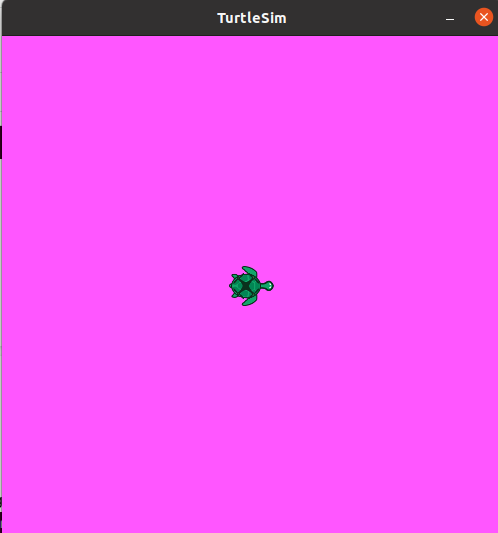
The value of a rosparam can be obtained and changed using the following commands



In launch file it can be changed using following syntax:

<param name="/tbsim\_1/background\_r" value="255"/>

Simulation after adding that line would look like:



A new rosparam can be created using the following syntax:

<rosparam param=""></rosparam>